

FLASHCARDS

Spring Boot

231 tarjetas de repaso rápido

frente = pregunta, dorso = respuesta · numeradas · 13 subtemas



ESTRELLA = pregunta clave de examen

Spring Boot · 231

El abajo a la izquierda; el mazo viene ordenado de lo BÁSICO a lo AVANZADO.

CÓMO USARLO

1 · IMPRIMIR

A4 a doble cara, volteando por el BORDE LARGO, escala 100% (sin "ajustar a página"). Corta por las líneas discontinuas: cada pregunta queda con su respuesta al dorso, verificable por la numeración.

2 · CELULAR

Abre flashcards.html en el navegador (sin internet): toca para voltear, marca «La sé» o «Repasar» y el progreso queda guardado. Incluye búsqueda, filtros por subtema, modo aleatorio y lista completa.

3 · ANKI / QUIZLET

Importa cards.csv (Frente ; Reverso ; Categoría ; Subtema) para estudiar con repetición espaciada.

MUESTRA A ESCALA REDUCIDA

FRENTE (página impar)

REVERSO (página par, espejado)

SPRING BOOT

S005

¿Qué son las interfaces @Enable* ?

SPRING BOOT

S005

RESPUESTA

Interruptores que importan configuración: @EnableAsync, @EnableScheduling, @EnableCaching, @EnableTransactionManagement, @EnableFeignClients...
Internamente usan @Import para registrar beans/proxies necesarios.
En Spring Boot muchas vienen activadas por autoconfiguración.

Al imprimir a doble cara, cada respuesta queda exactamente detrás de su pregunta; la numeración lo verifica.

Contenido: 231 tarjetas + chuleta exprés al dorso de esta portada · 60 páginas · generado 12/09/2026

CHULETA EXPRÉS

Lo básico que siempre se pregunta: HTTP, seguridad, testing, datos, Docker/K8s y GCP.

- Verbos HTTP: GET leer · POST crear · PUT reemplaza · PATCH parcial · DELETE borra. Idempotentes: GET/PUT/DELETE. Seguro: GET.
- Códigos HTTP: 200 OK · 201 Created (+Location) · 204 sin body · 400 validación · 401 no autenticado · 403 sin permiso · 404 no existe · 409 conflicto · 429 rate limit · 500 bug · 503 caído.
- ACID: Atomicidad, Consistencia, Aislamiento, Durabilidad. CAP: ante partición (P) eliges consistencia (C) o disponibilidad (A).
- Escalera REST (Richardson): 0 RPC/POX · 1 recursos · 2 verbos + códigos HTTP · 3 HATEOAS.
- JWT: header.payload.signature; firma HS256 (simétrica) o RS256 (asimétrica). Claims clave: sub, iss, aud, exp, iat. Nunca guardar datos sensibles: es legible (base64).
- OAuth2: Authorization Code + PKCE para apps/SPA · Client Credentials para servicio-a-servicio · Refresh Token para renovar sin re-login.
- SQL rápido: índices B-tree aceleran WHERE/JOIN · EXPLAIN para ver el plan · N+1 se resuelve con JOIN FETCH / batch · paginar con keyset, no solo OFFSET.
- Concurrencia en BD: optimista = @Version + retry (mejor para poca contención) · pesimista = SELECT FOR UPDATE (contención alta).
- Pool de conexiones: HikariCP (default en Boot). Ajustar maximumPoolSize (10-20 suele bastar) y connectionTimeout.
- Caché HTTP: Cache-Control (max-age, no-cache) + ETag/If-None-Match → 304 Not Modified. Caché de app: Spring @Cacheable.
- JUnit 5: @Test, @BeforeEach, AssertJ assertThat. Mockito: @Mock + @InjectMocks. Tipos de test Spring: @WebMvcTest (capa web), @DataJpaTest (repositorios), @SpringBootTest (integración completa).
- TDD: rojo (test que falla) → verde (código mínimo) → refactor. Pirámide: muchos unit, algunos integration, pocos E2E.
- SOLID: S una razón de cambio por clase · O abierto a extensión, cerrado a modificación · L subtipos sustituibles · I interfaces pequeñas · D depender de abstracciones.
- GoF que salen siempre: Singleton, Factory Method, Builder, Strategy, Adapter, Observer, Decorator, Proxy (Spring AOP usa proxies dinámicos).
- Maven: coordenadas groupId:artifactId:version · scopes: compile, provided, runtime, test · BOM para alinear versiones · mvnw wrapper para versionar Maven.
- Docker: imagen inmutable por capas con cache · multi-stage build para imagen liviana · no correr como root · ENTRYPOINT vs CMD · healthcheck.
- Kubernetes: Pod (unidad mínima) · Deployment (réplicas + rolling update) · Service (IP estable/DNS) · Ingress (rutas HTTP) · HPA (autoescalado) · ConfigMap/Secret (config).
- Logging: niveles TRACE < DEBUG < INFO < WARN < ERROR. Estructurado (JSON) y con correlation ID. Nunca datos personales ni secretos en logs.
- CORS: el navegador lanza preflight OPTIONS; el servidor responde con Access-Control-Allow-Origin/Methods/Headers. Es un mecanismo del navegador, no del servidor.
- Git/entrega: feature branches + Pull Request · squash merge · Conventional Commits (feat/fix/chore) · SemVer MAYOR.MINOR.PATCH (mayor = rompe compatibilidad).
- Rate limiting: 429 Too Many Requests + header Retry-After. Algoritmos: token bucket, sliding window. Se aplica en gateway.
- Async en Spring: @Async + @EnableAsync (executor), @EventListener para eventos en proceso, CompletableFuture para composición.
- Programación reactiva (WebFlux): Mono (0..1) y Flux (0..N), backpressure, no bloquea hilos; útil con mucha I/O concurrente. MVC por hilos sigue siendo válido para CRUD normal.
- Microservicios: contrato primero (OpenAPI), consumer-driven contract tests (Pact), CI/CD por servicio, feature flags para desacoplar deploy de release.
- En la nube: 12-factor (config en entorno, procesos stateless, logs como stream), probes de salud, escalado horizontal primero, idempotencia en todo endpoint que escribe.
- K8s probes: liveness reinicia el pod, readiness lo saca del balanceo, startup da margen a arranques lentos. Requests/limits de CPU-memoria definen la calidad de servicio.
- JOINs SQL: INNER (solo coincidencias) · LEFT (toda la izquierda + coincidencias) · RIGHT · FULL. Subconsultas correlacionadas fila a fila = sospechar de N+1.
- HTTP/2: multiplexa streams en una conexión, comprime headers (HPACK) y permite prioridades; gRPC exige HTTP/2. HTTP/3 va sobre QUIC/UDP.
- Complejidad: HashMap get/put O(1) medio · búsqueda binaria O(log n) · ordenar O(n log n) · recorrer O(n). ArrayList contains O(n).
- Colecciones thread-safe: ConcurrentHashMap (no bloquea el mapa entero) · CopyOnWriteArrayList (muchas lecturas) · BlockingQueue (productor/consumidor).
- Bean Validation: @Valid en el parámetro + @NotNull/@Size/@Email/@Pattern en el DTO; si falla, MethodArgumentNotValidException → 400 con detalles.
- Clientes REST en Spring: RestClient (síncrono, moderno) · WebClient (reactivo) · OpenFeign (declarativo por interfaz). Configura siempre timeout y retry.
- Docker Compose levanta el stack local (app + BD + cola) con docker compose up; perfiles para servicios opcionales. En prod: Kubernetes o Cloud Run.
- GCP rápido: proyecto = unidad de facturación/IAM · región (multi-zona) vs zona · Cloud Run = contenedor serverless · Pub/Sub = at-least-once · IAM = least privilege · ADC para autenticar apps sin claves.

SPRING BOOT ★

Núcleo IoC

S001

¿Qué son IoC y la inyección de dependencias?

BASICO

SPRING BOOT ★

Núcleo IoC

S002

@Component vs @Service vs @Repository vs @Controller

BASICO

SPRING BOOT ★

Núcleo IoC

S003

¿Por qué inyectar por constructor y no por campo?

BASICO

SPRING BOOT

Núcleo IoC

S004

@Bean vs @Component

BASICO

SPRING BOOT

Núcleo IoC

S005

¿Qué son las interfaces @Enable* ?

BASICO

SPRING BOOT

Núcleo IoC

S006

Component scan: paquetes base y filtros

BASICO

SPRING BOOT

Núcleo IoC

S007

Bean con nombre: convenciones y referencia por nombre

BASICO

SPRING BOOT ★

Núcleo IoC

S008

Scopes de beans en Spring

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT



Núcleo IoC

S002

RESPUESTA

Son estereotipos: todas registran un bean.
 @Component: genérico.
 @Service: lógica de negocio (semántica).
 @Repository: traduce excepciones de persistencia a
 DataAccessException.
 @Controller: capa web. @RestController = @Controller +
 @ResponseBody (JSON).

BASICO

SPRING BOOT



Núcleo IoC

S001

RESPUESTA

IoC: el contenedor (ApplicationContext) crea, cablea y gestiona el
 ciclo de vida de los beans; tú no haces new.
 DI: las dependencias llegan inyectadas (constructor > setter >
 campo).
 Beneficios: bajo acoplamiento, testeo con mocks, cambiar
 implementaciones fácil.

BASICO

SPRING BOOT

Núcleo IoC

S004

RESPUESTA

@Component: sobre TU clase; se registra por escaneo de
 componentes.
 @Bean: método dentro de @Configuration que DEVUELVE la
 instancia; para clases de terceros que no puedes anotar.
 @Configuration con proxy de métodos reutiliza llamadas a
 @Bean (respeta singleton).

BASICO

SPRING BOOT



Núcleo IoC

S003

RESPUESTA

Constructor:
 • dependencias obligatorias explícitas
 • campos pueden ser final
 • testeable sin Spring (new + mocks)
 • detecta dependencias circulares al arrancar.
 @Autowired en campo: oculta dependencias y exige reflejo en
 tests. Spring recomienda constructor.

BASICO

SPRING BOOT

Núcleo IoC

S006

RESPUESTA

@SpringBootApplication escanea desde SU paquete hacia
 abajo — clase main mal ubicada = beans no encontrados (error
 #1 de principiantes).
 Personalizar: @ComponentScan(basePackages, includeFilters,
 excludeFilters).
 Beans fuera del paquete base: registrarlos con @Import o @Bean.

BASICO

SPRING BOOT

Núcleo IoC

S005

RESPUESTA

Interruptores que importan configuración: @EnableAsync,
 @EnableScheduling, @EnableCaching,
 @EnableTransactionManagement, @EnableFeignClients...
 Internamente usan @Import para registrar beans/proxies
 necesarios.
 En Spring Boot muchas vienen activadas por autoconfiguración.

BASICO

SPRING BOOT



Núcleo IoC

S008

RESPUESTA

singleton (default): una instancia por contenedor.
 prototype: nueva por cada petición/inyección.
 request, session, websocket: por contexto web.
 Trampa: inyectar prototype dentro de un singleton se resuelve
 UNA vez → usar ObjectProvider o @Lookup.

INTERMEDIO

SPRING BOOT

Núcleo IoC

S007

RESPUESTA

Nombre default: primera letra minúscula del nombre de clase
 (userService) o del método @Bean.
 Custom: @Component("miNombre") / @Bean("miNombre").
 Inyectar por nombre: @Qualifier o campo cuyo nombre
 coincida — cuando tipo y nombre coinciden no hace falta
 qualifier.

BASICO

SPRING BOOT

★

Núcleo IoC

S009

Ciclo de vida de un bean

INTERMEDIO

SPRING BOOT

Núcleo IoC

S010

BeanFactory vs ApplicationContext

INTERMEDIO

SPRING BOOT

Núcleo IoC

S011

@Primary y @Qualifier: resoluciones de ambigüedad

INTERMEDIO

SPRING BOOT

Núcleo IoC

S012

@Lazy: qué hace y sus dos usos clave

INTERMEDIO

SPRING BOOT

Núcleo IoC

S013

@DependsOn, @Order y @PostConstruct/@PreDestroy

INTERMEDIO

SPRING BOOT

Núcleo IoC

S014

¿Cómo resuelve Spring las dependencias circulares?

AVANZADO

SPRING BOOT

Núcleo IoC

S015

BeanPostProcessor vs BeanFactoryPostProcessor

AVANZADO

SPRING BOOT

Núcleo IoC

S016

FactoryBean: qué es y cuándo aparece

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Núcleo IoC

S010

RESPUESTA

BeanFactory: contenedor básico, crea beans perezosamente (al pedirlos).
 ApplicationContext: la superinterfaz que usan las apps — añade eventos, internacionalización, recursos, y beans singleton se crean AL ARRANCAR (eager).
 En Spring Boot obtienes un ApplicationContext automáticamente.

INTERMEDIO

SPRING BOOT



Núcleo IoC

S009

RESPUESTA

Instanciación → inyección de dependencias → callbacks Aware → BeanPostProcessor (before) → @PostConstruct → afterPropertiesSet → LISTO → al cerrar: @PreDestroy.
 Los BeanPostProcessors envuelven los beans: así funcionan @Transactional, @Async, @Lazy.

INTERMEDIO

SPRING BOOT

Núcleo IoC

S012

RESPUESTA

1) Diferir la creación del bean hasta su primer uso (bean caro o poco usado).
 2) ROMPER dependencias circulares: inyectando un proxy perezoso, ambos beans se crean sin bloquearse.
 Trampa: los errores de configuración aparecen tarde (en runtime, no al arrancar).

INTERMEDIO

SPRING BOOT

Núcleo IoC

S011

RESPUESTA

Cuando 2+ beans implementan la misma interfaz, la inyección por tipo es ambigua → NoUniqueBeanDefinitionException.
 @Primary: el 'por defecto'.
 @Qualifier("nombre"): elige explícito en el punto de inyección.
 Alternativa: inyectar Map<String, MiServicio> (todos por nombre) o List<MiServicio> (todos en orden @Order).

INTERMEDIO

SPRING BOOT

Núcleo IoC

S014

RESPUESTA

3 niveles de caché (singletonFactories, earlySingletonReferences, singletons): expone una referencia TEMPRANA del bean mientras se crea el otro.
 Solo funciona con inyección por SETTER/campo y scopes singleton; con inyección por CONSTRUCTOR falla al arrancar (y está bien: es un smell).
 Fix moderno: @Lazy en una dependencia o rediseñar (extraer el código común).

AVANZADO

SPRING BOOT

Núcleo IoC

S013

RESPUESTA

@DependsOn("beanA"): fuerza orden de creación sin dependencia de tipo (side effects, caches de arranque).
 @Order/@Ordered: orden relativo en colecciones inyectadas, interceptores y filtros.
 @PostConstruct: init tras inyección; @PreDestroy: cleanup al cerrar contexto (solo singletons; prototype no gestiona destrucción).

INTERMEDIO

SPRING BOOT

Núcleo IoC

S016

RESPUESTA

Un bean que FABRICA otro tipo de objeto: getObject() devuelve la instancia real, getObjectType su clase, isSingleton su alcance.
 Uso: creación compleja de objetos de terceros (MyBatis SqlSessionFactoryBean, feign clients).
 Referencia con & prefijo obtiene la fábrica misma: context.getBean("&miFactoryBean").

AVANZADO

SPRING BOOT

Núcleo IoC

S015

RESPUESTA

BFPP: opera sobre DEFINICIONES de beans ANTES de instanciarlos (modificar propiedades, registrar más beans).
 BPP: envuelve cada bean DESPUÉS de instanciar (postProcessBeforeInitialization/After) — aquí nacen los proxies de @Transactional, @Async, @Validated, @Lazy.
 Es la extensibilidad del contenedor.

AVANZADO

SPRING BOOT

Núcleo IoC

S017

@Lookup y ObjectProvider: inyección de prototype en singleton

AVANZADO

SPRING BOOT

Núcleo IoC

S018

Interfaces Aware: qué inyectan

AVANZADO

SPRING BOOT

Núcleo IoC

S019

¿Cómo registras beans SIN anotaciones ni escaneo?

