

*Dibujo con letras

Descripción generada automáticamente con confianza media*

*SUDAMERICANA DE SOFTWARE S.A.*

*OFICINA DE DIRECCIÓN DE PROYECTOS*

*LÍNEA DE NEGOCIO:*

*Desarrollo*

*CAPACITACIÓN*

*Backend – Spring Boot*

*CLIENTE*

*SUDAMERICANA DE SOFTWARE*

*PROPIEDAD INTELECTUAL*

*Sudamericana de Software S.A.*

# INTRODUCCIÓN

El presente documento corresponde a la prueba de evaluación de conocimientos en desarrollo backend utilizando el framework Spring Boot. El objetivo de este examen es comprobar las competencias técnicas del participante en la creación de servicios REST, gestión de datos, implementación de buenas prácticas de programación y uso adecuado de herramientas y dependencias propias de Spring Boot. A través de los ejercicios propuestos, se evaluará la capacidad para diseñar soluciones backend eficientes, estructuradas y acordes a los estándares de desarrollo empresarial.

# OBJETIVO DEL CURSO

* Evaluar la lógica programática del participante mediante ejercicios de algoritmos, validando su capacidad de análisis y resolución de problemas.
* Comprobar los conocimientos teóricos relacionados con el desarrollo backend, incluyendo temas de diseño de servidor, despliegue de aplicaciones y arquitectura de software.
* Verificar el dominio práctico de la herramienta Spring Boot, su correcta utilización en la creación de servicios, configuración de proyectos y aplicación de buenas prácticas en el desarrollo de soluciones empresariales.

# EVALUACIÓN

**PARTE TEORICA**

1. ¿Cuál de las siguientes opciones describe mejor el patrón Repository en Spring Data JPA?
2. Permite exponer endpoints REST automáticamente
3. Se utiliza para conectarse directamente a APIs externas
4. Es una capa que abstrae el acceso y las operaciones sobre la base de datos
5. Gestiona la seguridad y autenticación de la aplicación
6. ¿Qué anotación se utiliza en Spring Boot para indicar que una clase es un servicio?
7. @Repository
8. @RestController
9. @Service
10. @ComponentScan
11. ¿Qué significa la anotación @Autowired en Spring Boot?
12. Define el punto de inicio de la aplicación
13. Genera un bean a partir de una clase automáticamente
14. Inyecta una dependencia en un componente gestionado por Spring
15. Declara un endpoint REST en un controlador
16. ¿Cuál de las siguientes opciones NO es una ventaja de usar contenedores Docker en el desarrollo backend?
17. Portabilidad entre entornos
18. Aislamiento de aplicaciones
19. Uso menor de recursos que máquinas virtuales
20. Mejora directamente el rendimiento de ejecución de código Java
21. En Spring Boot, ¿para qué sirve la anotación @Entity?
22. Define un bean en el contexto de Spring
23. Indica que una clase representa una tabla en la base de datos
24. Declara un endpoint REST
25. Configura la inyección de dependencias
26. ¿Cuál es el propósito principal de utilizar Spring Boot Starters en un proyecto?
27. Generar código fuente automáticamente
28. Gestionar dependencias comunes y configurarlas fácilmente
29. Ejecutar aplicaciones como scripts de consola
30. Generar documentación API automáticamente

**PARTE PRACTICA**

REGLAS GENERALES:

En el repositorio GIT asignado se debe seguir la siguiente jerarquía de carpetas, la cual se deben subir los archivos relacionados de este trabajo siguiendo la estructura:

* + PRUEBA\_LOGICA
    - ALGORITMO 1
    - ALGORITMO 2
  + PRUEBA\_PRACTICA
* DOCS
* PROYECTO
* SCRIPTS
* DDL

Nota: Si no cumple con la estructura del repositorio será penalizado con una reducción de puntos en su nota final.

**REGLAS DE CONFIGURACIÓN PARA EL PROYECTO SPRING BOOT**

1. Versión de Spring Boot > 3.0.0 (No deberá ser SNAPSHOT)

2. Versión de Java >= 1.8

3. Nombre del proyecto: ms-comp-prueba-[Nombre]-[Apellido]

4. Tipo de compilador: Maven

5. Group: ec.sasf

6. Artifact: Deberá llamarse como está en el punto 3.

7. Package name: ec.sasf.prueba.[Nombre].[Apellido]

**ALGORITMOS A DESARROLLAR:**

**Problema 1. Subarreglo con suma objetivo**

Enunciado:

Dado un arreglo de números enteros y un número objetivo, retorna true si existe un subarreglo contiguo cuya suma sea igual al número objetivo.

Ejemplo:

Input: arr = [1, 4, 20, 3, 10, 5], objetivo = 33

Output esperado: true (porque 20 + 3 + 10 = 33)

Input: arr = [1, 4, 0, 0, 3, 10, 5], objetivo = 7

Output esperado: true

Input: arr = [1, 4], objetivo = 0

Output esperado: false

**Problema 2. Suma de dos números sin repetir índice**

Enunciado:

Dado un arreglo de enteros y un número objetivo, encuentra dos elementos del arreglo cuya suma sea igual al objetivo, y retorna sus índices. No puedes usar el mismo índice dos veces.

Ejemplo:

Input: nums = [2, 7, 11, 15], objetivo = 9

Output esperado: [0, 1]

Input: nums = [3, 2, 4], objetivo = 6

Output esperado: [1, 2]

**PROBLEMA DE DESARROLLO**

**Sistema de Gestión de Alquiler de Autos**

Enunciado:

Desarrolla un sistema backend en Spring Boot para gestionar el alquiler de autos a clientes. El sistema debe permitir registrar autos, clientes, crear contratos de alquiler, consultar disponibilidad y calcular multas por devoluciones tardías**.**

**Entidades principales:**

**Auto**

* id: Long
* marca: String
* modelo: String
* anio: Integer
* placa: String
* disponible: Boolean

**Cliente**

* id: Long
* nombre: String
* cedula: String
* correo: String
* telefono: String

**ContratoAlquiler**

* id: Long
* auto: Auto
* cliente: Cliente
* fechaInicio: Date
* fechaFin: Date
* fechaDevolucion: Date (puede ser null hasta que el auto se devuelva)
* multa: Double

**Reglas de negocio:**

* Un auto solo puede estar alquilado si está disponible.
* Al crear un contrato, el auto debe marcarse como disponible = false.
* Al devolver un auto, se registra fechaDevolucion y se calcula una multa si se devuelve después de la fechaFin.
* La multa se calcula como $20 por cada día de retraso.

**APIs mínimas requeridas:**

1. Registrar un auto

* POST /autos
* Input: marca, modelo, anio, placa
* Output: auto registrado

1. Registrar un cliente

* POST /clientes
* Input: nombre, cedula, correo, telefono
* Output: cliente registrado

1. Crear un contrato de alquiler

* POST /alquileres
* Input: idAuto, idCliente, fechaInicio, fechaFin
* Output: contrato creado o error si el auto no está disponible

1. Registrar la devolución de un auto

* PUT /alquileres/{id}/devolucion
* Input: fechaDevolucion
* Output: multa calculada y auto marcado como disponible

1. Consultar disponibilidad de autos

* GET /autos/disponibles
* Output: lista de autos con disponible = true

1. Listar contratos con multa activa

* GET /alquileres/multas
* Output: lista de contratos cuya fechaDevolucion excedió la fechaFin y multa > 0
* Modularizar y cargar de manera dinámica secciones clave de la aplicación web

