**Cuestionario Evaluador del Curso**

**DEVOPS**

1. **¿Qué es DevOps?**

DevOps es una cultura y conjunto de prácticas que combina desarrollo de software (Dev) y operaciones de TI (Ops) para acortar el ciclo de vida del desarrollo de sistemas y proporcionar entrega continua con alta calidad. Promueve la colaboración, automatización y monitoreo continuo.

1. **Menciona las fases principales del ciclo de vida de DevOps.**
   * Planificación (Plan).
   * Desarrollo (Code).
   * Integración continua (Build).
   * Pruebas (Test).
   * Implementación (Deploy).
   * Operaciones (Operate).
   * Monitoreo (Monitor).
   * Retroalimentación (Feedback).
2. **¿Qué es JUnit y para qué se utiliza?**  
   JUnit es un framework de pruebas unitarias para Java que permite escribir y ejecutar pruebas automatizadas para validar el comportamiento del código.
3. **¿Cómo se crea un proyecto con IntelliJ para trabajar con JUnit?**
   * Crear un nuevo proyecto en IntelliJ.
   * Agregar la dependencia de JUnit en el archivo pom.xml (para Maven) o build.gradle (para Gradle).
   * Crear una clase de prueba en el directorio src/test/java.
4. **¿Qué anotaciones de JUnit se utilizaron para escribir pruebas unitarias?**
   * @Test: Define un método de prueba.
   * @BeforeEach: Ejecuta antes de cada prueba.
   * @AfterEach: Ejecuta después de cada prueba.
   * @BeforeAll: Ejecuta una vez antes de todas las pruebas.
   * @AfterAll: Ejecuta una vez después de todas las pruebas.
   * @DisplayName: Define un nombre legible para la prueba.
   * @Disabled: Ignora una prueba.
5. **Explica brevemente qué es TDD (Desarrollo Guiado por Pruebas).**  
   TDD es una metodología de desarrollo en la que primero se escriben pruebas unitarias para una funcionalidad y luego se implementa el código para que pase esas pruebas. El ciclo es: Red (escribir prueba fallida), Green (implementar código para pasar la prueba), Refactor (mejorar el código).
6. **¿Qué significa la anotación @DisplayName en JUnit?**  
   @DisplayName permite asignar un nombre descriptivo y legible a una prueba, lo que facilita la comprensión de su propósito.
7. **¿Para qué sirve la anotación @Disabled en JUnit?**  
   @Disabled se utiliza para desactivar temporalmente una prueba sin eliminarla del código.
8. **¿Qué es assertThrows y cómo se utiliza en JUnit?**  
   assertThrows verifica que un bloque de código lance una excepción específica. Ejemplo:

*assertThrows(IllegalArgumentException.class, () -> { metodoQueLanzaExcepcion(); });*

1. **Describe un ejemplo práctico de cómo se implementa TDD para operaciones de débito y crédito.**
   * Escribir una prueba para una operación de crédito que incremente el saldo.
   * Implementar el código para pasar la prueba.
   * Escribir una prueba para una operación de débito que decremente el saldo.
   * Implementar el código para pasar la prueba.
   * Refactorizar el código para mejorar su estructura.
2. **¿Qué es Mockito y cuál es su propósito?**  
   Mockito es un framework de mocking para Java que permite crear objetos simulados (mocks) para probar componentes en aislamiento.
3. **¿Cómo se configura un proyecto con JUnit y Mockito en IntelliJ?**
   * Agregar las dependencias de JUnit y Mockito en pom.xml (Maven) o build.gradle (Gradle).
   * Configurar las clases de prueba con anotaciones como @Mock y @InjectMocks.
4. **¿Qué es una dependencia mock y cómo se crea con Mockito?**  
   Una dependencia mock es un objeto simulado que imita el comportamiento de una dependencia real. Se crea usando Mockito.mock() o la anotación @Mock.
5. **¿Para qué se utiliza el método verify en Mockito?**  
   verify se utiliza para asegurarse de que un método en un mock fue llamado con los parámetros esperados. Ejemplo:

