

# Prueba Técnica

## Arquitectura Microservicio (2023)

### Indicaciones generales

- Aplique todas las buenas prácticas, patrones Repository, etc que considere necesario (se tomará en cuenta este punto para la calificación).
- El manejo de entidades se debe manejar JPA / Entity Framework Core
- Se debe manejar mensajes de excepciones.
- Se debe realizar como mínimo dos pruebas unitarias de los endpoints.
- La solución se debe desplegar en Docker.
- Posterior a la entrega de este ejercicio, se estará agendando una entrevista técnica donde el candidato deberá defender la solución planteada.

### Herramientas y tecnologías utilizadas

- Java spring boot
- IDE de su preferencia
- Base de Datos Relacional
- Postman v9.13.2 (validador de API) / Karate DSL

### Complejidad por Seniority

**Nota: Considerar las siguientes indicaciones en base al perfil al que esta aplicando, Ejemplo: si es un perfil SemiSenior solo realizar lo indicado para este perfil.**

**Junior:** Implementar los diferentes endpoints para cumplir las funcionalidades: F1, F2, F3, no es mandatorio funcionalidades: F4, F5, F6.

**SemiSenior:** Separar en 2 microservicios, agrupando (Cliente, Persona) y (Cuenta, Movimientos) donde se contemple una comunicación asíncrona entre los 2 microservicios. Cumplir las funcionalidades: F1, F2, F3, F4, F5 deseable la funcionalidad F6.

**Senior:** Implementar en 2 microservicios, agrupando (Cliente, Persona) y (Cuenta, Movimientos) donde se contemple una comunicación asíncrona entre los 2 microservicios. Cumplir las funcionalidades F1, F2, F3, F4, F5, F6, F7  
La solución debe contemplar (no necesariamente implementado) factores como: rendimiento, escalabilidad, resiliencia.

### Generación de Api Rest “Application Programming Interface”

Manejar los verbos: Get, Post, Put, Push, delete

## Persona

- Implementar la clase persona con los siguientes datos: nombre, genero, edad, identificación, dirección, teléfono
- Debe manejar su clave primaria (PK)

## Cliente

- Cliente debe manejar una entidad, que herede de la clase persona.
- Un cliente tiene: clienteid, contraseña, estado.
- El cliente debe tener una clave única. (PK)

## Cuenta.

- Cuenta debe manejar una entidad
- Una cuenta tiene: número cuenta, tipo cuenta, saldo Inicial, estado.
- Debe manejar su Clave única

## Movimientos

- Movimientos debe manejar una entidad
- Un movimiento tiene: Fecha, tipo movimiento, valor, saldo
- Debe manejar su Clave única

## Funcionalidades del API

Los API's debe tener las siguientes operaciones:

**F1:** Generación de CRUDS (Crear, editar, actualizar y eliminar registros - Entidades: Cliente, Cuenta y Movimiento).

Los nombres de los endpoints a generar son:

- /cuentas
- /clientes
- /movimientos

**F2:** Registro de movimientos: al registrar un movimiento en la cuenta se debe tener en cuenta lo siguiente:

- Para un movimiento se pueden tener valores positivos o negativos.
- Al realizar un movimiento se debe actualizar el saldo disponible.
- Se debe llevar el registro de las transacciones realizadas

**F3:** Registro de movimientos: Al realizar un movimiento el cual no cuente con saldo, debe alertar mediante el siguiente mensaje "Saldo no disponible"

- Defina, según su expertise, la mejor manera de capturar y mostrar el error.

**F4:** Reportes: Generar un reporte de "Estado de cuenta" especificando un rango de fechas y cliente.

- Este reporte debe contener:

- Cuentas asociadas con sus respectivos saldos
- Detalle de movimientos de las cuentas
- El endpoint de que se debe utilizar para esto debe ser el siguiente:
  - (/reportes?fecha=rango fechas)
- El servicio del reporte debe retornar la información en formato JSON
  - Defina, según su expertise, la mejor manera de solicitar y retornar esta información.

**F5:** Pruebas unitarias: Implementar 1 prueba unitaria para la entidad de dominio Cliente.

**F6:** Pruebas de Integración: Implementar 1 prueba de integración.

**F7:** Despliegue de la solución en contenedores

## Casos de Uso (Ejemplos)

### 1. Creación de Usuarios.

Nombres	Dirección	Teléfono	Contraseña	estado
Jose Lema	Otavaló sn y principal	098254785	1234	True
Marianela Montalvo	Amazonas y NNUU	097548965	5678	True
Juan Osorio	13 junio y Equinoccial	098874587	1245	True

### 2. Creación de Cuentas de Usuario.

Numero Cuenta	Tipo	Saldo Inicial	Estado	Cliente
478758	Ahorro	2000	True	Jose Lema
225487	Corriente	100	True	Marianela Montalvo
495878	Ahorros	0	True	Juan Osorio
496825	Ahorros	540	True	Marianela Montalvo

### 3. Crear una nueva Cuenta Corriente para Jose Lema

Numero Cuenta	Tipo	Saldo Inicial	Estado	Cliente
585545	Corriente	1000	True	Jose Lema

### 4. Realizar los siguientes movimientos

Numero Cuenta	Tipo	Saldo Inicial	Estado	Movimiento
478758	Ahorro	2000	True	Retiro de 575

225487	Corriente	100	True	Deposito de 600
495878	Ahorros	0	True	Deposito de 150
496825	Ahorros	540	True	Retiro de 540

5. Listado de Movimiento, por fechas x usuario.

Fecha	Cliente	Numero Cuenta	Tipo	Saldo Inicial	Estado	Movimiento	Saldo Disponible
10/2/2022	Marianela Montalvo	225487	Corriente	100	True	600	700
8/2/2022	Marianela Montalvo	496825	Ahorros	540	True	-540	0

Ejemplo Json:

```
{
  "Fecha": "10/2/2022",
  "Cliente": "Marianela Montalvo",
  "Numero Cuenta": "225487",
  "Tipo": "Corriente",
  "Saldo Inicial": 100,
  "Estado": true,
  "Movimiento": 600,
  "Saldo Disponible": 700
}
```

## Instrucciones de despliegue

- Generar el script de base datos, entidades y esquema datos, con el nombre BaseDatos.sql.
- Ejecutar Postman para poder realizar las verificaciones  
(<http://{servidor}:{puerto}/api/{metodo}...{Parámetros}>)

## Entregables

- La Solución debe ser cargado a un repositorio Git público, se debe enviar la ruta de este repositorio.
- Descarga archivo Json, de Aplicación Postman, para validación de los endpoints.
- Se debe entregar antes de la fecha y hora indicada por correo.