

*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 principal ventaja de utilizar microservicios frente a un monolito?
   1. Menor complejidad de implementación inicial
   2. Mayor dificultad de despliegue
   3. Escalabilidad independiente y desacoplamiento de servicios
   4. Reducción de la cantidad de servidores necesarios
2. **¿Qué principio describe mejor** Single Responsibility**?**
3. Cada clase debe tener múltiples responsabilidades para ser flexible
4. Cada clase o módulo debe tener una única razón para cambiar
5. Cada clase debe depender de interfaces y no de implementaciones
6. Cada método debe tener múltiples retornos
7. ¿Cuál es la función principal de un API Gateway en una arquitectura de microservicios?
   1. Almacenar datos en memoria compartida
   2. Realizar descubrimiento de servicios
   3. Centralizar la entrada de solicitudes y aplicar seguridad, balanceo y enrutamiento
   4. Ejecutar consultas SQL distribuidas
8. ¿Cuál de las siguientes opciones corresponde a comunicación asíncrona entre microservicios?
   1. REST con HTTP
   2. RPC (Remote Procedure Call)
   3. Mensajería mediante RabbitMQ o Kafka
   4. Llamadas directas usando Feign Client
9. ¿Cuál es el propósito de la inyección de dependencias en Spring Boot?
   1. Incrementar el acoplamiento entre clases
   2. Gestionar y reutilizar objetos, facilitando el mantenimiento y pruebas
   3. Reducir el tiempo de compilación
   4. Permitir la conexión directa a bases de datos NoSQL
10. ¿Qué patrón permite manejar fallas temporales de un servicio evitando saturación de solicitudes?
    1. Retry Pattern
    2. Circuit Breaker
    3. Singleton
    4. Builder Pattern
11. ¿Cuál es una desventaja de usar bases de datos NoSQL?
    1. No permiten escalar horizontalmente
    2. Carecen de esquemas flexibles
    3. No soportan transacciones ACID completas en todos los casos
    4. No permiten almacenar documentos o datos no estructurados
12. ¿Qué herramienta de Spring Cloud se usa para implementar Service Discovery?
    1. Spring Cloud Gateway
    2. Spring Data JPA
    3. Eureka
    4. Spring Security

**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 máxima (Kadane modificado)**

Enunciado:

Dado un arreglo de números enteros (que puede contener negativos), encuentra el subarreglo contiguo que tenga la suma máxima posible y devuelve dicha suma.

Ejemplo:

Input: arr = [-2,1,-3,4,-1,2,1,-5,4]

Output esperado: 6 (porque el subarreglo [4,-1,2,1] tiene suma máxima de 6)

Restricciones:

El arreglo tendrá al menos un elemento.

**Problema 2. Producto máximo de tres números**

Enunciado:

Dado un arreglo de números enteros, encuentra el producto máximo posible entre cualquier combinación de tres números del arreglo.

Ejemplo:

Input: arr = [-10, -10, 5, 2]

Output esperado: 500 (producto de -10, -10 y 5)

Restricciones:

El arreglo tendrá al menos tres elementos.

Considera números negativos y positivos.

**PROBLEMA DE DESARROLLO**

Sistema de Gestión de Órdenes para una Cafetería

Enunciado

Desarrolla un sistema backend para la gestión de una cafetería, utilizando Spring Boot, que permita administrar productos, clientes y órdenes de compra. El objetivo es evaluar tu dominio en la creación de proyectos backend con múltiples entidades y relaciones, aplicando buenas prácticas de desarrollo.

No se requiere implementar capa de seguridad ni autenticación para este ejercicio.

**Entidades principales**

Producto

* id: Long
* nombre: String
* precio: Double
* disponible: Boolean

Cliente

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

Orden

* id: Long
* cliente: Cliente
* productos: Lista de Productos
* fecha: Date
* total: Double

**Requerimientos funcionales (APIs mínimas)**

Implementa al menos 6 APIs REST con las siguientes funcionalidades:

1. Crear un producto

* Endpoint: POST /productos
* Input: nombre, precio, disponible
* Output: producto creado con su ID generado

1. Obtener todos los productos disponibles

* Endpoint: GET /productos/disponibles

1. Crear un cliente

* Endpoint: POST /clientes
* Input: nombre, correo, telefono
* Output: cliente creado con su ID generado

1. Crear una orden para un cliente existente

* Endpoint: POST /ordenes
* Input: ID del cliente, lista de IDs de productos
* Output: orden creada con el cálculo automático del total

1. Obtener todas las órdenes de un cliente específico

* Endpoint: GET /ordenes/cliente/{idCliente}

1. Actualizar la disponibilidad de un producto

* Endpoint: PUT /productos/{id}/disponibilidad
* Input: nuevo estado de disponibilidad (true/false)
* Modularizar y cargar de manera dinámica secciones clave de la aplicación web