AVANZADO

SPRING BOOT

★ Configuración

S020

application.properties, YAML y profiles

BASICO

SPRING BOOT

★ Configuración

S021

@ConfigurationProperties vs @Value: cuál y por qué

BASICO

SPRING BOOT

Configuración

S022

Relaxed binding: ¿app.max-connections mapea a qué campo?

BASICO

SPRING BOOT

Configuración

S023

Orden de precedencia de la configuración en Spring Boot

BASICO

SPRING BOOT

Configuración

S024

¿Cómo validar que la configuración externa sea correcta al arrancar?

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Núcleo IoC

S018

RESPUESTA

Callbacks para 'conciencia' del entorno: BeanNameAware (nombre del bean), ApplicationContextAware (el contexto), EnvironmentAware, ApplicationEventPublisherAware, BeanFactoryAware.

Se llaman tras la inyección de dependencias, antes de @PostConstruct.

Hoy prefieres inyección directa o implementar solo lo necesario.

AVANZADO

SPRING BOOT

Núcleo IoC

S017

RESPUESTA

Problema: un singleton con dependencia prototype la recibe UNA vez.

@Lookup: método abstracto que Spring sobrescribe para dar una instancia nueva por llamada.

ObjectProvider<T>: provider.getObject() por petición, getIfAvailable/getIfUnique — más moderno y testeable.

AVANZADO

SPRING BOOT



Configuración

S020

RESPUESTA

Config externa en application.properties / .yml.

Profiles: application-prod.yml + spring.profiles.active=prod; activan beans con @Profile.

Precedencia: variables de entorno y args de CLI > perfiles > application.yml base.

BASICO

SPRING BOOT

Núcleo IoC

S019

RESPUESTA

- 1) @Configuration + métodos @Bean.
- 2) ImportSelector / ImportBeanDefinitionRegistrar (mecanismo de @Import — así funcionan @Enable*).
- 3) BeanDefinitionRegistryPostProcessor programático.
- 4) spring.factories / AutoConfiguration.imports para autoconfigs.

Es lo que hacen las librerías para registrar sus beans.

AVANZADO

SPRING BOOT

Configuración

S022

RESPUESTA

A maxConnections. Formatos equivalentes: camelCase (maxConnections), kebab-case (max-connections, recomendado en YAML), snake_case (MAX_CONNECTIONS en env vars), UPPER_CASE env var APP_MAXCONNECTIONS.

Listas: app.servers[0].host=...

Es el puente entre propiedades y variables de entorno de K8s/Cloud Run.

BASICO

SPRING BOOT



Configuración

S021

RESPUESTA

@ConfigurationProperties(prefix="app"): POJO con muchos valores agrupados, type-safe, IDE-friendly, validable con @Validated+JSR-303, relaxed binding (app.max-retry → maxRetry).

@Value("\${clave}"): valores sueltos y rápidos, con SpEL, pero disperso y sin metadatos.

Regla: grupos de config → @ConfigurationProperties; un dato puntual → @Value.

BASICO

SPRING BOOT

Configuración

S024

RESPUESTA

@ConfigurationProperties + @Validated + anotaciones JSR-303 en los campos (@NotNull, @Min, @URL).

Configuración inválida → falla el ARRANQUE con mensaje claro, no un NPE a las 3 a.m.

Alternativa: checks manuales en @PostConstruct o fail-fast con ApplicationContextRunner en tests.

BASICO

SPRING BOOT

Configuración

S023

RESPUESTA

De mayor a menor: argumentos CLI (--server.port=9090) > atributos SPRING_APPLICATION_JSON > variables de entorno (SERVER_PORT) > application-{profile} > application.yml/properties (externo > interno al jar) > @PropertySource > defaults.

Dev/test sobrescriben sin recompilar.

BASICO

SPRING BOOT

Configuración

S025

Placeholders, defaults y referencias entre propiedades

BASICO

SPRING BOOT

Configuración

S026

@Profile: activar beans por entorno

BASICO

SPRING BOOT

Configuración

S027

spring.config.import y multi-documento YAML

BASICO

SPRING BOOT

Configuración

S028

@PropertySource y conversiones de tipos custom

BASICO

SPRING BOOT

Configuración

S029

SpEL (Spring Expression Language): qué permite

INTERMEDIO

SPRING BOOT

★

Boot y starters

S030

¿Qué es Spring Boot y qué aporta?

BASICO

SPRING BOOT

Boot y starters

S031

¿Qué es un starter y cuáles usas siempre?

BASICO

SPRING BOOT

Boot y starters

S032

Servidor embebido: configuración y swap

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Configuración

S026

RESPUESTA

@Profile("prod") / @Profile({"dev","test"}) / @Profile("!prod") en @Component o @Bean.

Activar: spring.profiles.active=prod (propiedad, env var SPRING_PROFILES_ACTIVE, o CLI).

Combinar con application-{profile}.yml para configuración por entorno.

Un perfil default existe si no se activa ninguno.

BASICO

SPRING BOOT

Configuración

S025

RESPUESTA

@Value("\${app.timeout:30}") — default tras los ':'.

Referencias: app.url=http://\${app.host}:\${app.port}/api.

Variables especiales: \${random.value}, \${random.int[1024,65536]} (puertos de test).

SpEL en @Value: #{systemProperties['x']}.

BASICO

SPRING BOOT

Configuración

S028

RESPUESTA

@PropertySource("classpath:app.properties") añade fuentes al Environment (no soporta YAML directamente).

Conversiones: Spring usa ConversionService — String→enum, duraciones ("30s" → Duration), tamaños ("10MB" → DataSize) automáticamente en @ConfigurationProperties; registrar CustomConverter para tipos propios.

BASICO

SPRING BOOT

Configuración

S027

RESPUESTA

spring.config.import=optional:file:env[.properties], classpath:secrets.yml — importa configs adicionales (secrets, docker).

YAML multi-documento: '---' separa bloques, cada uno con spring.config.activate.on-profile — un solo archivo para todos los perfiles.

Orden: el último gana.

BASICO

SPRING BOOT



Boot y starters

S030

RESPUESTA

Capa opinionada sobre Spring que elimina la configuración manual:

- Autoconfiguración según el classpath
- Starters (spring-boot-starter-web)
- Servidor embebido (Tomcat/Jetty)
- Actuador listo para producción

@SpringBootApplication = @Configuration +

@EnableAutoConfiguration + @ComponentScan.

BASICO

SPRING BOOT

Configuración

S029

RESPUESTA

Lenguaje de expresiones para valores dinámicos en tiempo de ejecución:

@Value("#{systemProperties['user.name']}") — con #{...} (SpEL) vs \${...} (properties).

Operadores, llamadas a métodos, colecciones: #{lista.[edad > 18]} (selección), #{objeto.nombre?.toUpperCase()}.

Usos: @PreAuthorize("#id == principal.id"), @Cacheable(key="#id"), @EventListener(condition="...").

Poderoso pero NO metas lógica de negocio en SpEL.

INTERMEDIO

SPRING BOOT

Boot y starters

S032

RESPUESTA

Tomcat por defecto; starter-web-reactor trae Netty (WebFlux), y puedes cambiar a Jetty/Undertow excluyendo starter-tomcat.

Config: server.port, server.servlet.context-path, server.tomcat.threads.max, graceful shutdown (server.shutdown=graceful).

Jar ejecutable con java -jar — sin instalar servlet container.

BASICO

SPRING BOOT

Boot y starters

S031

RESPUESTA

Dependencia 'todo en uno' que agrupa librerías compatibles: spring-boot-starter-web (MVC+Tomcat+Jackson+validación), starter-data-jpa, starter-security, starter-actuator, starter-test, starter-validation.

Eliminan versiones incompatibles: el BOM de Boot gobierna versiones transitivas.

BASICO

SPRING BOOT ★

Boot y starters

S033

¿Qué hace @SpringBootApplication exactamente?

BASICO

SPRING BOOT

Boot y starters

S034

Anatomía de una autoconfiguración (condiciones)

BASICO

SPRING BOOT

Boot y starters

S035

Cómo depurar la autoconfiguración (por qué se activó X)

BASICO

SPRING BOOT

Boot y starters

S036

Clase main: SpringApplication y run() qué devuelve

BASICO

SPRING BOOT

Boot y starters

S037

Baner, nombres de app y metadata útil

BASICO

SPRING BOOT

Boot y starters

S038

Excluir autoconfiguración: las 3 formas

BASICO

SPRING BOOT

Boot y starters

S039

Propiedades comunes de arranque que conviene saber de memoria

BASICO

SPRING BOOT

Boot y starters

S040

Spring Boot DevTools: qué aporta en desarrollo

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Boot y starters

S034

RESPUESTA

@AutoConfiguration +
 @ConditionalOnClass(DataSource.class) — si la clase está en el classpath, activa.
 @ConditionalOnMissingBean: si TÚ ya definiste uno, respeta el tuyo.
 @ConditionalOnProperty(prefix="spring.datasource", name="url").
 Listadas en META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports.

BASICO

SPRING BOOT



Boot y starters

S033

RESPUESTA

Agrupar 3: @SpringBootConfiguration (es una @Configuration), @EnableAutoConfiguration y @ComponentScan (paquete actual hacia abajo).
 Atributos útiles: exclude=DataSourceAutoConfiguration.class (quita autoconfigs), scanBasePackages (mover el escaneo).
 Debe vivir en el paquete raíz.

BASICO

SPRING BOOT

Boot y starters

S036

RESPUESTA

ApplicationContext context = SpringApplication.run(App.class, args).
 Obtener beans: context.getBean(...), publicar eventos, perfiles activos.
 Customizar: new SpringApplicationBuilder(App.class).bannerMode(OFF).profiles("x").run(args).
 CommandLineRunner/ApplicationRunner: código de arranque post-contexto (reciben args crudos/parsed).

BASICO

SPRING BOOT

Boot y starters

S035

RESPUESTA

--debug o debug=true → CONDITION EVALUATION REPORT al arrancar: cada autoconfig con positivo/negativo y POR QUÉ (conditional on class, missing bean, property).
 También: actuador/conditions endpoint.
 Es el primer paso ante 'no se configura mi DataSource/conector'.

BASICO

SPRING BOOT

Boot y starters

S038

RESPUESTA

1) @SpringBootApplication(exclude = {DataSourceAutoConfiguration.class}).
 2) spring.autoconfigure.exclude=... (propiedad, soporta lista).
 3) Definir TÚ el bean: @ConditionalOnMissingBean cede (no es exclusión, pero anula el efecto).
 Caso típico: BD en memoria de Boot cuando la quieres manual.

BASICO

SPRING BOOT

Boot y starters

S037

RESPUESTA

spring.application.name: identificador en logs/tracing/eureka.
 banner.txt custom o off.
 build-info: spring-boot-maven-plugin con build-info → /actuador/info muestra versión y timestamps.
 Los metadatos de propiedades (additional-spring-configuration-metadata.json) dan autocomplete y documentación en el IDE.

BASICO

SPRING BOOT

Boot y starters

S040

RESPUESTA

Reinicio RÁPIDO al guardar (restart con classloader separado: tus clases recargan, las libs no → segundos vs minutos).
 LiveReload integrado para frontends.
 Deshabilita cachés por defecto en dev.
 Se desactiva SOLO al empaquetar (jar de producción no lo lleva).
 spring-boot-devtools en scope runtime (optional).

BASICO

SPRING BOOT

Boot y starters

S039

RESPUESTA

server.port=0 (puerto aleatorio para tests), spring.profiles.active,
 spring.datasource.url/username/password,
 spring.jpa.hibernate.ddl-auto (none/validate/update/create),
 spring.jpa.show-sql, management.endpoints...,
 logging.level.paquete=DEBUG,
 spring.servlet.multipart.max-file-size.
 En YAML con kebab-case.

BASICO

SPRING BOOT ★

Boot y starters

S041

¿Cómo funciona la autoconfiguración?

INTERMEDIO

SPRING BOOT ★

Web MVC y REST

S042

@Controller vs @RestController

BASICO

SPRING BOOT ★

Web MVC y REST

S043

**@PathVariable vs @RequestParam vs
@RequestBody**

BASICO

SPRING BOOT ★

Web MVC y REST

S044

**¿Cómo manejas excepciones globalmente
en una API?**

BASICO

SPRING BOOT

Web MVC y REST

S045

**Anotaciones de mapeo: combinadas y
específicas**

BASICO

SPRING BOOT

Web MVC y REST

S046

ResponseEntity: cuándo y cómo

BASICO

SPRING BOOT ★

Web MVC y REST

S047

**Jackson: anotaciones que resuelven el 90%
de los casos**

BASICO

SPRING BOOT ★

Web MVC y REST

S048

**¿Por qué NUNCA devolver entidades JPA en
un controller?**

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT



Web MVC y REST

S042

RESPUESTA

@Controller: devuelve nombres de VISTA (HTML con Thymeleaf, etc.).

@RestController: añade @ResponseBody → cada método retorna el objeto que Spring serializa a JSON con `HttpMessageConverters` (Jackson).

Para APIs REST: siempre @RestController.

BASICO

SPRING BOOT



Boot y starters

S041

RESPUESTA

@EnableAutoConfiguration carga configuraciones candidatas que se activan SOLO si se cumplen condiciones:

@ConditionalOnClass, @ConditionalOnMissingBean, @ConditionalOnProperty.

Ej.: detecta H2 en el classpath → configura un DataSource.

Se excluye con `exclude=...` en `@SpringBootApplication`.

INTERMEDIO

SPRING BOOT



Web MVC y REST

S044

RESPUESTA

@RestControllerAdvice con métodos

@ExceptionHandler(Excepcion.class): mapean excepciones de toda la app a respuestas HTTP uniformes (status + cuerpo de error).

Spring 6 trae ProblemDetail (RFC 7807).

@ResponseStatus para casos simples.

BASICO

SPRING BOOT



Web MVC y REST

S043

RESPUESTA

@PathVariable: del path → GET /users/{id}

@RequestParam: query string → ?page=2&size=10 (defaultValue, required)

@RequestBody: deserializa el body (JSON → DTO) en POST/PUT.

@RequestHeader para cabeceras.

BASICO

SPRING BOOT

Web MVC y REST

S046

RESPUESTA

Control total de status + headers + body: `ResponseEntity.status(HttpStatus.CREATED).location(uri).body(dto)`.

Uso típico: 201 con Location en POST, 204 en DELETE, 304 con etag.

Alternativa moderna: devolver el DTO directo + @ResponseStatus, o ProblemDetail en excepciones.

BASICO

SPRING BOOT

Web MVC y REST

S045

RESPUESTA

@RequestMapping(method=...) → atajos: @GetMapping, @PostMapping, @PutMapping, @PatchMapping, @DeleteMapping.

Atributos: path/value, params (condición por query param), headers, produces/consumes (negociación).

Class-level + method-level se CONCATENAN los paths.

BASICO

SPRING BOOT



Web MVC y REST

S048

RESPUESTA

- 1) LazyInitializationException o SELECTs N+1 al serializar relaciones lazy.
- 2) Recursión infinita en relaciones bidireccionales.
- 3) Expone tu esquema de BD (campos sensibles, breaking changes).
- 4) Acopla API y dominio.

Fix: DTOs + MapStruct/constructor; relaciones fetcheadas explícitas.

BASICO

SPRING BOOT



Web MVC y REST

S047

RESPUESTA

@JsonProperty("nombre_json") renombra; @JsonIgnore oculta;

@JsonFormat(shape/pattern, timezone) para fechas;

@JsonInclude(NON_NULL) evita nulls; @JsonAlias acepta varios nombres al deserializar; @JsonCreator/@JsonProperty para constructores inmutables;

@JsonManagedReference/@JsonBackReference corta recursión en relaciones bidireccionales (o mejor: usa DTOs).

BASICO

SPRING BOOT

★

Web MVC y REST

S049

Validación: @Valid, @Validated y constraints

BASICO

SPRING BOOT

Web MVC y REST

S050

Interceptor vs Filter vs AOP: ¿cuál para cada caso?