*verify(mockObject).metodoEspecifico(parametro);*

1. **Explica el uso de las anotaciones @Mock y @InjectMocks.**
   * @Mock: Crea un objeto mock de una dependencia.
   * @InjectMocks: Inyecta los mocks en la clase bajo prueba.
2. **¿Cómo se prueba un repositorio utilizando Mockito?**
   * Crear un mock del repositorio.
   * Configurar el comportamiento del mock usando when().thenReturn().
   * Ejecutar métodos de la clase de servicio que usa el repositorio.
   * Verificar interacciones con verify().
3. **¿Qué es Spring Boot y cómo se relaciona con las pruebas unitarias?**  
   Spring Boot es un framework para desarrollar aplicaciones Java con configuración automática. Facilita las pruebas unitarias e integración con herramientas como JUnit y Mockito.
4. **¿Qué anotaciones se utilizan para inyectar dependencias en Spring Boot?**
   * @Autowired: Inyecta dependencias automáticamente.
   * @Component, @Service, @Repository: Definen beans para inyección.
5. **¿Cómo se configuran las clases de servicio en un proyecto Spring Boot?**
   * Anotar la clase con @Service.
   * Usar @Autowired para inyectar dependencias.
6. **¿Qué es MockMvc y para qué se utiliza en Spring Boot?**  
   MockMvc es una herramienta para probar controladores de Spring MVC simulando solicitudes HTTP.
7. **¿Cómo se configura Swagger en un proyecto Spring Boot?**
   * Agregar la dependencia de Swagger en pom.xml.
   * Configurar un bean de Docket en una clase de configuración.
8. **¿Qué son los endpoints y cómo se prueban con MockMvc?**  
   Los endpoints son rutas HTTP expuestas por un controlador. Se prueban con MockMvc usando métodos como perform() y andExpect().
9. **¿Qué anotación se utiliza para escribir pruebas de controladores con MockMvc?**  
   @WebMvcTest: Configura el contexto de Spring para pruebas de controladores.
10. **¿Qué es la cobertura de código y cómo se mide?**  
    La cobertura de código mide el porcentaje de código ejecutado durante las pruebas. Se mide con herramientas como JaCoCo.
11. **¿Qué es WebTestClient y cómo se utiliza para probar servicios REST?**  
    WebTestClient es una herramienta para probar endpoints REST de forma reactiva. Ejemplo:

*webTestClient.get().uri("/endpoint").exchange().expectStatus().isOk();*

1. **Explica cómo escribirías una prueba de integración para listar elementos usando WebTestClient.**
   * Configurar WebTestClient.
   * Realizar una solicitud GET al endpoint.
   * Verificar el estado de la respuesta y el contenido.
2. **¿Qué diferencia hay entre WebTestClient y TestRestTemplate?**  
   WebTestClient es reactivo y más moderno, mientras que TestRestTemplate es síncrono y tradicional.
3. **¿Cómo se configura TestRestTemplate para probar servicios REST?**
   * Crear una instancia de TestRestTemplate.
   * Usar métodos como getForObject() o postForEntity().
4. **Escribe un ejemplo de cómo probarías un endpoint de eliminación con TestRestTemplate.**

*ResponseEntity<Void> response = testRestTemplate.exchange("/endpoint/{id}", HttpMethod.DELETE, null, Void.class, id);*

*assertEquals(HttpStatus.NO\_CONTENT, response.getStatusCode());*

1. **¿Qué es Jenkins y para qué se utiliza?**  
   Jenkins es una herramienta de integración continua que automatiza la construcción, prueba y despliegue de aplicaciones.
2. **¿Cómo se descarga e instala Jenkins?**
   * Descargar el archivo WAR desde el sitio oficial.
   * Ejecutar java -jar jenkins.war.
   * Acceder a http://localhost:8080 para completar la instalación.
3. **Menciona tres plugins importantes que se pueden usar en Jenkins.**
   * Git Plugin: Integración con Git.
   * Maven Integration Plugin: Soporte para proyectos Maven.
   * Pipeline: Creación de pipelines.
4. **¿Qué es un proyecto de estilo libre (Freestyle) en Jenkins?**  
   Un proyecto configurable manualmente que permite definir pasos de construcción, triggers y post-acciones.
5. **Explica cómo funcionan los Build Triggers en Jenkins.**  
   Los Build Triggers activan construcciones automáticamente, por ejemplo, al detectar cambios en un repositorio Git.
6. **¿Cómo se envían parámetros a un Job en Jenkins?**  
   Usando parámetros definidos en la configuración del Job, como String, Choice, o Boolean.
7. **¿Qué herramientas se pueden integrar con Jenkins para proyectos Java y Maven?**
   * Git, Maven, JUnit, SonarQube, Docker.
8. **¿Qué son las vistas en Jenkins y cómo se crean?**  
   Las vistas son paneles personalizados para organizar Jobs. Se crean desde la interfaz de Jenkins.
9. **Explica qué es una Pipeline en Jenkins.**  
   Una Pipeline es un conjunto de pasos automatizados definidos en un archivo Jenkinsfile para construir, probar y desplegar aplicaciones.
10. **¿Qué son las directivas en una Pipeline de Jenkins?**  
    Instrucciones en un Jenkinsfile, como pipeline, agent, stages, y steps.
11. **¿Cómo se implementa el paralelismo en una Pipeline de Jenkins?**  
    Usando la directiva parallel dentro de un stage. Ejemplo:

*parallel {*

*stage('Test 1') { steps { ... } }*

*stage('Test 2') { steps { ... } }*

*}*

1. **Menciona un plugin visual útil en Jenkins y describe su función.**  
   Blue Ocean: Proporciona una interfaz visual moderna para crear y monitorear pipelines.
2. **¿Qué son las credenciales en Jenkins y cómo se configuran?**  
   Las credenciales son datos sensibles (contraseñas, tokens) almacenados de forma segura. Se configuran en "Manage Jenkins" > "Manage Credentials".
3. **¿Cómo se integra Jenkins con herramientas de control de versiones como Git?**  
   Usando el Git Plugin y configurando el repositorio en la configuración del Job.
4. **Explica cómo configurarías una Pipeline para desplegar una aplicación Spring Boot.**
   * Definir un Jenkinsfile con etapas para construir, probar y desplegar.
   * Usar herramientas como Docker o Kubernetes para el despliegue.