Prueba técnica teórica – Desarrollador Fullstack Java

Cristian Camilo Aya Álvarez

29 de julio de 2025

## ¿Cuál es el ciclo de vida de una aplicación Spring Boot y cómo se gestiona el contexto de aplicación?

El ciclo de vida de una aplicación Spring Boot inicia con la ejecución del método main, habitualmente mediante SpringApplication.run(). Durante este proceso, se desarrollan las siguientes etapas principales:  
  
- Inicialización del contexto de aplicación (ApplicationContext): se cargan los beans, propiedades, configuraciones y se escanean anotaciones como @Component, @Service o @Repository.  
- Inyección de dependencias: se aplica mediante anotaciones como @Autowired, @Value o a través de constructores.  
- Ejecución de eventos: se generan eventos como ApplicationStartedEvent o ApplicationReadyEvent, permitiendo ejecutar lógica personalizada.  
- Ciclo de vida de los beans: los beans pueden pasar por métodos como @PostConstruct, InitializingBean, @PreDestroy o DisposableBean.  
  
El ApplicationContext actúa como contenedor central que administra los beans, sus dependencias y su ciclo de vida, manteniéndose activo durante toda la ejecución de la aplicación.

## ¿Cómo se gestiona la configuración en Spring Boot para diferentes entornos (desarrollo, prueba y producción)?

Spring Boot facilita la configuración por entornos mediante el uso de perfiles. Esta gestión se realiza con archivos de propiedades diferenciados, como:  
  
- application-dev.properties  
- application-test.properties  
- application-prod.properties  
  
El perfil activo puede definirse de las siguientes maneras:  
- A través de variables de entorno (SPRING\_PROFILES\_ACTIVE=dev)  
- En el archivo application.properties (spring.profiles.active=dev)  
- Desde la línea de comandos (--spring.profiles.active=dev)  
  
Además, se recomienda externalizar propiedades sensibles utilizando herramientas como Spring Cloud Config, HashiCorp Vault o mediante variables de entorno.

## ¿Qué estrategias se implementan para garantizar la seguridad en una aplicación Spring Boot?

La seguridad se implementa principalmente mediante Spring Security, utilizando diferentes enfoques según el tipo de aplicación:  
  
- Autenticación:  
 - En memoria (para ambientes de prueba)  
 - Basada en base de datos, usando UserDetailsService  
 - Con tokens JWT para aplicaciones REST sin estado  
 - Mediante OAuth2 o OpenID Connect para integración con proveedores externos  
  
- Autorización:  
 - Mediante anotaciones como @PreAuthorize o @Secured  
 - Configuración de rutas mediante SecurityFilterChain  
  
- Buenas prácticas:  
 - Uso de algoritmos como BCrypt para el cifrado de contraseñas  
 - Protección contra CSRF (cuando aplica)  
 - Configuración de encabezados de seguridad (como X-Frame-Options, Content-Security-Policy)  
 - Registro de auditoría y trazabilidad de eventos

## ¿Cómo se implementa un servicio RESTful en Spring Boot?

La construcción de servicios REST se realiza aplicando las siguientes anotaciones:  
  
- @RestController: combina @Controller y @ResponseBody, indicando que se trata de un controlador REST.  
- Métodos específicos como @GetMapping, @PostMapping, @PutMapping y @DeleteMapping para definir las operaciones HTTP.  
- @RequestBody: para deserializar el cuerpo de la solicitud en objetos DTO.  
- @PathVariable y @RequestParam: para extraer parámetros desde la URL.  
- @ResponseStatus: para personalizar los códigos de respuesta HTTP.  
  
También se utilizan DTOs para la entrada y salida de datos, evitando exponer directamente las entidades del dominio. La validación se realiza mediante @Valid, en combinación con restricciones como @NotNull y @Size del paquete javax.validation. Finalmente, se recomienda implementar manejo global de errores con @ControllerAdvice y excepciones personalizadas.

## ¿Cuál es el ciclo de vida de un componente en Angular?

Angular proporciona un conjunto de hooks que definen el ciclo de vida de un componente:  
  
1. ngOnChanges: se ejecuta cuando cambian las propiedades de entrada.  
2. ngOnInit: ideal para inicializar variables o realizar llamadas a servicios externos.  
3. ngDoCheck: permite realizar detección manual de cambios.  
4. ngAfterContentInit y ngAfterContentChecked: se invocan después de proyectar contenido mediante ng-content.  
5. ngAfterViewInit y ngAfterViewChecked: se activan tras cargar las vistas hijas (por ejemplo, mediante @ViewChild).  
6. ngOnDestroy: se ejecuta justo antes de que el componente sea destruido, útil para liberar recursos o cancelar suscripciones.

## ¿Qué es un sistema de ruteo en Angular y cómo funciona?

El enrutamiento en Angular se implementa con el módulo RouterModule, que permite mapear rutas a componentes específicos. Para ello:  
  
- Se define una tabla de rutas:  
const routes: Routes = [  
 { path: 'home', component: HomeComponent },  
 { path: 'detalle/:id', component: DetalleComponent },  
 { path: '', redirectTo: 'home', pathMatch: 'full' }  
];  
  
- Se importa RouterModule.forRoot(routes) en el módulo principal (AppModule).  
- La navegación se realiza con directivas como [routerLink] o con métodos como this.router.navigate.  
- Se extraen parámetros de ruta con ActivatedRoute.  
- El sistema permite incorporar guards como CanActivate o CanDeactivate para controlar el acceso a rutas protegidas.

## ¿Qué es el patrón Observable y cómo se utiliza en Angular?

El patrón Observable, parte de la librería RxJS, permite trabajar con flujos de datos asincrónicos y reactivos. En Angular se emplea en varios contextos:  
  
- Llamadas HTTP con HttpClient, que retorna Observables.  
- Escucha de eventos del DOM.  
- Formularios reactivos, mediante FormControl.valueChanges.  
- Comunicación entre componentes usando Subject o BehaviorSubject.  
  
Entre sus ventajas destacan:  
- Cancelación de suscripciones para evitar pérdidas de memoria.  
- Disponibilidad de operadores como map, filter, switchMap, entre otros.  
- Mejora en la composición de flujos complejos y evita la anidación excesiva en llamadas asíncronas.