BASICO

SPRING BOOT

Web MVC y REST

S051

HandlerInterceptor: implementación y registro

BASICO

SPRING BOOT

Web MVC y REST

S052

CORS: las 3 formas de configurarlo en Spring

BASICO

SPRING BOOT

Web MVC y REST

S053

Paginación y ordenación estándar en Spring

BASICO

SPRING BOOT

Web MVC y REST

S054

Upload de archivos (multipart) en Spring

BASICO

SPRING BOOT

Web MVC y REST

S055

Manejo de errores REST moderno: ProblemDetail

BASICO

SPRING BOOT

Web MVC y REST

S056

@ExceptionHandler jerarquía: quién gana

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Web MVC y REST

S050

RESPUESTA

Filter (Servlet): ANTES de que llegue a Spring — auth cruda, compresión, charset, wrapping de request.
 HandlerInterceptor: dentro de Spring MVC — preHandle/postHandle/afterCompletion, acceso a handler elegido: logging de requests, métricas, tenant resolution.
 AOP: capa de servicio/dominio — transacciones, caché, auditoría de negocio.
 Orden: Filters → DispatcherServlet → Interceptors → Controller → AOP.

BASICO

SPRING BOOT



Web MVC y REST

S049

RESPUESTA

@Valid en @RequestBody dispara Bean Validation (Hibernate Validator): @NotNull, @NotBlank, @Size, @Min/@Max, @Email, @Pattern, @Positive, @Past/Future.
 Fallo → MethodArgumentNotValidException → 400 con detalles (capturar en @ControllerAdvice).
 @Validated en clase: valida parámetros de métodos simples (y habilita groups).
 Custom: @Constraint + ConstraintValidator.

BASICO

SPRING BOOT

Web MVC y REST

S052

RESPUESTA

1) @CrossOrigin(origins, methods) por controller/método.
 2) WebMvcConfigurer.addCorsMappings: registry.addMapping("/api/**").allowedOrigins(...).allowedMethods(...).
 3) Con Security: http.cors(Customizer) + CorsConfigurationSource bean (este manda).
 Preflight OPTIONS: Spring responde automáticamente si el origen está permitido.

BASICO

SPRING BOOT

Web MVC y REST

S051

RESPUESTA

implements HandlerInterceptor { preHandle (puede abortar con false), postHandle (antes de renderizar body), afterCompletion (siempre, para cleanup).
 Registro: WebMvcConfigurer addInterceptors → registry.addInterceptor(miInterceptor).addPathPatterns("/api/**").excludePathPatterns("/actuator/**").

BASICO

SPRING BOOT

Web MVC y REST

S054

RESPUESTA

@PostMapping(consume = MediaType.MULTIPART_FORM_DATA_VALUE) con @RequestParam("file") MultipartFile file.
 Config: spring.servlet.multipart.max-file-size / max-request-size.
 Guardado: Files.copy(file.getInputStream(), destino) — nunca en memoria para grandes.
 Validar content-type y extensión (no confiar en el filename del cliente).

BASICO

SPRING BOOT

Web MVC y REST

S053

RESPUESTA

Método recibe Pageable (Spring resuelve ?page=0&size=20&sort=nombre,asc) → devuelve Page<Dto> (content + totalElements + totalPages).
 Customizar nombres de parámetros: spring.data.web.pageable.*.
 Límite duro para size (máx 100) — evita 'page size 1 millón'.
 En DTO de salida: exponer solo lo necesario o devolver Slice (¿hay más?) si no quieres COUNT.

BASICO

SPRING BOOT

Web MVC y REST

S056

RESPUESTA

El más ESPECÍFICO y más CERCANO gana: método del controller > @ControllerAdvice > handler por defecto (BasicErrorController).
 Varios advices: @Order decide.
 Patrón: un advice global con ExceptionHandler para BusinessException, ValidationException, AccessDenied, y un catch-all 500 que loguea con stack.

BASICO

SPRING BOOT

Web MVC y REST

S055

RESPUESTA

Spring 6+: @ExceptionHandler puede devolver ProblemDetail (RFC 9457): { type, title, status, detail, properties... }.
 Autoconfigurable: spring.mvc.problemdetails.enabled=true.
 Cliente recibe error MACHINE-READABLE y humano, estándar entre APIs.
 En @RestControllerAdvice mapear cada excepción de negocio a su status.

BASICO

SPRING BOOT

Web MVC y REST

S057

Clientes HTTP modernos: RestClient vs WebClient vs Feign

BASICO

SPRING BOOT

Web MVC y REST

S058

RestClient: ejemplo canónico

BASICO

SPRING BOOT

Web MVC y REST

S059

DTOs: mapeo manual vs MapStruct

BASICO

SPRING BOOT

Web MVC y REST

S060

Arquitectura en capas vs hexagonal dentro de un servicio Spring

BASICO

SPRING BOOT

Web MVC y REST

S061

¿Dónde va la lógica de negocio: controller, service o repository?

BASICO

SPRING BOOT

Web MVC y REST

S062

¿Cómo manejar ids en URIs y respuestas de creación?

BASICO

SPRING BOOT

Web MVC y REST

S063

Multipart + DTO combinados en un endpoint

BASICO

SPRING BOOT

Web MVC y REST

S064

Anatomía de una petición HTTP (lo básico que nunca está de más)

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Web MVC y REST

S058

RESPUESTA

RestClient rc = RestClient.create();
 Pedido p = rc.get().uri("http://svc/api/pedidos/{id}",
 id).retrieve().body(Pedido.class);
 POST: rc.post().uri(...).contentType(APPLICATION_JSON).body(dto)
 .retrieve().toBodilessEntity();
 Errores: .onStatus(HttpStatusCode::isError, (req,res) -> { throw
 new MiEx(); })
 Builder con baseUrl para reutilizar.

BASICO

SPRING BOOT

Web MVC y REST

S057

RESPUESTA

RestClient (Spring 6.1): API fluida SÍNCRONA, el reemplazo
 natural de RestTemplate.
 WebClient: reactivo (o síncrono bloqueando), necesario en
 WebFlux.
 RestTemplate: en mantenimiento, no para código nuevo.
 OpenFeign: interfaz declarativa + anotaciones — genera la
 implementación (común en microservicios con Spring Cloud).

BASICO

SPRING BOOT

Web MVC y REST

S060

RESPUESTA

Capas clásica: controller → service → repository, DTOs en la orilla.
 Simple y conocida.
 Hexagonal: dominio en el CENTRO sin dependencias de
 Spring/JPA; puertos (interfaces) + adapters (web, persistencia)
 inyectados.
 Beneficio hexagonal: dominio testeable sin levantar Spring, swap
 de infraestructura.
 Costo: más código; justificado en dominios complejos.

BASICO

SPRING BOOT

Web MVC y REST

S059

RESPUESTA

Manual: constructor/copiar a mano — explícito, cero magia,
 verboso.
 MapStruct: genera el mapper EN COMPILACIÓN (sin reflejos,
 rápido): @Mapper interface con métodos; mapea por nombres,
 customiza con @Mapping(source, target, ignore).
 ModelMapper: runtime + reflejo — cómodo pero silencioso ante
 refactor.
 Elige: MapStruct para proyectos serios.

BASICO

SPRING BOOT

Web MVC y REST

S062

RESPUESTA

POST → 201 + header Location: /api/pedidos/42 (URI absoluta o
 relativa); ServletUriComponentsBuilder.fromCurrentRequest().pat
 h("/{id}").buildAndExpand(dto.getId()).toUri().
 Body: el recurso creado (o solo id).
 Ids: no exponer secuenciales de BD si importan (UUID o ids
 opacos).

BASICO

SPRING BOOT

Web MVC y REST

S061

RESPUESTA

Controller: HTTP puro — validar formato, mapear, responder.
 Service: REGLAS DE NEGOCIO, orquestación, transaccionalidad.
 Repository: acceso a datos únicamente (queries), sin lógica.
 Anemia (services con if/else y entidades tontas) es común pero
 mejorable: lógica en entidades de dominio cuando pertenece a
 un solo agregado.

BASICO

SPRING BOOT

Web MVC y REST

S064

RESPUESTA

Request: MÉTODO (GET/POST/PUT...) + PATH (/api/users/1) +
 HEADERS (Content-Type, Authorization, Accept) + BODY
 opcional.
 Response: STATUS (200/404/500) + HEADERS + BODY.
 Sin estado: cada request se entiende sola; la 'sesión' viaja en
 tokens o cookies.
 En Spring: método=GetMapping, path=RequestMapping,
 headers=@RequestHeader, body=@RequestBody,
 status=ResponseEntity.

BASICO

SPRING BOOT

Web MVC y REST

S063

RESPUESTA

@RequestPart("datos") MiDto dto + @RequestPart("archivo")
 MultipartFile file — cada parte con su propio
 converter/validación (@Valid sobre la parte).
 Consumes MULTIPART_FORM_DATA. El cliente envía form-data
 con partes tipadas.
 Caso típico: metadata JSON + archivo.

BASICO

SPRING BOOT

★

Web MVC y REST

S065

Flujo de una petición en Spring MVC

INTERMEDIO

SPRING BOOT

Web MVC y REST

S066

@ModelAttribute y formularios clásicos vs @RequestBody

INTERMEDIO

SPRING BOOT

Web MVC y REST

S067

Versión de API: las 3 estrategias aplicadas en Spring

INTERMEDIO

SPRING BOOT

Web MVC y REST

S068

Documentar la API: OpenAPI/Swagger en Spring Boot 3

INTERMEDIO

SPRING BOOT

★

Web MVC y REST

S069

GlobalExceptionHandler: esqueleto que debes saber escribir

INTERMEDIO

SPRING BOOT

Web MVC y REST

S070

Content negotiation: cómo Spring elige JSON o XML

AVANZADO

SPRING BOOT

Web MVC y REST

S071

Grupos de validación: crear vs actualizar

AVANZADO

SPRING BOOT

Web MVC y REST

S072

ConstraintValidator: validación custom paso a paso

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Web MVC y REST

S066

RESPUESTA

@RequestBody: body JSON/XML → DTO (via `HttpMessageConverter`). APIs.

@ModelAttribute: binding de form-data/query a objeto con setters + validación — apps MVC con vistas, formularios HTML.

Errores de binding de form: `BindingResult` inmediatamente tras el parámetro para manejarlos sin excepción.

INTERMEDIO

SPRING BOOT



Web MVC y REST

S065

RESPUESTA

Request → `DispatcherServlet` (front controller) → `HandlerMapping` elige el método → interceptores → argumentos resueltos (@PathVariable, @RequestBody...) → tu método → `HttpMessageConverter` serializa → respuesta.

Excepciones → @ExceptionHandler / @ControllerAdvice.

INTERMEDIO

SPRING BOOT

Web MVC y REST

S068

RESPUESTA

springdoc-openapi-starter-webmvc-ui: expone /v3/api-docs (JSON OpenAPI) y /swagger-ui.html.

Enriquecer: @Operation(summary, description), @ApiResponse, @Schema en DTOs, @Tag por controller.

Exportar spec para contratos/gateways. Documentar es parte del contrato — pruebas de contrato la validan.

INTERMEDIO

SPRING BOOT

Web MVC y REST

S067

RESPUESTA

URI: @GetMapping("/api/v1/pedidos") + controllers por versión (clara, cacheable, la más común).

Header: @GetMapping(headers="X-API-Version=2") o Accept versionado — URL limpia pero menos visible.

Query param: ?version=2 (flexible, no estándar).

Regla: no romper contratos; versiones nuevas = código nuevo coexistiendo, deprecarse con aviso.

INTERMEDIO

SPRING BOOT

Web MVC y REST

S070

RESPUESTA

Por headers Accept (y Content-Type para el body de entrada) contra los produce/consume declarados y los `HttpMessageConverters` disponibles.

Jackson (JSON) viene con starter-web.

URL con sufijo (.json) está DESACTIVADO por defecto (seguridad).

Formato param: favor-parameter (raro).

AVANZADO

SPRING BOOT



Web MVC y REST

S069

RESPUESTA

@RestControllerAdvice class:

@ExceptionHandler(RecursoNoEncontrado.class) → 404 ProblemDetail.

@ExceptionHandler(MethodArgumentNotValidException.class) → 400 + lista de campos inválidos (ex.getBindingResult().getFieldErrors()).

@ExceptionHandler(AccesoDenegado.class) → 403.

@ExceptionHandler(Exception.class) → 500 + log.error con stack (sin exponer detalles internos al cliente).

INTERMEDIO

SPRING BOOT

Web MVC y REST

S072

RESPUESTA

1) @interface @Target(FIELD) @Constraint(validatedBy = EstadoValidator.class).

2) class EstadoValidator implements ConstraintValidator<MiAnon, String> { isValid(value, ctx) }.

3) ctx.buildConstraintViolationWithTemplate(msg) para mensajes dinámicos.

Se aplica como cualquier constraint; parametrizable con atributos de la anotación.

AVANZADO

SPRING BOOT

Web MVC y REST

S071

RESPUESTA

POST requiere algunos campos, PUT no (id autogenerado).

public interface OnCreate {} y @Null(groups=OnCreate.class) Long id.

@Validated(OnCreate.class) en el POST.

Alternativa moderna: DTOs separados por operación (CreateUserRequest/UpdateUserRequest) — más claro que grupos.

AVANZADO

SPRING BOOT

Web MVC y REST

S073

ETags y caché HTTP con Spring

AVANZADO

SPRING BOOT

Web MVC y REST

S074

Streaming de respuestas: grandes exportaciones

AVANZADO

SPRING BOOT

Web MVC y REST

S075

Async requests MVC: DeferredResult y Callable

AVANZADO

SPRING BOOT

Web MVC y REST

S076

HATEOAS: qué es y cuándo (no) usarlo

AVANZADO

SPRING BOOT ★

Spring Data JPA

S077

Query derivation: keywords que se preguntan

BASICO

SPRING BOOT ★

Spring Data JPA

S078

@Query: JPQL vs nativo y cuándo cada uno

BASICO

SPRING BOOT

Spring Data JPA

S079

Proyecciones: interface-based y class-based

BASICO

SPRING BOOT ★

Spring Data JPA

S080

save() vs saveAndFlush() vs merge

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Web MVC y REST

S074

RESPUESTA

SseEmitter (eventos), StreamingResponseBody (escribe en el OutputStream en otro hilo — no retiene el servlet thread con todo el body), ResponseEntity<Resource> con InputStreamResource para archivos.
Para exportación de reportes: escribir CSV/Excel por chunks directamente al stream.
Evita OOM al exportar millones de filas.

AVANZADO

SPRING BOOT

Web MVC y REST

S073

RESPUESTA

ResponseEntity con eTag + condicionales:
ResponseEntity.ok().eTag(version).body(dto); manejo de If-None-Match → 304.
ShallowETagHeaderFilter (hash del body) o eTag manual por versión de entidad (@Version).
Cache-Control header + CacheControl.maxAge(...).
Reduce ancho de banda; no sustituye caché de aplicación.

AVANZADO

SPRING BOOT

Web MVC y REST

S076

RESPUESTA

Hypermedia: la respuesta incluye links a acciones posibles (_links.self, .cancel) — el cliente navega sin conocer URIs fijas.
Spring HATEOAS: EntityModel/WebMvcLinkBuilder.
Realismo: rara vez se implementa completo (nivel 3 Richardson); valor real en APIs públicas longevas. Internamente, links simples bastan.

AVANZADO

SPRING BOOT

Web MVC y REST

S075

RESPUESTA

Devolver Callable<T> o DeferredResult<T> libera el servlet thread y responde cuando la tarea termina — más throughput con I/O lenta sin migrar a WebFlux.
Timeouts configurables; CompletableFuture es la base moderna.
StreamingResponseBody para escribir progresivamente.
WebFlux es otra escala (reactivo completo), no siempre necesario.

AVANZADO

SPRING BOOT



Spring Data JPA

S078

RESPUESTA

@Query("select p from Pedido p where p.total > :min") — JPQL contra ENTIDADES (portable, tipo-aware con @Param).
@Query(value="select * from pedidos...", nativeQuery=true) — SQL real: funciones de BD, hints, UNION; acopla al motor.
Retornos posibles: entidades, DTOs con constructor expression (select new com.x.Dto(p.id, p.total)), proyecciones de interfaz, scalars.

