

*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 es la anotación principal para iniciar una aplicación Spring Boot?

a) @EnableAutoConfiguration

b) @SpringBootApplication

c) @ComponentScan

d) @EnableJpaRepositories

2. ¿Qué anotación define una clase como controlador REST en Spring Boot?

a) @Service

b) @RestController

c) @Component

d) @Configuration

3. ¿Cuál es la función de Spring Boot Starter?

a) Inicializar un servidor remoto

b) Iniciar el sistema operativo

c) Simplificar la gestión de dependencias

d) Compilar código Java

4. ¿Para qué sirve @Autowired en Spring Boot?

a) Crear una tabla en base de datos

b) Especificar una dependencia manualmente

c) Inyectar automáticamente una dependencia

d) Iniciar una tarea programada

5. ¿Qué archivo se usa para configuración en Spring Boot?

a) config.yaml

b) spring.xml

c) application.properties o application.yml

d) setup.json

**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. Primer número faltante positivo**

**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. Longitud del subarreglo más largo sin elementos repetidos**

Enunciado:

Dado un arreglo de personas, cada una representada por un objeto con los siguientes campos:

* + nombre: String
  + edad: Integer
  + genero: String ("M" para masculino, "F" para femenino)
  + actividad: String (puede ser "estudia", "trabaja", "jubilado" o "desempleado")

Clasifica a las personas en un mapa estructurado con el siguiente formato:

{

"menor": {

"M": [...],

"F": [...]

},

"adulto": {

"M": [...],

"F": [...]

},

"mayor": {

"M": [...],

"F": [...]

}

}

Cada lista debe contener los nombres de las personas del grupo correspondiente.

Además, filtra y excluye del resultado final a cualquier persona que esté marcada como "desempleado" en el campo actividad.

Reglas de clasificación por edad:

"menor": edad < 18

"adulto": 18 ≤ edad ≤ 60

"mayor": edad > 60

**Ejemplo de entrada:**

[

{ nombre: "Ana", edad: 17, genero: "F", actividad: "estudia" },

{ nombre: "Luis", edad: 25, genero: "M", actividad: "trabaja" },

{ nombre: "Mario", edad: 70, genero: "M", actividad: "jubilado" },

{ nombre: "Sofía", edad: 22, genero: "F", actividad: "desempleado" }

]

**Output esperado:**

{

"menor": {

"M": [],

"F": ["Ana"]

},

"adulto": {

"M": ["Luis"],

"F": []

},

"mayor": {

"M": ["Mario"],

"F": []

}

}

**PROBLEMA DE DESARROLLO**

Sistema de Venta y Alquiler de Libros

Enunciado:

Desarrolla un sistema backend en Spring Boot para una librería que gestiona tanto la venta como el alquiler de libros físicos. El sistema debe permitir registrar libros, controlar el stock, gestionar clientes, procesar ventas y alquileres, y calcular recargos por retrasos en las devoluciones.

**Entidades principales:**

Libro

* id: Long
* titulo: String
* autor: String
* precioVenta: Double
* precioAlquiler: Double
* stockDisponible: Integer
* activo: Boolean

Cliente

* id: Long
* nombre: String
* cedula: String
* email: String

Venta

* id: Long
* cliente: Cliente
* libro: Libro
* fechaVenta: Date
* cantidad: Integer
* total: Double

Alquiler

* id: Long
* cliente: Cliente
* libro: Libro
* fechaInicio: Date
* fechaFin: Date
* fechaDevolucion: Date (puede ser null)
* multa: Double

**Reglas de negocio:**

* No se puede alquilar o vender un libro si el stockDisponible es menor que la cantidad solicitada.
* El total de una venta es precioVenta \* cantidad.
* Al registrar una venta o alquiler, el sistema debe descontar automáticamente el stock.
* Si el libro se devuelve después de la fechaFin, se aplica una multa de $1.50 por día de retraso.
* Los libros pueden ser vendidos y alquilados múltiples veces, pero solo si hay stock disponible.

**🌐 APIs mínimas requeridas:**

* POST /libros – Registrar un nuevo libro
* POST /clientes – Registrar un cliente
* POST /ventas – Registrar una venta de uno o más libros
* POST /alquileres – Registrar un nuevo alquiler
* PUT /alquileres/{id}/devolucion – Registrar la devolución de un libro alquilado (y calcular multa si aplica)
* GET /clientes/{id}/historial – Consultar el historial completo de ventas y alquileres de un cliente

**⭐ APIs BONUS sugeridas:**

* GET /libros/disponibles – Listar libros activos con stock disponible
* GET /reportes/top-vendidos – Obtener los libros más vendidos en el mes
* GET /reportes/top-alquilados – Obtener los libros más alquilados en el mes
* GET /alquileres/pendientes – Ver todos los alquileres activos que no han sido devueltos

**GET /alquileres/vencidos – Ver alquileres que superaron la fecha límite sin devolución**

* Modularizar y cargar de manera dinámica secciones clave de la aplicación web