BASICO

SPRING BOOT



Spring Data JPA

S077

RESPUESTA

findByEmailAndActivo(email, bool),
findByNombreStartingWith/Containing/EndingWith, Between, IsNull/IsNotNull, In, OrderByEdadDesc (en el nombre), Top5/First3 (limit), findByEdadGreaterThanEqual.
Cuento: countBy...; existencia: existsBy...; borrado: deleteBy... (requiere @Transactional y carga entidades).
Si el nombre crece demasiado → @Query.

BASICO

SPRING BOOT



Spring Data JPA

S080

RESPUESTA

save: si es nueva → persist; si existe (no transient) → merge; el INSERT/UPDATE puede diferirse al flush.
saveAndFlush: fuerza el SQL YA (necesitas el id generado o triggers al instante).
merge: copia el estado de una entidad detached a una managed (devuelve la MANAGED — ignorar el retorno es un bug clásico).

BASICO

SPRING BOOT

Spring Data JPA

S079

RESPUESTA

Interface projection: interface NombreVista { String getNombre(); BigDecimal getTotal(); } — el repo devuelve List<NombreVista>;
Spring genera proxy que solo trae esas columnas (SQL optimizado).
Class/DTO projection: @Query con 'select new com.x.Dto(p.nombre, sum(p.total))'.
Dynamic: <T> List<T> findBy...(Class<T> tipo).
Evita cargar entidades para solo leer columnas.

BASICO

SPRING BOOT

★

Spring Data JPA

S081

Relaciones JPA: anotaciones y el lado dueño

BASICO

SPRING BOOT

★

Spring Data JPA

S082

Cascade: tipos y cuándo usarlos

BASICO

SPRING BOOT

Spring Data JPA

S083

@Embeddable y @Embedded: value objects en entidades

BASICO

SPRING BOOT

Spring Data JPA

S084

AttributeConverter: mapear enums/objetos custom

BASICO

SPRING BOOT

Spring Data JPA

S085

Repositorios múltiples y @NoRepositoryBean

BASICO

SPRING BOOT

★

Spring Data JPA

S086

¿Qué es JPA? ¿Y Hibernate? ¿Y Spring Data JPA?

INTERMEDIO

SPRING BOOT

★

Spring Data JPA

S087

Estados de una entidad JPA

INTERMEDIO

SPRING BOOT

★

Spring Data JPA

S088

¿Qué es LazyInitializationException y cómo se evita?

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT ★

Spring Data JPA

S082

RESPUESTA

PERSIST (guardar hijos con el padre), MERGE, REMOVE (borrar hijos con el padre — cuidado), ALL, REFRESH, DETACH.
Uso: ciclo de vida COMPLETAMENTE dependiente (Pedido→Items: ALL + orphanRemoval=true).
NUNCA cascade a entidades compartidas (no pongas ALL de Pedido→Cliente: borrar un pedido no debe borrar un cliente).

BASICO

SPRING BOOT ★

Spring Data JPA

S081

RESPUESTA

@OneToOne, @OneToMany/@ManyToOne, @ManyToMany.
El lado DUEÑO tiene la FK: @JoinColumn("cliente_id") en ManyToOne.
@OneToMany(mappedBy="cliente"): lado inverso (no dueño) — sin tabla intermedia propia.
En @ManyToMany ALWAYS solo un lado mappedBy + @JoinTable; cuida cascade y orphanRemoval.

BASICO

SPRING BOOT

Spring Data JPA

S084

RESPUESTA

@Converter(autoApply=true) class RolConverter implements AttributeConverter<Rol, String> { convertToDatabaseColumn / convertToEntityAttribute }.
Casos: enums por código legado, clases de valor (Dinero→2 columnas via Embeddable mejor), listas como JSON.
Aplica a todas las entidades sin repetir.

BASICO

SPRING BOOT

Spring Data JPA

S083

RESPUESTA

Clase @Embeddable Direccion (calle, ciudad, cp) embebida en @Embedded Direccion direccion → columnas en la MISMA tabla.
@AttributeOverrides renombra columnas si se repite.
Perfecto para value objects inmutables con equals/hashCode (dinero, dirección, rango de fechas).

BASICO

SPRING BOOT ★

Spring Data JPA

S086

RESPUESTA

JPA: especificación de ORM en Java (mapear entidades con @Entity, @Id, relaciones).
Hibernate: implementación de referencia.
Spring Data JPA: capa encima que genera repositorios (JpaRepository) y queries por nombre de método (findByEmail) o @Query.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S085

RESPUESTA

Base repository compartido: interface BaseRepo<T> extends JpaRepository<T, ID> con métodos comunes — anótala @NoRepositoryBean para que Spring NO la instancie.
Cada entidad extiende la base. También exist: fragmentos custom (interface custom + Impl) para lógica de acceso compartida.

BASICO

SPRING BOOT ★

Spring Data JPA

S088

RESPUESTA

Acceder a una relación LAZY cuando la sesión de Hibernate ya cerró.
Soluciones:
• JOIN FETCH en JPQL (@Query)
• @EntityGraph
• proyecciones a DTO
• transacción abierta en el servicio (evita depender de open-in-view).

INTERMEDIO

SPRING BOOT ★

Spring Data JPA

S087

RESPUESTA

Transient: nuevo, fuera del contexto.
Managed: attached — el dirty checking detecta cambios y los sincroniza en flush (sin llamar a save).
Detached: el contexto cerró; save() hace merge para re-adjuntar.
Removed: marcado para DELETE.

INTERMEDIO

SPRING BOOT

★

Spring Data JPA

S089

JOIN FETCH y @EntityGraph: resolver N+1

INTERMEDIO

SPRING BOOT

Spring Data JPA

S090

@Modifying y updates masivos

INTERMEDIO

SPRING BOOT

Spring Data JPA

S091

orphanRemoval vs CascadeType.REMOVE

INTERMEDIO

SPRING BOOT

Spring Data JPA

S092

Herencia en entidades: las 3 estrategias

INTERMEDIO

SPRING BOOT

Spring Data JPA

S093

Optimistic locking con @Version

INTERMEDIO

SPRING BOOT

Spring Data JPA

S094

Pessimistic locking: cuándo y cómo

INTERMEDIO

SPRING BOOT

Spring Data JPA

S095

Specifications (Criteria): queries dinámicas tipadas

INTERMEDIO

SPRING BOOT

Spring Data JPA

S096

Auditoría automática: @CreatedDate y compañía

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Spring Data JPA

S090

RESPUESTA

@Modifying + @Query("update Pedido p set p.estado = :e where p.creado < :fecha") — UPDATE/DELETE directo en BD.
 Requiere @Transactional.
 NO actualiza el contexto de persistencia: entidades en memoria quedan desactualizadas (clearAutomatically=true para forzar).
 Mucho más rápido que cargar+modificar miles de entidades.

INTERMEDIO

SPRING BOOT



Spring Data JPA

S089

RESPUESTA

N+1: 1 query de pedidos + N queries de cliente al tocar cada relación lazy.
 @Query("select p from Pedido p join fetch p.cliente where p.id in :ids") — fetch join trae la relación EAGER en ESA query.
 @EntityGraph(attributePaths = {"cliente", "items"}) sobre el método del repo — mismo efecto declarativo.
 Distinct + fetch de colecciones → in-memory dedup (o hibernate.default_batch_fetch_size).

INTERMEDIO

SPRING BOOT

Spring Data JPA

S092

RESPUESTA

@Inheritance(strategy=...):
 SINGLE_TABLE (default): una tabla + columna discriminador — rápida, columnas nullable.
 JOINED: tabla por clase unidas por FK — normalizada, joins en cada lectura.
 TABLE_PER_CLASS: tabla completa por clase concreta — uniones costosas, ids compartidos.
 Default razonable: SINGLE_TABLE.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S091

RESPUESTA

orphanRemoval=true: si QUITAS un hijo de la colección del padre, se DELETE de BD (los huérfanos se eliminan).
 CascadeType.REMOVE: borrar el PADRE borra a los hijos.
 Combinación estándar en composiciones:
 @OneToMany(mappedBy=..., orphanRemoval=true, cascade=ALL).
 Orphan removal es reemplazo de hijo:
 pedido.getItems().remove(item) → DELETE.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S094

RESPUESTA

Contención alta donde reintentar es caro (reservas, saldos):
 @Lock(LockModeType.PESSIMISTIC_WRITE) en el método del repo → SELECT ... FOR UPDATE (bloquea la fila hasta el commit).
 Con timeout: @QueryHints(@QueryHint(name="javax.persistence.lock.timeout", value="3000")).
 Cuidado: orden consistente de bloqueo para evitar deadlocks.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S093

RESPUESTA

Campo @Version Long version en la entidad: cada UPDATE incluye WHERE version = leída y la incrementa.
 Concurrencia: segundo hilo con versión vieja → OptimisticLockException (o ObjectOptimisticLockingFailureException en Spring).
 Sin locks de BD: ideal para baja contención.
 Reintentar la operación es responsabilidad de la app.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S096

RESPUESTA

@EnableJpaAuditing en config +
 @EntityListeners(AuditingEntityListener.class).
 @CreatedDate/@LastModifiedDate (fechas),
 @CreatedBy/@LastModifiedBy (necesitas AuditorAware<T> bean que devuelve el usuario actual).
 Campos auditables en un MappedSuperclass base para reusar en todas las entidades.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S095

RESPUESTA

extend JpaSpecificationExecutor<T>: findAll(Specification<T> spec, Pageable).
 Construir con Specification.where(A.and(B)): (root, query, cb) -> cb.equal(root.get("estado"), estado).
 Caso de uso: filtros de búsqueda contruidos por combinación (campos opcionales).
 Tipado con Metamodel para evitar strings.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S097

Batching: cómo hacer que inserts/updates masivos no sean 1 a 1

INTERMEDIO

SPRING BOOT



Spring Data JPA

S098

ddl-auto: valores y cuál en cada entorno

INTERMEDIO

SPRING BOOT



Spring Data JPA

S099

Flyway/Liquibase: migraciones versionadas

INTERMEDIO

SPRING BOOT

Spring Data JPA

S100

Dirty checking: cómo Hibernate detecta cambios sin save()

INTERMEDIO

SPRING BOOT

Spring Data JPA

S101

flush y el orden: queries antes del commit

INTERMEDIO

SPRING BOOT

Spring Data JPA

S102

Repositorio custom: interface + Impl (el patrón completo)

INTERMEDIO

SPRING BOOT

Spring Data JPA

S103

EntityManager vs repositorio: cuándo bajar al EntityManager

INTERMEDIO

SPRING BOOT



Spring Data JPA

S104

JdbcTemplate: para qué sirve y por qué sigue vivo

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT ★
RESPUESTA

Spring Data JPA

S098

create: borra y crea TODO al arrancar (solo dev).
 create-drop: crea al arrancar, borra al cerrar (tests).
 update: altera tablas existentes (NUNCA en prod: sin rollback ni historial).
 validate: solo verifica el esquema (prod recomendado con migraciones).
 Esquema real de prod: Flyway/Liquibase + validate.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S099

spring.jpa.properties.hibernate.jdbc.batch_size=50 +
 order_inserts=true + order_updates=true.
 REQUISITO con MySQL: rewriteBatchedStatements=true en la URL.
 OJO: IDs IDENTITY DESACTIVAN batching en inserts (usa SEQUENCE/SEQUENCE-style o asignados).
 Guardar en chunks de 50 con clear() periódico del contexto.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S100

Al cargar, Hibernate guarda snapshot del estado; al flush compara y genera UPDATE solo con columnas cambiadas.
 Consecuencia: dentro de una transacción, entidad cargada + setter = UPDATE automático (sin llamar save).
 Fuera de transacción (o detached) no ocurre — por eso la gente 'pierde' cambios.

INTERMEDIO

SPRING BOOT ★
RESPUESTA

Spring Data JPA

S099

Scripts SQL versionados en resources/db/migration:
 V1__crear_tabla.sql, V2__add_column.sql.
 Al arrancar, Flyway aplica los pendientes en orden y registra en flyway_schema_history.
 Beneficios: esquema reproducible, migraciones revisadas en PR, rollback claro (V_down o revert).
 Regla: cambiar entidad = nueva migración, nunca editar la vieja aplicada.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S102

```
interface PedidoRepoCustom { List<Pedido>
  buscarComplejo(Filtro f); }
class PedidoRepoCustomImpl implements PedidoRepoCustom {
  @PersistenceContext EntityManager em; ... } — nombre CLAVE:
  Impl + nombre de la interface.
interface PedidoRepo extends JpaRepository<Pedido,Long>,
  PedidoRepoCustom {} — Spring fusiona ambos.
Para SQL complejo y dinámico sin pelear con JPQL.
```

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S101

Flush = enviar SQL pendiente a BD (sin commit). Ocurre: en commit, en queries JPQL (para ver cambios pendientes, flush AUTO), y con flush() explícito.
 Trampa: query nativa NO dispara flush de la entidad (FlushModeType.AUTO no aplica igual) → leer datos viejos; usar entityManager.flush() antes.
 Commit = flush + transacción confirmada.

INTERMEDIO

SPRING BOOT ★
RESPUESTA

Spring Data JPA

S104

Acceso JDBC SIN el boilerplate de abrir/cerrar Connection, Statement y ResultSet — tú pones el SQL y los parámetros:
 jdbc.query("SELECT * FROM users WHERE id = ?", rowMapper, id);
 Ideal para: queries puntuales, reportes, batch SQL directo, apps sin JPA.
 Maneja recursos y excepciones (traduce SQLException a DataAccessException).
 Menos mágico que JPA: tú controlas cada SQL.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S103

Repositorio: 90% de casos CRUD/queries nombradas.
 EntityManager (@PersistenceContext): queries criteria dinámicas, flush/clear manuales en procesos masivos, detach para liberar memoria, batching con clear cada N, locks con hints, referencias lazy getReference.
 Regla: repositorio primero; EM cuando elORM necesita control fino.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S105

Spring Data JDBC vs Spring Data JPA:
cuándo cada uno

INTERMEDIO

SPRING BOOT

Transacciones

S106

@Transactional: ¿cómo funciona y qué
trampas tiene?

INTERMEDIO

SPRING BOOT

Transacciones

S107

Niveles de aislamiento de transacciones

INTERMEDIO

SPRING BOOT

Transacciones

S108

Transacciones y tests: @Transactional en
tests hace rollback

INTERMEDIO

SPRING BOOT ★

Transacciones

S109

Las 7 propagaciones de @Transactional
(cuáles y cuándo)

AVANZADO

SPRING BOOT

Transacciones

S110

readOnly=true: qué garantiza de verdad

AVANZADO

SPRING BOOT

Transacciones

S111

Timeout y trampas del rollback

AVANZADO

SPRING BOOT

Transacciones

S112

¿Qué pasa si @Transactional está en un
método private o se llama desde el mismo
bean?

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Transacciones

S106

RESPUESTA

AOP crea un proxy que abre/commit/rollback la transacción.
Rollback SOLO con RuntimeException por defecto (usa rollbackFor=Exception.class).

NO funciona: self-invocation (this.metodo()), métodos private/static.

Propagación: REQUIRED (une tx existente), REQUIRES_NEW (suspende y crea nueva), NESTED.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S105

RESPUESTA

Spring Data JDBC: ORM MÍNIMO — mapea agregados a tablas sin lazy loading, sin dirty checking, sin sesiones; tú escribes SQL simple. Menos sorpresas, más predecible.

Spring Data JPA (Hibernate): relaciones lazy, caché, dirty checking, JPQL — más features, más complejidad.

Regla: dominio simple con agregados planos → Data JDBC; dominio relacional rico → JPA.

INTERMEDIO

SPRING BOOT

Transacciones

S108

RESPUESTA

@Transactional en un test: cada test corre en tx que se revierte al final — tests aislados y rápidos SIN limpiar BD.

Trampas: no ves el commit real (problemas de constraints deferred, triggers de commit no se prueban), y con Testcontainers/threads separados NO comparten la tx (usa commit real ahí o TestTransaction).

TestTransaction framework para control manual.

INTERMEDIO

SPRING BOOT

Transacciones

S107

RESPUESTA

READ_UNCOMMITTED: lee datos sin confirmar (dirty read).

READ_COMMITTED: solo datos confirmados (default Postgres/Oracle).

REPEATABLE_READ: lecturas repetibles (default MySQL/InnoDB).

SERIALIZABLE: como ejecución en serie.

Más aislamiento → menos anomalías, más bloqueos.

INTERMEDIO

SPRING BOOT

Transacciones

S110

RESPUESTA

Hint al driver y al ORM: Hibernate desactiva dirty checking y flush → consultas mucho más rápidas en lecturas masivas.

En BD con soporte (Postgres SET TRANSACTION READ ONLY) puede impedir escrituras.

NO es una garantía de integridad por sí sola: úsalo en métodos de consulta como optimización + intención.

AVANZADO

SPRING BOOT



Transacciones

S109

RESPUESTA

REQUIRED (default): une o crea.

REQUIRES_NEW: suspende la actual y abre una NUEVA (logs de auditoría que deben persistir aunque falle el negocio).

NESTED: savepoint dentro de la actual (rollback parcial posible).

SUPPORTS: usa si existe.

NOT_SUPPORTED: suspende (código sin tx dentro de una).

MANDATORY: exige tx existente (servicio interno).

NEVER: prohíbe.

AVANZADO

SPRING BOOT

Transacciones

S112

RESPUESTA

NO hace nada (silenciosamente sin transacción):

1) private/static/final: el proxy CGLIB no puede sobrescribirlo.

2) Self-invocation: this.metodo() llama directo, SIN pasar por el proxy.

Fix: mover a otro bean, inyectarse a sí mismo (@Lazy), o TransactionTemplate programático.

Causa #1 de 'mi transacción no funciona'.

AVANZADO

SPRING BOOT

Transacciones

S111

RESPUESTA

@Transactional(timeout=30): rollback si excede (protege locks).

Rollback por defecto SOLO en RuntimeException/Error; checked exceptions NO revierten → rollbackFor = Exception.class si el negocio lanza checked.

noRollbackFor para excepciones 'esperables' que no deben revertir el trabajo ya hecho.

AVANZADO

SPRING BOOT

Transacciones

S113

TransactionTemplate: la alternativa programática

AVANZADO

SPRING BOOT

Transacciones

S114

@Transactional y eventos: post-commit

AVANZADO

SPRING BOOT

Transacciones

S115

¿Qué es el problema de dual-write y cómo se evita?

AVANZADO

SPRING BOOT

Transacciones

S116

Isolation level: anomalías concretas que resuelve cada nivel

AVANZADO

SPRING BOOT

Transacciones

S117

Locks de BD y transacciones largas: por qué son veneno

AVANZADO

SPRING BOOT

AOP

S118

Vocabulario AOP: aspect, join point, pointcut, advice, weaving

AVANZADO

SPRING BOOT

AOP

S119

Tipos de advice y cuál usar

AVANZADO

SPRING BOOT

AOP

S120

Pointcut expressions: los patrones que debes saber

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Transacciones

S114

RESPUESTA

@TransactionalEventListener(phase = AFTER_COMMIT): el listener corre SOLO si la tx confirmó — para notificaciones, emails, invalidar caché, publicar a otro sistema.
BEFORE_COMMIT, AFTER_ROLLBACK, AFTER_COMPLETION.
@EventListener normal corre DENTRO de la misma tx (verá datos sin confirmar).
Clave para no enviar emails de pedidos que luego fallaron.

AVANZADO

SPRING BOOT

Transacciones

S113

RESPUESTA

transactionTemplate.execute(status -> { ...; return resultado; });
Rollback: status.setRollbackOnly() o excepción.
Ventajas: control explícito, transacciones PARCIALES dentro de un método (sin mover código a otro bean), imposible olvidar el proxy.
Inyecta PlatformTransactionManager.

AVANZADO

SPRING BOOT

Transacciones

S116

RESPUESTA

Dirty read (leer no confirmado): evita READ_COMMITTED+.
Non-repeatable read (misma fila cambia entre lecturas): evita REPEATABLE_READ+.
Phantom read (aparecen filas nuevas en el rango): evita SERIALIZABLE (aprox; Postgres usa MVCC).
Default típico: Postgres/Oracle READ_COMMITTED, MySQL REPEATABLE_READ.

AVANZADO

SPRING BOOT

Transacciones

S115

RESPUESTA

Escribir en BD + publicar en cola/mensaje son 2 sistemas: puede confirmar uno y fallar el otro → inconsistencia.
Soluciones: Outbox pattern (evento en tabla outbox en la MISMA tx + proceso que publica), eventos en la misma tx a un broker transaccional, o idempotencia + reconciliación.
Base de la consistencia entre servicios.

AVANZADO

SPRING BOOT

AOP

S118

RESPUESTA

Aspect: la clase modular que agrupa funcionalidad transversal (@Aspect + @Component).
Join point: punto del programa donde PUEDE intervenir (en Spring AOP: ejecución de métodos).
Pointcut: la EXPRESIÓN que selecciona join points (execution(...)).
Advice: la acción (@Before/@After/@Around).
Weaving: tejer aspects en los beans (Spring: proxies en runtime).

AVANZADO

SPRING BOOT

Transacciones

S117

RESPUESTA

Tx larga = locks sostenidos → otros hilos esperan (timeouts en cascada), conexiones del pool ocupadas, replays de GC del COMMIT...
Buenas prácticas: transacciones cortas (abrir dentro del servicio justo para persistir), NUNCA I/O externa (HTTP, email) dentro de la tx, procesar en chunks.
Síntoma: 'connection is not available, request timed out' (HikariCP).

AVANZADO

SPRING BOOT

AOP

S120

RESPUESTA

execution(* com.app.service.*(..)) — todo el paquete service y subpaquetes.
execution(public * *(..)), execution(* Servicio*.metodo(..)).
@annotation(com.app.MiAnotacion) — métodos anotados.
within(com.app.*), this/target(bean type), bean(miBean*).
Combinar con &&, ||, !; reutilizar con @Pointcut nombrado.

AVANZADO

SPRING BOOT

AOP

S119

RESPUESTA

@Before: antes (validación, log de entrada).
@AfterReturning: tras retorno normal.
@AfterThrowing: tras excepción.
@After: tras ambos (finally).
@Around: ENVUELVE todo — puedes decidir si llamar proceed(), medir tiempo, cachear, transformar resultado. Es el más poderoso (timers, rate limiting).

AVANZADO

SPRING BOOT

AOP

S121

@Around con timer: el ejemplo que todo examen pide

AVANZADO

SPRING BOOT

AOP

S122

Limitaciones de Spring AOP (proxy-based)

AVANZADO

SPRING BOOT

AOP

S123

Ordenar aspects y handlers con @Order

AVANZADO

SPRING BOOT

AOP

S124

Casos de uso legítimos de AOP en apps de negocio

AVANZADO

SPRING BOOT

AOP

S125

¿AspectJ vs Spring AOP?

AVANZADO

SPRING BOOT

Seguridad

S126

Authentication vs Authorization (con vocabulario Spring)

INTERMEDIO

SPRING BOOT

★

Seguridad

S127

PasswordEncoder: BCrypt y mejores prácticas

INTERMEDIO

SPRING BOOT

★

Seguridad

S128

JWT flow completo en un backend Spring Boot

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

AOP

S122

RESPUESTA

Solo métodos públicos visibles por el proxy: private/final/static NO se interceptan.
 Self-invocation (this.metodo()) salta el proxy →
 @Transactional/@Cacheable/@Async 'no funcionan' dentro del mismo bean.
 Solo ejecución de métodos (no campos ni constructores, eso es AspectJ completo).
 Objetos internos creados con new nunca son proxies.

AVANZADO

SPRING BOOT

AOP

S121

RESPUESTA

```
@Around("@annotation(Monitoreado)")
public Object medir(ProceedingJoinPoint pjp) throws Throwable {
    long t0 = System.nanoTime();
    try { return pjp.proceed(); } finally {
        log.info("{} tardó {} ms", pjp.getSignature(),
            (System.nanoTime()-t0)/1_000_000);
    }
}
```

proceed(args) permite MODIFICAR argumentos/resultado. Sin proceed, el método original NO se ejecuta.

AVANZADO

SPRING BOOT

AOP

S124

RESPUESTA

Logging/auditoría estructurada (@Loggable custom), métricas/timing, manejo de errores transversal, rate limiting, autorización custom (@RequirePermission), retry declarativo, enmascarar datos sensibles en logs, propagar tenant/ctx. NO para lógica de negocio (invisible al leer el flujo — 'magia'). Regla: si lo borras, la app sigue CORRECTA (solo menos observable/protegida).

AVANZADO

SPRING BOOT

AOP

S123

RESPUESTA

@Order(n): menor = más EXTERNO (corre antes en entry, después en exit).
 Casos: un aspect de retry fuera del de logging; transacción típicamente muy interna.
 También @Order en interceptores, filtros y ExceptionHandlerAdvices.
 Sin order = indefinido entre aspects (no entre categorías distintas).

AVANZADO

SPRING BOOT

Seguridad

S126

RESPUESTA

Authentication: ¿quién eres? — AuthenticationManager → ProviderManager → AuthenticationProviders (uno valida usuario/pass, otro JWT...).
 Authorization: ¿qué puedes? — AuthorizationManager contra GrantedAuthorities.
 En el contexto:
 SecurityContextHolder.getContext().getAuthentication() (principal, authorities, credentials).

INTERMEDIO

SPRING BOOT

AOP

S125

RESPUESTA

Spring AOP: proxies dinámicos, solo métodos de beans Spring, runtime weaving — cubre 95% de casos (usar con @EnableAspectJAutoProxy).
 AspectJ completo: bytecode weaving en compilación/carga — intercepta constructores, campos, métodos private/static, objetos no-Spring; requiere procesador/loader (ajc, LTW).
 Usar AspectJ solo si el límite del proxy realmente duele.

AVANZADO

SPRING BOOT

Seguridad

S128

RESPUESTA

1) POST /login con credenciales → AuthenticationManager valida → generas JWT (jwt/nimbus) con claims (sub, roles, exp) firmado (HS256 secreto / RS256 par de claves).
 2) Cliente envía Authorization: Bearer <token>.
 3) Resource server: oauth2ResourceServer().jwt() valida firma y exp → authorities desde claims (JwtAuthenticationConverter).
 Stateless: sin sesión en servidor.

INTERMEDIO

SPRING BOOT

Seguridad

S127

RESPUESTA

SIEMPRE almacenar hashes, jamás texto plano.
 DelegatingPasswordEncoder: PasswordEncoderFactories.createDelegatingPasswordEncoder() — soporta {bcrypt}..., migración de formatos.
 BCrypt incluye salt y factor de trabajo (coste 10 default; 10-12 típico).
 Nunca MD5/SHA-1 solos para passwords.

INTERMEDIO

SPRING BOOT

Seguridad

S129

HS256 vs RS256 para firmar JWT

INTERMEDIO

SPRING BOOT

Seguridad

S130

Validar JWT a mano: qué DEBES chequear

INTERMEDIO

SPRING BOOT

Seguridad

S131

Session vs Stateless: sessionCreationPolicy

INTERMEDIO

SPRING BOOT

★

Seguridad

S132

CSRF: qué es y cuándo aplica en APIs

INTERMEDIO

SPRING BOOT

Seguridad

S133

401 vs 403 en Spring Security y cómo customizar la respuesta

INTERMEDIO

SPRING BOOT

Seguridad

S134

OAuth2 Client: login social/terceros con Spring

INTERMEDIO

SPRING BOOT

Seguridad

S135

OAuth2 Resource Server: proteger tu API con tokens de terceros

INTERMEDIO

SPRING BOOT

Seguridad

S136

Arquitectura de Spring Security: la cadena de filtros

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Seguridad

S130

RESPUESTA

1) Firma contra JWKS/secreto.
 2) exp (expiración) y nbf/iat razonables.
 3) iss (emisor esperado) y aud (audiencia = tu servicio).
 4) Algoritmo esperado (rechazar 'alg: none' o cambios).
 Nunca decodificar SIN validar (Base64 NO es seguridad).
 Spring: `oauth2ResourceServer().jwt()` con `issuer-uri` o `jwt-set-uri` hace todo esto.

INTERMEDIO

SPRING BOOT

Seguridad

S129

RESPUESTA

HS256: secreto compartido — simple, pero TODO servicio que valida necesita el secreto (riesgo, no se puede verificar 'solo').
 RS256: par privado/público — firmas con clave privada, cualquiera valida con la pública → ideal multi-servicio y OAuth servers.
 Rotación: publicar claves en JWKS endpoint (`/.well-known/jwks.json`) — el resource server las descarga y cachea.

INTERMEDIO

SPRING BOOT



Seguridad

S132

RESPUESTA

Ataque: un sitio malicioso hace el navegador enviar requests con TU cookie de sesión (form oculto, fetch same-site). Solo aplica con autenticación por COOKIE.
 Defensa: token CSRF (Spring: `CookieCsrfTokenRepository` con `HttpOnly=false` para SPAs) o `SameSite=Lax` cookies.
 APIs stateless con Authorization header → `http.csrf(AbstractHttpConfigurer::disable)` es correcto.

INTERMEDIO

SPRING BOOT

Seguridad

S131

RESPUESTA

IF_REQUIRED (default): crea sesión (apps web clásicas).
 STATELESS: nunca crea ni usa sesión — cada request trae credenciales (JWT). OBLIGATORIO en APIs con tokens.
 NEVER, ALWAYS, INVALIDATE_ON_AUTHENTICATION... para casos mixtos.
 Con STATELESS, CSRF se deshabilita (no hay cookie de sesión que abuse — ver CSRF).

INTERMEDIO

SPRING BOOT

Seguridad

S134

RESPUESTA

`spring-boot-starter-oauth2-client` + properties del provider (`client-id`, `secret`, `authorization-grant-type`, `redirect-uri`).
`http.oauth2Login()` → flujo Authorization Code completo gestionado por Spring.
 Custom: provider propio en `spring.security.oauth2.client.registration`.
 El principal queda `OAuth2User` (attributes del userinfo).

INTERMEDIO

SPRING BOOT

Seguridad

S133

RESPUESTA

401 no autenticado (sin/vencido token) — `AuthenticationEntryPoint`.
 403 autenticado pero sin permiso — `AccessDeniedHandler`.
 Customizar: `exceptionHandling(e -> e.authenticationEntryPoint(mi401).accessDeniedHandler(mi403))` — devolver JSON `ProblemDetail` en APIs (default redirige a `/error` o login).

INTERMEDIO

SPRING BOOT

Seguridad

S136

RESPUESTA

`SecurityFilterChain`: una secuencia de filtros servlet antes de tu controller — `SecurityContextPersistenceFilter` (carga sesión/token), autenticación (`UsernamePassword`, `BearerToken`...), `ExceptionTranslationFilter` (401/403), `FilterSecurityInterceptor/AuthorizationFilter` (permisos).
 El resultado: `SecurityContext` con Authentication disponible en cualquier punto.

AVANZADO

SPRING BOOT

Seguridad

S135

RESPUESTA

`oauth2ResourceServer().jwt().issuer-uri(https://authserver)` — descarga JWKS y valida.
 Roles desde claims: `JwtAuthenticationConverter` → `setJwtGrantedAuthoritiesConverter` (mapear claim 'roles' → `SimpleGrantedAuthority`).
 Props: `spring.security.oauth2.resourceserver.jwt.issuer-uri`.
 El auth server (Keycloak, Auth0, Identity Platform) emite; tú solo validas.

INTERMEDIO

SPRING BOOT

★

Seguridad

S137

Config moderna: SecurityFilterChain bean
(Spring Security 6)

AVANZADO

SPRING BOOT

Seguridad

S138

UserDetailsService y UserDetails:
implementación típica

AVANZADO

SPRING BOOT

Seguridad

S139

Roles vs Authorities: anotaciones y
expresiones

AVANZADO

SPRING BOOT

Seguridad

S140

Multi-tenancy ligero: tenant por header

AVANZADO

SPRING BOOT

Seguridad

S141

¿Cómo probar endpoints protegidos en
tests?

AVANZADO

SPRING BOOT

Seguridad

S142

Secrets en Spring Boot: dónde NO ponerlos
y dónde sí

AVANZADO

SPRING BOOT

Testing

S143

La pirámide de testing aplicada a Spring
Boot

INTERMEDIO

SPRING BOOT

★

Testing

S144

@WebMvcTest: qué levanta y qué debes
mockear

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Seguridad

S138

RESPUESTA

UserDetailsService.loadUserByUsername(String) → UserDetails (username, password hash, authorities, enabled).
 Impl con tu repo de usuarios; Spring valida password con PasswordEncoder.matches.
 Si usas JWT/OAuth sin usuarios internos, quizá no necesites UserDetailsService (custom provider o JWT decoder).

AVANZADO

SPRING BOOT



Seguridad

S137

RESPUESTA

```
@Bean SecurityFilterChain chain(HttpSecurity http) {
    http.csrf(...).authorizeHttpRequests(a -> a.requestMatchers("/public/**").permitAll().anyRequest().authenticated()).oauth2ResourceServer(...).sessionManagement(s -> s.sessionCreationPolicy(STATELESS)); return http.build(); }
WebSecurityConfigurerAdapter está DEPRECADO — todo con beans.
```

AVANZADO

SPRING BOOT

Seguridad

S140

RESPUESTA

Filter propio al inicio de la chain: extrae X-Tenant-Id → guarda en ThreadLocal/TenantContext.
 Datasource o schema dinámico: AbstractRoutingDataSource decide por tenant.
 Validar SIEMPRE que el tenant coincida con los claims del token (no confiar en el header solo).
 Limpieza del ThreadLocal al final (pools de hilos).

AVANZADO

SPRING BOOT

Seguridad

S139

RESPUESTA

Authority = permiso fino ("pedido:leer"); ROLE_ es una convención de prefijo (ROLE_ADMIN).
 http.authorizeHttpRequests: hasAuthority("x"), hasRole("ADMIN") (añade ROLE_ automáticamente).
 Método: @PreAuthorize("hasRole('ADMIN') or #id == authentication.principal.id") — SpEL con parámetros (#id).
 Habilitar: @EnableMethodSecurity.

AVANZADO

SPRING BOOT

Seguridad

S142

RESPUESTA

NO: application.yml commiteado, código, images, logs.
 SÍ: variables de entorno (12-factor), Secret Manager/Vault (leer al arranque o con Spring Cloud Vault), archivos montados por K8s.
 Placeholder: \${DB_PASS} desde env.
 Rotación: backend con relectura o restart controlado; jamás loguear properties con password (logging.mask).

AVANZADO

SPRING BOOT

Seguridad

S141

RESPUESTA

@WithMockUser(username, roles={"ADMIN"}) sobre el test o método.
 @WithJwt / @WithSecurityContext para JWTs custom.
 MockMvc + SecurityMockMvcRequestPostProcessors.jwt() (de spring-security-test): .with(jwt().authorities(...)).
 CSRF en MockMvc: .with(csrf()) para POST.

AVANZADO

SPRING BOOT



Testing

S144

RESPUESTA

Levanta SOLO la capa web: controllers, @ControllerAdvice, filtros/converters MVC, MockMvc autoconfigurado.
 NO levanta services/repos → inyectas @MockitoBean (Spring Boot 3.4+, antes @MockBean) MiService.
 Uso: validar mappings, status codes, validación de DTOs, manejo de errores, serialización.

INTERMEDIO

SPRING BOOT

Testing

S143

RESPUESTA

Base (muchos): tests UNITARIOS de lógica pura sin Spring (rápidos, JUnit+Mockito).
 Medio: tests de SLICE — @WebMvcTest, @DataJpaTest, @JsonTest (levantan SOLO esa capa, segundos).
 Cima (pocos): @SpringBootTest integración completa (+Testcontainers con BD real).
 Anti-patrón: todo @SpringBootTest → suite de horas.

INTERMEDIO

SPRING BOOT

★

Testing

S145

MockMvc: el recetario de asserts

INTERMEDIO

SPRING BOOT

Testing

S146

@DataJpaTest y repositorios: qué prueba y qué trae

INTERMEDIO

SPRING BOOT

Testing

S147

Testcontainers: BD real en Docker para tus tests

INTERMEDIO

SPRING BOOT

Testing

S148

@MockitoBean (ex @MockBean) vs Mockito @Mock: diferencia clave

INTERMEDIO

SPRING BOOT

Testing

S149

Mockito: verify, ArgumentCaptor y stubbing útil

INTERMEDIO

SPRING BOOT

Testing

S150

@SpringBootTest con puerto real: @LocalServerPort y TestRestTemplate

INTERMEDIO

SPRING BOOT

Testing

S151

Cómo probar código async y schedulers

INTERMEDIO

SPRING BOOT

Testing

S152

Fixture de datos de prueba: @Sql, builders y fixtures

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Testing

S146

RESPUESTA

Levanta JPA + DataSource EM_MEMORIA (H2 por defecto) + repositorios; transacción por test con rollback.
 Configurar BD real: @AutoConfigureTestDatabase(replace = NONE) + Testcontainers.
 Probar: queries derivadas/@Query críticas, constraints, mapeo de entidades, paginación.
 H2 ≠ tu motor: para SQL nativo complejo usa Testcontainers con el motor real.

INTERMEDIO

SPRING BOOT



Testing

S145

RESPUESTA

mockMvc.perform(get("/api/pedidos/1").header(...)).andExpect(status().isOk()).andExpect(jsonPath("\$.cliente.nombre").value("Ana")).andExpect(jsonPath("\$.items", hasSize(3))).andDo(print() / document()).
 POST con body: contentType(APPLICATION_JSON).content(objectMapper.writeValueAsString(dto)).
 jsonPath usa expresiones JsonPath (\$) = raíz.

INTERMEDIO

SPRING BOOT

Testing

S148

RESPUESTA

@MockitoBean: Spring reemplaza el bean EN EL CONTEXTO — para tests con Spring que inyectan dependencias.
 @Mock (Mockito): mock puro sin Spring — para tests unitarios con constructor (new Servicio(mock)).
 Trampa: @MockitoBean crea un NUEVO contexto (cache de contextos se rompe si varían) → suite lenta; minimizar combinaciones distintas.

INTERMEDIO

SPRING BOOT

Testing

S147

RESPUESTA

@Testcontainers + @Container static PostgreSQLContainer postgres = new PostgreSQLContainer("postgres:16").
 @DynamicPropertySource registra spring.datasource.url/username/password dinámicos.
 Beneficio: el MISMO motor que producción, migraciones Flyway reales ejecutadas.
 Costo: Docker requerido y segundos extra (contenedores estáticos reutilizados lo mitigan).

INTERMEDIO

SPRING BOOT

Testing

S150

RESPUESTA

@SpringBootTest(webEnvironment = RANDOM_PORT): levanta el servidor completo en un puerto libre.
 @LocalServerPort int port inyecta el puerto; cliente: TestRestTemplate (auto-configurado) o RestClient apuntando a localhost:port.
 Para pruebas end-to-end del flujo HTTP real (auth incluida) — pocas, críticas.

INTERMEDIO

SPRING BOOT

Testing

S149

RESPUESTA

when(repo.findById(1L)).thenReturn(Optional.of(entidad));
 thenThrow para excepciones.
 verify(repo).save(captor.capture()); captor.getValue() → assert del objeto guardado.
 never(), times(2), any(), eq() — mezclar matchers requiere TODOS matchers.
 anyMatch: when(x.metodo(anyString())).thenReturn(...).
 Comportamiento default: null/0/false (relaxed).

INTERMEDIO

SPRING BOOT

Testing

S152

RESPUESTA

@Sql("/data/test-usuarios.sql") ejecuta scripts antes del test (@Sql(executionPhase = AFTER_TEST_METHOD) para limpiar).
 @BeforeEach con builders de entidades (ObjectMother/builder pattern) — datos legibles en el propio test.
 Base de datos en memoria por test: @Transactional rollback automático.
 Evitar shared mutable state entre tests (orden aleatorio ayuda: JUnit method order RANDOM).

INTERMEDIO

SPRING BOOT

Testing

S151

RESPUESTA

Async: capturar el CompletableFuture y .get(timeout), o Awaitility: await().atMost(5, SECONDS).untilAsserted(() -> assertThat(...)).
 Scheduled: no duermas 5 min — extraer la lógica a un método testeable y llamarlo directo; o @TestPropertySource con cron acelerado.
 Eventos: @RecordApplicationEvents + events.stream(...).

INTERMEDIO

SPRING BOOT

Testing

S153

AssertJ: los asserts fluidos que reemplazan a JUnit asserts

INTERMEDIO

SPRING BOOT

Testing

S154

¿Cómo medir y actuar sobre cobertura (JaCoCo)?

INTERMEDIO

SPRING BOOT

Testing

S155

Test de una clase con @Transactional: ¿se prueba la transacción?

INTERMEDIO

SPRING BOOT

★

Producción y observabilidad

S156

¿Qué ofrece Spring Boot Actuator?

BASICO

SPRING BOOT

Producción y observabilidad

S157

Micrometer: cómo Spring Boot mide todo

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S158

Health indicator custom y health groups

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S159

Logging con traceld correlacionado (MDC + Micrometer Tracing)

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S160

Config de logging que debe saber un backend

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Testing

S154

RESPUESTA

Plugin jacoco-maven-plugin: report en target/site/jacoco; check con reglas (BUNDLE line ratio 0.7) que fallen el build si bajo.
Realismo: 100% es ruido; apunta a lógica de negocio alta, config/trivial baja.
Cobertura no prueba decisiones: un test que ejecuta sin asserts cubre igual (mutation testing lo delata — PIT).

INTERMEDIO

SPRING BOOT

Testing

S153

RESPUESTA

assertThat(lista).hasSize(3).extracting("nombre").containsExactly("a","b").
assertThat(excepcion).assertInstanceOf(...).hasMessageContaining(...).
assertThat(dto).usingRecursiveComparison().ignoringFields("id").isEqualTo(otro).
withFailMessage, satisfies(v -> ...). Viene incluido en starter-test.

INTERMEDIO

SPRING BOOT



Producción y observabilidad

S156

RESPUESTA

Endpoints de operación:
/actuator/health (con liveness/readiness), /metrics (JVM, HTTP, pools), /info, /env, /prometheus (exporter para Grafana).
Se exponen con management.endpoints.web.exposure.include.
Base del monitoreo y de los probes de K8s.

BASICO

SPRING BOOT

Testing

S155

RESPUESTA

El test @Transactional hace rollback → el commit real NO se ejecuta: constraints deferred, triggers ON COMMIT y behaviours de aislamiento no se validan.
Para validar eso: test SIN @Transactional + limpieza manual, o TestTransaction.commit() explícito dentro del test.
Elije conscientemente.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S158

RESPUESTA

implements HealthIndicator: Health.up().withDetail("latencia", ms).build() / down(excepción).
Bean nombrado xxxHealthIndicator → /actuator/health/{xxx}.
Health groups:
management.endpoint.health.group.readiness.include=db,ping → /actuator/health/readiness para K8s.
Cuidado: dependencia lenta DOWN = app fuera del balanceo (usar cache/status opcional).

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S157

RESPUESTA

Micrometer es la fachada de métricas (SLF4J para métricas):
Timer, Counter, Gauge, DistributionSummary.
registry.timer("pedidos.proceso", "tipo", "web").record(...); o @Timed/@Counted.
/exporter: /actuator/prometheus → Prometheus → Grafana.
Métricas auto: JVM, HTTP (duración, status), Hikari, cachés.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S160

RESPUESTA

logging.level.root=INFO, logging.level.com.miapp=DEBUG (por paquete).
Pattern con timestamp, hilo, level, logger, traceId.
Salida estructurada JSON en prod (logstash-logback-encoder) para ingest.
Rolling: size + días retenidos. NUNCA logear passwords/tokens/PII completa.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S159

RESPUESTA

Micrometer Tracing (ex Sleuth) propaga traceId/spanId en MDC → patrón de log: %X{traceId}.
logging.pattern.level="%\${LOG_LEVEL} [%X{traceId:-}]".
Un request a través de 5 servicios = un traceId en los logs de todos → grep único para investigar.
W3C traceparent o B3 headers entre servicios.

INTERMEDIO

SPRING BOOT

★

Producción y observabilidad

S161

Graceful shutdown y readiness durante deploy

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S162

Dockerizar Spring Boot: layered jars

AVANZADO

SPRING BOOT

Producción y observabilidad

S163

Virtual threads en Spring Boot 3.2+: un flag

AVANZADO

SPRING BOOT

Producción y observabilidad

S164

GraalVM native image en Spring Boot: qué cambia

AVANZADO

SPRING BOOT

Producción y observabilidad

S165

Perfil prod: checklist de propiedades

AVANZADO

SPRING BOOT

Async, caché y scheduling

S166

@Async: setup y reglas para que funcione

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S167

@Scheduled: cron, fixedRate vs fixedDelay y zona

INTERMEDIO

SPRING BOOT

★

Async, caché y scheduling

S168

@Cacheable, @CacheEvict, @CachePut: semántica exacta

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Producción y observabilidad

S162

RESPUESTA

spring-boot-jarmode-layers: java -Djarmode=tools -jar app.jar
 extract → Dockerfile con COPY de capas separadas
 (dependencies, snapshot, app) → el cache de Docker solo
 re-descarga la capa 'app' cuando cambia TU código (imagen
 rebuild 10x más rápida).
 Buildpacks (bootBuildImage) es la alternativa cero-Dockerfile.

AVANZADO

SPRING BOOT



Producción y observabilidad

S161

RESPUESTA

server.shutdown=graceful +
 spring.lifecycle.timeout-per-shutdown-phase=30s: al recibir
 SIGTERM, la app deja de aceptar requests NUEVOS, termina los
 en curso y recién cierra.
 Combinar con readiness probe: quitar del balanceo ANTES de
 matar (preStop hook en K8s).
 Evita 502s y requests cortados en cada deploy.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S164

RESPUESTA

AOT: compila a binario nativo — arranque <100ms y footprint
 mínimo (ideal serverless/scale-to-zero).
 Costos: build lento, reflexión/JPA dinámicos requieren hints,
 alguna librería incompatible.
 Spring Boot 3 trae soporte AOT + plugins. Evaluar solo si el
 arranque/frío importa de verdad.

AVANZADO

SPRING BOOT

Producción y observabilidad

S163

RESPUESTA

spring.threads.virtual.enabled=true → Tomcat, @Async y
 schedulers usan hilos virtuales.
 I/O-bound masivo con código bloqueante simple: miles de
 requests concurrentes sin WebFlux ni pools gigantes.
 Cuidado: synchronized prolongado (pinning), CPU-bound no
 mejora, y pools de conexiones quedan como cuello (ajustar
 Hikari).

AVANZADO

SPRING BOOT

Async, caché y scheduling

S166

RESPUESTA

1) @EnableAsync en una @Configuration.
 2) Configurar executor: AsyncConfigurer o spring.task.execution.*
 (default SimpleAsyncTaskExecutor crea un hilo por llamada —
 ¡peligroso en prod!).
 3) Método público llamado DESDE OTRO bean (proxy).
 Retorno: void o Future/CompletableFuture. Excepciones:
 AsyncUncaughtExceptionHandler si void.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S165

RESPUESTA

ddl-auto=validate, show-sql=false, logging INFO + JSON,
 actuator expone solo health/info/prometheus (nada de /env),
 CORS restricto, TLS en el LB, secrets por env/Secret Manager,
 timeouts de clientes configurados, Hikari sized, graceful
 shutdown, management probes habilitados
 (management.endpoint.health.probes.enabled=true).

AVANZADO

SPRING BOOT



Async, caché y scheduling

S168

RESPUESTA

@Cacheable("productos", key="#id"): si está en caché NO ejecuta
 el método.
 @CachePut: SIEMPRE ejecuta y actualiza la caché (para writes que
 deben refrescar).
 @CacheEvict: invalida (allEntries=true borra todo).
 unless="#result == null" evita cachear vacíos; condition filtra.
 Requiere @EnableCaching y un CacheManager (Caffeine local,
 Redis distribuido).

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S167

RESPUESTA

@Scheduled(cron = "0 0 2 * *", zone = "America/Lima") — 2
 a.m. (seg min hora día mes díaSem).
 fixedRate = cada N ms DESDE EL INICIO; fixedDelay = N ms
 DESDE EL FIN (nunca solapa) — el seguro para tareas lentas.
 Requiere @EnableScheduling. default es single-thread: tareas
 largas retrasan las demás → configurar TaskScheduler pool.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S169

Caffeine: configuración típica de caché local

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S170

Caché con Redis en Spring: consideraciones

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S171

Cuándo cachear y cuándo NO (decisiones que evaluán en entrevista)

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S172

Eventos de aplicación: @EventListener y publishing

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S173

Schedulers en cluster: por qué necesitas un lock (ShedLock)

AVANZADO

SPRING BOOT

Async, caché y scheduling

S174

Cache stampede (thundering herd): qué es y cómo mitigarlo

AVANZADO

SPRING BOOT

Async, caché y scheduling

S175

Cuándo @EventListener vs @TransactionalEventListener

AVANZADO

SPRING BOOT

★

Spring Cloud y resiliencia

S176

¿Qué es un Circuit Breaker (Resilience4j)?

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Async, caché y scheduling

S170

RESPUESTA

spring-boot-starter-data-redis + RedisCacheManager (Jackson2JsonRedisSerializer para guardar DTOs).
 Claves: nombres de caché como prefijo (productos::42).
 TTL obligatorio (todo dato cached expira), invalidación cruzada entre servicios por eventos.
 Serialización estable: sin classnames frágiles, versionar DTOs cached.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S169

RESPUESTA

Caffeine.newBuilder().maximumSize(10_000).expireAfterWrite(Duration.ofMinutes(10)).refreshAfterWrite(5min).
 .maximumWeight + weigheer para limitar por memoria.
 recordStats() para métricas (hit rate).
 En Spring: SpringApplicationBuilder o CaffeineCacheManager;
 expiración variable con expireAfter(Expiry custom).

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S172

RESPUESTA

publisher.publishEvent(new PedidoCreadoEvent(id)) →
 @EventListener corren sincrónicos EN EL MISMO HILO/tx por defecto.
 Async: @Async sobre el listener.
 Order con @Order. Eventos desacoplan módulos internos (no son mensajes de microservicios).
 Payload inmutable; listener NO debe fallar el flujo principal (try/catch o async).

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S171

RESPUESTA

Sí: lecturas frecuentes de datos poco cambiantes (catálogos, configs, cálculos costosos), alivio a BD bajo picos.
 NO: datos que cambian con cada write, respuestas personalizadas sin clave correcta, datos críticos sin tolerancia a stale, tasa de hit baja (overhead gratis).
 TTL + invalidación + stampede protection (refreshAhead/singleflight) = el kit completo.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S174

RESPUESTA

Clave expira → 1000 requests concurrentes golpean la BD simultáneamente para rehacerla.
 Mitigación: lock/singleflight (uno recalcula, el resto espera el nuevo valor), refreshAhead (renovar antes de expirar), TTL con jitter aleatorio, y fallback a valor stale mientras se recalcula (serve-stale).

AVANZADO

SPRING BOOT

Async, caché y scheduling

S173

RESPUESTA

Con 3 réplicas, @Scheduled corre 3 VECES (una por instancia).
 Soluciones: ShedLock (@SchedulerLock(name, lockAtMostFor) + tabla/Redis: solo una instancia ejecuta), Quartz clusterizado, o mover al scheduler del cloud (Cloud Scheduler → endpoint, Cloud Tasks).
 Elegir idempotencia SIEMPRE como red de seguridad.

AVANZADO

SPRING BOOT



Spring Cloud y resiliencia

S176

RESPUESTA

Patrón anti-cascada: si un servicio remoto falla N veces, el circuito se ABRE y falla rápido sin llamarlo (fallback); tras un tiempo pasa a HALF-OPEN y prueba de nuevo; si funciona, se CIERRA.
 @CircuitBreaker(name="servicio", fallbackMethod="respaldo").

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S175

RESPUESTA

@EventListener: inmediato (misma tx — ve cambios sin confirmar; si el listener escribe, comparte tx).
 @TransactionalEventListener(AFTER_COMMIT): solo si la tx CONFIRMA — el estándar para efectos externos (emails, colas, invalidar caché).
 Trampa AFTER_COMMIT: nueva tx propia para escribir (REQUIRES_NEW o TransactionTemplate).
 fallbackExecution=true si no hay tx.

AVANZADO

SPRING BOOT

★

Spring Cloud y resiliencia

S177

Spring Cloud Gateway: qué hace y config básica

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S178

Resilience4j: los 4 módulos y anotaciones

INTERMEDIO

SPRING BOOT

★

Spring Cloud y resiliencia

S179

Retry bien hecho: backoff, jitter y cuándo NO reintentar

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S180

OpenFeign: cliente declarativo y sus settings críticos

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S181

CircuitBreaker config: qué significa cada número

AVANZADO

SPRING BOOT

Spring Cloud y resiliencia

S182

Spring Cloud Config Server: qué aporta hoy

AVANZADO

SPRING BOOT

Troubleshooting clásico

S183

NoUniqueBeanDefinitionException: diagnóstico en 3 pasos

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S184

BeanCurrentlyInCreationException (dependencia circular): salidas

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Spring Cloud y resiliencia

S178

RESPUESTA

@CircuitBreaker (corta tras N fallos), @Retry (n reintentos + backoff), @RateLimiter (límite por periodo), @Bulkhead (semáforo o pool de hilos), @TimeLimiter (timeout para futures). Combinables en un método (orden configurable). fallbackMethod = "miMetodo(Throwable)" con firma compatible. Métricas expuestas vía Micrometer: estado del breaker en /metrics.

INTERMEDIO

SPRING BOOT



Spring Cloud y resiliencia

S177

RESPUESTA

Gateway reactivo (Netty) como entrada única: rutas por path/host con filtros.
spring.cloud.gateway.routes: id, uri, predicates (Path=/api/**), filters (StripPrefix=1, AddRequestHeader, Retry, CircuitBreaker).
Filters custom: GatewayFilterFactory — auth, rate limit (RequestRateLimiter con Redis).
No pongas lógica de negocio en el gateway.

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S180

RESPUESTA

@FeignClient(name="pagos", url="\${servicios.pagos.url}", configuration=FeignCfg.class) interface PagosClient {
@PostMapping("/cargar") Carga cargar(@RequestBody CargaReq req); }
@EnableFeignClients. Timeouts SIEMPRE (connect/read por client), retry acotado, error decoder (ErrorDecoder → excepciones de negocio), y circuit breaker integrado (circuitbreaker.enabled=true).
Interceptors para tokens: RequestInterceptor con Authorization header.

INTERMEDIO

SPRING BOOT



Spring Cloud y resiliencia

S179

RESPUESTA

maxAttempts=3, waitDuration=200ms, enableExponentialBackoff (x2), enableRandomizedJitter (evita sincronización de reintentos).
Reintentar SOLO errores transitorios (503, timeouts, conexiones) — NO 400/401/404 (deterministas).
Idempotencia obligatoria en la operación reintentada.
Retry + circuit breaker juntos: el breaker evita martillar cuando todo está caído.

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S182

RESPUESTA

Centraliza properties por app/perfil en un repo Git: los servicios las cargan al arranque desde http://config-server/app/prod.
@RefreshScope + actuador/refresh recarga valores SIN redeploy (con bus de eventos: refresca todas las réplicas).
Alternativa moderna en k8s: ConfigMaps + reloder (spring-cloud-kubernetes). Aun así, el patrón externalized config es el que importa.

AVANZADO

SPRING BOOT

Spring Cloud y resiliencia

S181

RESPUESTA

slidingWindowSize=10 (ventana de llamadas evaluada), failureRateThreshold=50 (% de fallos para abrir), waitDurationInOpenState=30s (tiempo abierto), permittedNumberOfCallsInHalfOpenState=3, slowCallRateThreshold + slowCallDurationThreshold (latencia alta cuenta como fallo).
En yaml: resilience4j.circuitbreaker.instances.miServicio.*

AVANZADO

SPRING BOOT

Troubleshooting clásico

S184

RESPUESTA

Mensaje típico: 'Requested bean is currently in creation'.
Opciones en orden de preferencia: 1) Rediseñar — extraer lo común a un tercer bean. 2) @Lazy en UNA de las inyecciones. 3) Setter injection en lugar de constructor (solo como parche).
La circularidad casi siempre indica responsabilidades mal separadas.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S183

RESPUESTA

1) Leer el mensaje: lista los beans candidatos.
2) Decidir: @Primary (default), @Qualifier (punto de inyección), o renombrar el campo para que coincida con el nombre del bean.
3) Si es involuntario: revisar componentes duplicados por escaneo (paquete repetido, clases de prueba escaneadas).
En tests: @MockitoBean reemplaza y elimina la ambigüedad.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S185

Port 8080 was already in use: soluciones rápidas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S186

Cannot resolve property / Property or field 'x' cannot be found: causas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S187

HikariPool - Connection is not available, request timed out: por qué ocurre

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S188

JSON parse error / 415 Unsupported Media Type: checklist

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S189

Infinite recursion con Jackson (StackOverflow): 4 salidas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S190

Tabla not found / SQLGrammarException al arrancar: causas típicas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S191

El test pasa en local pero falla en CI: los 5 clásicos

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S192

502/504 detrás de un load balancer con Spring Boot: dónde mirar

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Troubleshooting clásico

S186

RESPUESTA

- 1) En SpEL/JPQL: typo o nombre de propiedad inexistente en la entidad (JPQL usa campos de entidad, NO columnas).
 - 2) En @ConfigurationProperties: relaxed binding mal aplicado (lista sin corchetes).
 - 3) Typo en application.yml bajo el prefijo equivocado.
- Validar: arranque con --debug y CONDITIONS report; correr el query a mano.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S185

RESPUESTA

Ver quién ocupa: Windows netstat -ano | findstr :8080 + taskkill /PID x /F (o cambiar server.port=0/otro).

En dev: apagar el otro servicio (o el Docker que publicó el puerto), o correr con --server.port=9090.

En tests: RANDOM_PORT siempre.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S188

RESPUESTA

415: el cliente NO envía Content-Type: application/json, o el endpoint exige consumes distinto.

JSON error 400: payload mal formado, campo con tipo incorrecto (string vs number), DTO sin constructor vacío/setters (Jackson necesita), o getter/setter faltantes.

Debug: loguear el body recibido (solo en dev) y probar la serialización del DTO aislada.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S187

RESPUESTA

Pool agotado: conexiones en uso > maximumPoolSize y nadie libera.

Causas: transacciones largas (I/O externa dentro de tx), locks de BD, conexiones sin cerrar (streams no cerrados), pool chico para la carga.

Fixes: tx cortas, raise pool (con margen de BD), leakDetectionThreshold=30000 para culprits, métricas de Hikari (active/idle/wait).

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S190

RESPUESTA

- 1) ddl-auto sin create + BD vacía y sin migraciones.
 - 2) Flyway con versión de script menor a la registrada (schema history).
 - 3) Naming: tabla snake_case en BD vs entidad sin naming strategy (spring.jpa.hibernate.naming.physical-strategy).
 - 4) Esquema/schema search path distinto (Postgres).
- Validar contra la BD real, no contra la entidad.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S189

RESPUESTA

Relación A→B→A serializando.

- 1) DTOs (la solución real).
- 2) @JsonIgnore en el lado inverso.
- 3) @JsonManagedReference (padre) + @JsonBackReference (hijo).
- 4) @JsonIdentityInfo(generator=ObjectIdGenerators.PropertyGenerator.class, property="id").
- 5) @JsonProperty(access=WRITE_ONLY) para entrada sin salida.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S192

RESPUESTA

502: la app rechaza/cierra — durante deploy sin graceful shutdown, proceso caído, puerto equivocado en el target group.

504: la app tarda más que el timeout del LB/ingress (60s default) — perfil de lentitud, timeouts del LB ajustados o request asíncrono.

Confirmar: logs de la app a esa hora + health del backend + readiness antes del switch de tráfico.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S191

RESPUESTA

- 1) Depende de recursos locales (puerto, archivo, red) — mockear/contenerizar.
 - 2) TZ/locáles distintas → fechas rotas: fijar TZ/UTF-8 en el runner.
 - 3) Orden de tests / estado compartido.
 - 4) Docker indisponible para Testcontainers.
 - 5) Flaky por timing (async sin await — usar Awaitility).
- Correr tests en paralelo/aleatorio localmente los vuelve robustos.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S193

Cómo diagnosticar un endpoint lento (metodología)

INTERMEDIO

SPRING BOOT

Ecosistema y build

S194

Maven lifecycle: clean, compile, test, package, verify, install, deploy

BASICO

SPRING BOOT

Ecosistema y build

S195

scope de dependencias en Maven

BASICO

SPRING BOOT

Ecosistema y build

S196

dependencyManagement vs dependencies (y BOMs)

BASICO

SPRING BOOT

Ecosistema y build

S197

Exclusions y conflictos: el flujo de diagnóstico

BASICO

SPRING BOOT

Ecosistema y build

S198

Maven vs Gradle: diferencias prácticas

BASICO

SPRING BOOT

Ecosistema y build

S199

Lombok: anotaciones y las 2 trampas

BASICO

SPRING BOOT

Trampas de entrevista

S200

¿Dos beans del mismo tipo pero uno con @Profile: es ambiguo?

BASICO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Ecosistema y build

S194

RESPUESTA

Fases en orden: compile < test < package (jar/war) < verify (integration tests, jacoco check) < install (al repo local) < deploy (a repo remoto).

Ejecutar 'mvn package' corre TODAS las fases anteriores.

Plugins se enganchan a fases (surefire→test, boot plugin→repackage crea el jar ejecutable).

-mvn clean borra target/.

BASICO

SPRING BOOT

Troubleshooting clásico

S193

RESPUESTA

- 1) Medir por capas: gateway log → micrometer
http.server.requests.percentile → logs con timing.
- 2) Activar show-sql/log de Hikari SOLO en repro local; buscar N+1 y queries sin índice (EXPLAIN).
- 3) Revisar llamadas externas (timeouts, latencia del cliente HTTP).
- 4) GC/hilos en JFR si todo interno luce bien.
- 5) Fix + test de carga que demuestre la mejora.

INTERMEDIO

SPRING BOOT

Ecosistema y build

S196

RESPUESTA

dependencyManagement: declara VERSIONES sin incluir la dependencia (los hijos heredan versión).

BOM (bill of materials): pom solo de versiones importado con scope=import — spring-boot-dependencies alinea 300 librerías.

Resultado: cambiar versión en un solo lugar; sin versiones duplicadas ni 'inferno de dependencias'.

BASICO

SPRING BOOT

Ecosistema y build

S195

RESPUESTA

compile (default): todo. provided: compilación sí, runtime NO lo empaca (servlet-api, Lombok). runtime: solo runtime. test: solo tests (JUnit, Mockito). import: solo en dependencyManagement (importar BOMs).

Transitivo: compile fluye; test/provided no.

Conflicto de versiones: gana el más CERCANO al root (nearest wins) — dependency:tree para diagnosticar.

BASICO

SPRING BOOT

Ecosistema y build

S198

RESPUESTA

Maven: XML declarativo, convención sólida, el más extendido en empresas; build ligeramente más lento.

Gradle: Kotlin/Groovy DSL, build incremental y cacheado (más rápido), flexible (a veces demasiado).

Ambos con mwnw/gradlew wrapper para versionar el build tool. Elige el estándar del equipo.

BASICO

SPRING BOOT

Ecosistema y build

S197

RESPUESTA

mvn dependency:tree -Dincludes=com.google.guava:guava → quién trae qué versión.

Excluir en la dependencia culpable:

<exclusions> <exclusion>...</exclusion> </exclusions>, y declarar la versión deseada explícita (o en dependencyManagement).

Síntomas típicos: NoSuchMethodError/ClassNotFoundError en runtime por versiones mezcladas.

BASICO

SPRING BOOT

Trampas de entrevista

S200

RESPUESTA

No, si los perfiles son excluyentes: solo un bean EXISTE por perfil activo → sin ambigüedad en runtime.

Pero en tests sin perfil activo ambos pueden no existir o existir según configuración.

Patrón: impl por entorno (FakeNotificador dev / EmailNotificador prod) — ejemplo perfecto de @Profile.

BASICO

SPRING BOOT

Ecosistema y build

S199

RESPUESTA

@Getter/@Setter, @RequiredArgsConstructor (inyección de dependencias final — la base del estilo constructor), @Builder, @Data (¡incluye equals/hashCode/toString con TODOS los campos — peligro en entidades JPA!), @Slf4j, @Value (inmutable), @EqualsAndHashCode(exclude={...}) (onlyExplicitlyIncluded). Trampas: IDE sin plugin no compila; @Data en entidades rompe equals/hashCode con lazy/collections.

BASICO

SPRING BOOT

WebFlux y reactivo

S201

WebFlux vs MVC: cuál elegir con datos en la mano

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S202

Mono y Flux: semántica exacta

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S203

¿Puedes inyectar en un campo static? ¿Y por qué no?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S204

¿Cuándo se crea un bean @Lazy si nadie lo usa? ¿Y si otro bean lo inyecta?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S205

¿Un bean prototype con @PostConstruct corre por cada instancia?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S206

¿Qué pasa si dos @Bean devuelven el MISMO tipo y un tercero lo inyecta por constructor?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S207

¿getBean() dentro de un service es buena idea?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S208

¿@Transactional en la clase Y en un método: qué gana? ¿Y anidado en un bean interno?

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

WebFlux y reactivo

S202

RESPUESTA

Mono<T>: 0 o 1 elemento (o vacío/error) — la versión reactiva de Optional/Future.
 Flux<T>: 0..N elementos con señal de completado y backpressure.
 NADA se ejecuta sin SUBSCRIBIRSE: flux.map(...) devuelve otro flujo frío; el trabajo arranca en subscribe/return al framework.
 Operaciones declarativas: map/flatMap/filter/concatWith/merge.

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S201

RESPUESTA

MVC (bloqueante, 1 request = 1 hilo): óptimo para CRUD, JPA, ecosistema maduro. Con 200 hilos sostiene cientos de req/s.
 WebFlux (no bloqueante, pocos event-loop threads): gana con MUCHA concurrencia + I/O (streaming, proxies, backpressure), no con más CPU.
 Regla: si tu stack es JPA/JDBC (bloqueante), WebFlux NO ayuda — jdbc bloquea el event loop. R2DBC para reactivo real.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S204

RESPUESTA

Nunca se crea si nadie lo pide.
 Con @Lazy en la inyección: el CONSUMIDOR recibe un proxy; el bean real se crea al PRIMER método llamado.
 Sin @Lazy pero con singleton eager (default): se crea al arrancar aunque nadie lo use.
 @Lazy en @Configuration(proxyBeanMethods) es otra cosa: proxies de métodos @Bean.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S203

RESPUESTA

NO. Spring inyecta en instancias de beans; los static pertenecen a la clase, y el contenedor no los toca (además fomenta estado global no testeable).
 Soluciones: ponerlo en un bean normal e inyectar, o pattern holder inicializado por el contexto (ApplicationContextHolder) solo como último recurso legado.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S206

RESPUESTA

NoUniqueBeanDefinitionException al arrancar (o al primer uso si lazy).
 Resuelve: @Primary en uno, @Qualifier en el punto de inyección, o nombre de parámetro = nombre del bean (por nombre).
 En @ConfigurationProperties-style: clases wrapper distintas para cada configuración.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S205

RESPUESTA

Sí: el ciclo completo (inyección, @PostConstruct) corre por CADA creación.
 Pero @PreDestroy NO se llama: Spring no rastrea prototipos tras entregarlos — cleanup manual por el cliente (destroy callback o try-with-resources).
 Por eso prototipos con recursos necesitan destroyMethod o gestión explícita.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S208

RESPUESTA

Método SOBREENSCRIBE el de la clase (regla de resolución: el más específico).
 Métodos SIN anotación propia heredan la de la clase.
 Beans internos (nueva clase anidada como @Component) Sí son beans con su propio proxy → transacción funciona.
 Cuidado con clases de configuración anidadas no anotadas: son configuración, no beans.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S207

RESPUESTA

Casi nunca: service locator pattern oculta dependencias (constructor no muestra lo que usa), imposible de mockear limpio, y ancla al contexto.
 Excepciones aceptables: código legacy no-manageado, resolución dinámica de plugins (con ObjectProvider mejor), herramientas.
 Preferir: inyección explícita siempre.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S209

¿Diferencia entre @ComponentScan con lazy-init global y @Lazy?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S210

¿Cómo inyectarías un valor que cambia en runtime (refresh)?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S211

¿Cuál es la diferencia entre filtros de servlet y @WebFilter vs FilterRegistrationBean?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S212

¿Cómo arrancar código justo después de levantar el contexto?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S213

¿Cómo harías un feature flag en Spring sin librería externa?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S214

¿Qué es un 'fat jar' de Boot y por qué importa en contenedores?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S215

¿Diferencia entre @RequestParam, @RequestPart y @ModelAttribute para forms?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S216

¿Cómo proteger contra N+1 sin notarlo en dev?

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Trampas de entrevista

S210

RESPUESTA

@ConfigurationProperties + @RefreshScope + actuador/refresh (Cloud Config) — o re-lectura del Environment.
 Con Kubernetes: spring-cloud-kubernetes reloda ConfigMaps (polling/eventos).
 Anti-patrón: @Value cacheado sin refresh (nunca cambia).
 Para config MUY dinámica: feature flags service (LaunchDarkly/Unleash) o DB-backed properties con caché corta.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S209

RESPUESTA

spring.main.lazy-initialization=true: TODOS los beans se crean perezosamente — arranque rapidísimo, pero errores de config aparecen tarde y primer request más lento.
 @Lazy: quirúrgico por bean.
 En prod se prefiere eager (fail-fast); lazy global útil para arranques en dev o lambdas con cold start.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S212

RESPUESTA

CommandLineRunner (args crudos) / ApplicationRunner (ApplicationArguments parseados) — beans ejecutados al final del arranque.
 @EventListener(ApplicationReadyEvent.class): equivalente desacoplado.
 SmartLifecycle para procesos con start/stop (consumers).
 @PostConstruct NO: corre durante la creación del bean (contexto aún incompleto).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S211

RESPUESTA

@WebFilter: anotación servlet nativa — requiere @ServletComponentScan y da poco control.
 FilterRegistrationBean (recomendado en Boot): bean que registra el Filter con URL patterns, ORDEN explícito y activación condicional.
 Orden importa: auth debe correr antes que logging de negocio; security filter chain corre antes de los tuyos si así lo ordenas.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S214

RESPUESTA

Jar único con TODAS las dependencias (BOOT-INF/lib) + loader propio (java -jar directo).
 Contra: rebuild = reenviar 80MB aunque solo cambió tu código → layered extraction resuelve (capas por cambio de dependencias).
 Alternativa: war en Tomcat externo (legado).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S213

RESPUESTA

@ConfigurationProperties(prefix="features") record Features(boolean nuevoCheckout, ...) + consumidor inyecta Features.
 Toggle por properties/env → redeploy o refresh.
 Con @ConditionalOnProperty para beans enteros (activar/desactivar implementaciones).
 Nivel siguiente: flags runtime (Unleash/LaunchDarkly) sin redeploy, con % de usuarios.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S216

RESPUESTA

Tests de conteo: datasource-proxy o Hibernate statistics (assert selectCount == esperado).
 hibernate.generate_statistics + log WARN de hibernate (session-metrics).
 En prod: métricas por query, trace con spans por SELECT (Micrometer + JDBC observer).
 Code review: toda relación navegada en loop = sospechoso.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S215

RESPUESTA

@RequestParam: campo individual del form/query.
 @ModelAttribute: objeto completo binded por setters (form-encode).
 @RequestPart: parte de un MULTIPART con converter propio (JSON + archivo juntos).
 Multipart simple: @RequestParam MultipartFile alcanza.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S217

¿Cómo probarías que un @Scheduled corre con la frecuencia correcta?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S218

¿Cómo migrarías una propiedad de @Value a @ConfigurationProperties sin romper nada?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S219

¿Cómo expondrías 2 versiones de la misma API sin duplicar todo?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S220

¿Qué pasa si una ExceptionHandler lanza otra excepción?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S221

¿Cómo validar límites de rate por usuario SIN librerías?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S222

Un archivo de 500MB rompe el upload: config correcta

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S223

¿Diferencia entre @Scheduled y un Queue worker (Cloud Tasks)?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S224

¿Cómo loguear el SQL real con parámetros en prod sin reventar performance?

INTERMEDIO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

Trampas de entrevista

S218

RESPUESTA

- 1) Crear el POJO con el campo igual (mismo nombre relaxed binding).
- 2) Registrar (@Component o @EnableConfigurationProperties).
- 3) Migrar consumidores uno a uno (compilación guía).
- 4) Tests de binding con ApplicationContextRunner.
- 5) Deprecar la clave vieja con placeholder de compatibilidad temporal: @Value("\${vieja:\${nueva:default}}").

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S217

RESPUESTA

No esperes tiempos reales: extraer la lógica a un método y testearlo directo.
Para el wiring: Awaitility con ventana corta, o SchedulerTask de test con fixedDelay mínimo y count.
En cluster: verificar ShedLock (una sola ejecución con 2 instancias en test de integración).
Métricas de ejecución en prod (contador + timer).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S220

RESPUESTA

Spring cae al handler de último recurso (ErrorAttributes/BasicErrorController) → 500 genérico; la excepción original se loguea por el stack.
El catch-all Exception.class del advice DEBE ser infalible: construir respuesta simple sin lógica que pueda fallar (y loguear AMBAS excepciones).
Nunca confiar en datos del request ahí (pueden ser la causa).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S219

RESPUESTA

Núcleo único (service) + controllers por versión que mapean DTOs distintos: /v1 y /v2 comparten la lógica; transformadores versionados.
Solo duplicar lo que ROMPIÓ compatibilidad.
Deprecación: header Sunset + métricas de uso por versión para saber cuándo retirar v1.
Nunca: fork del service por versión.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S222

RESPUESTA

spring.servlet.multipart.max-file-size=600MB y max-request-size mayor; gateway/ingress también tiene su límite (client_max_body_size en nginx, timeout de LB).
Streaming: guardar el InputStream directo a disco/S3 (nunca byte[] completo).
Alternativa para archivos grandes: upload directo a Cloud Storage con signed URL — tu API solo firma y registra.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S221

RESPUESTA

Filtro/interceptor + Redis INCR con EXPIRE (ventana fija) o script Lua para sliding window: clave por userId+minuto.
429 con Retry-After; headers X-RateLimit-Limit/Remaining informativos.
Distribuido porque Redis comparte entre réplicas.
Producción seria: Bucket4j + Redis o el rate limiter del gateway.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S224

RESPUESTA

Nunca show-sql en prod. Opciones: datasource-proxy/psql wrapper con log SLOW queries (>200ms) + parámetros en DEBUG activado dinámicamente (actuador loggers level cambian SIN restart).
PG: pg_stat_statements / slow query log del motor.
Tracing: span por query con el statement (sampling) — equilibrando PII.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S223

RESPUESTA

@Scheduled: periódico fijo, en-proceso, no distribuido sin lock, jobs 'batch'.
Queue worker (Tasks/PubSub): por-evento, con retry persistido, dead-letter, visibilidad del fallo, escala horizontal natural.
Regla: cron interno para mantenimiento simple; colas para trabajo que NO puede perderse.

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S225

flatMap vs concatMap y threading en WebFlux

AVANZADO

SPRING BOOT

WebFlux y reactivo

S226

Backpressure en una frase + estrategias

AVANZADO

SPRING BOOT

WebFlux y reactivo

S227

R2DBC: JPA pero reactivo — qué cambia

AVANZADO

SPRING BOOT

WebFlux y reactivo

S228

SSE vs WebSocket vs Long-polling

AVANZADO

SPRING BOOT



Trampas de entrevista

S229

**¿Qué mide y qué no
spring.jpa.open-in-view?**

AVANZADO

SPRING BOOT

Trampas de entrevista

S230

**Si debes garantizar que un email se envía
EXACTAMENTE una vez tras crear un
pedido, ¿qué haces?**

AVANZADO

SPRING BOOT

Trampas de entrevista

S231

**¿Cuándo un @Bean de @Configuration se
llama 'lite mode' y qué pierde?**

AVANZADO

REVERSO — cada respuesta queda al dorso de su pregunta (misma numeración)

SPRING BOOT

WebFlux y reactivo

S226

RESPUESTA

El consumidor avisa cuántos elementos puede procesar (request(n)) para que el productor no lo ahogue.
Estrategias: buffer (acumular con límite), drop/latest (descartar), onError (abortar), limitRate en pipelines.
La cola con límite es backpressure aplicado a mensajes.

AVANZADO

SPRING BOOT

WebFlux y reactivo

S225

RESPUESTA

flatMap: ejecuta en PARALELO (intercala elementos según llegan) — para llamadas independientes.
concatMap: SECUENCIAL, respeta orden.
publishOn/subscribeOn cambian de scheduler (boundedElastic para código bloqueante: blockHound detecta violaciones).
Nunca bloquear el event loop: JDBC, sleep, CompletableFuture.get → boundedElastic.

AVANZADO

SPRING BOOT

WebFlux y reactivo

S228

RESPUESTA

SSE (text/event-stream): unidireccional servidor→cliente, reconexión automática, HTTP puro — notificaciones, dashboards.
WebSocket: bidireccional full-duplex (STOMP sobre WS en Spring) — chat, edición colaborativa.
Long-polling: legado/fallback.
Escalar WS/SSE: sesiones pegajosas o pub/sub compartido (Redis) entre réplicas.

AVANZADO

SPRING BOOT

WebFlux y reactivo

S227

RESPUESTA

spring-boot-starter-data-r2dbc + DatabaseClient/repositorios reactivos (ReactiveCrudRepository).
SQL no bloqueante real (driver por BD), transacciones con ReactiveTransactionManager.
Sin lazy loading (JOIN FETCH no existe): proyecciones/queries explícitas.
Menos conveniente que JPA; usar solo en pipelines reactivos.

AVANZADO

SPRING BOOT

Trampas de entrevista

S230

RESPUESTA

Exactamente una vez es un mito distribuido → lo asegurable: AL MENOS una vez + idempotencia.
Diseño: outbox (evento PedidoCreado en tabla, misma tx) → worker publica → consumidor envía email guardando dedup por eventId (si ya existe, skip).
Reintentos con backoff + dead letter para fallos persistentes.

AVANZADO

SPRING BOOT



Trampas de entrevista

S229

RESPUESTA

OSIV (default TRUE): mantiene la sesión de Hibernate abierta durante TODO el request → los controllers pueden tocar relaciones lazy 'gratis'.
Problemas: conexiones del pool retenidas durante serialización (menos throughput), queries sorpresa en la vista, N+1 invisible.
Prod recomendado: false + DTOs/JOIN FETCH explícitos. Boot lo avisa al arrancar.

AVANZADO

SPRING BOOT

Trampas de entrevista

S231

RESPUESTA

@Bean en una clase SIN @Configuration (en @Component por ejemplo) = lite mode: los métodos NO pasan por proxy CGLIB → llamar un @Bean desde otro NO reutiliza el singleton (crea instancia nueva cada vez).
Con @Configuration completo: intercepta llamadas internas a métodos @Bean y devuelve el singleton.
Pierde inter-bean references — bug sutil y clásico.

AVANZADO