ENUNCIADO DE LA PRUEBA DE DESEMPEÑO – MÓDULO 6

Arquitectura Hexagonal + JPA Avanzado + Seguridad JWT + Observabilidad + Microservicios + Docker + Pruebas

---

Caso de uso principal

CoopCredit, una cooperativa de ahorro y crédito con sedes en varias ciudades del país, actualmente gestiona el proceso de solicitud y evaluación de créditos mediante hojas de cálculo, documentos físicos y validaciones manuales. Esto ha generado problemas significativos:

- Inconsistencias en los historiales de crédito.
- Errores en la aprobación de solicitudes.
- Falta de trazabilidad en la evaluación del riesgo.
- Largas demoras en los estudios de crédito.
- Ausencia total de autenticación segura y control de accesos.
- Cero estandarización en manejo de errores y validaciones.
- Dificultad para operar en entornos distribuidos.
- Nula capacidad de pruebas y monitoreo.

La dirección de CoopCredit ha decidido construir un **Sistema Integral de Solicitudes de Crédito**, modular, seguro, escalable y con arquitectura profesional de tipo **Hexagonal**, que permita operar de forma adecuada en entornos reales.

Objetivo de la Prueba

Construir un sistema profesional compuesto por **uno o dos microservicios**, cumpliendo estándares empresariales de arquitectura, seguridad, pruebas y despliegue. El servicio principal será:

1. credit-application-service (Servicio central de Solicitudes de Crédito)

Con arquitectura hexagonal, dominios puros, casos de uso independientes, puertos de entrada y salida, adaptadores REST/JPA, validaciones, seguridad, métricas, pruebas y contenedorización.

2. risk-central-mock-service (Servicio externo simulado de riesgo crediticio)

Microservicio simple que devuelve un **score aleatorio único por documento**, simulando una “central de riesgo”.

El desarrollador deberá integrar **TODAS** las competencias de las semanas 2–6.

Requerimientos funcionales

1) Gestión de Afiliados

- Registrar afiliados con:
    - documento, nombre, salario, fecha de afiliación, estado (*ACTIVO* / *INACTIVO*)
- Editar información básica.
- Validar:
    - documento único,
    - salario > 0,
    - afiliado ACTIVO para solicitar crédito.

---

2) Gestión de Solicitudes de Crédito

Cada solicitud debe incluir:

- afiliado solicitante
- monto solicitado
- plazo (meses)
- tasa propuesta
- fecha de solicitud
- estado (*PENDIENTE*, *APROBADO*, *RECHAZADO*)
- evaluación asociada

Flujo obligatorio:

1. El afiliado registra una solicitud (estado **PENDIENTE**).
2. El sistema invoca al **risk-central-mock-service** vía un **adapter REST**, enviando documento + monto + plazo.
3. Recibe score y nivel de riesgo.
4. Aplica políticas internas:
    - relación cuota/ingreso
    - monto máximo según salario

- antigüedad mínima (p.ej. 6 meses)

5. El caso de uso **Evalúa Solicitud**:
    - genera EvaluaciónRiesgo
    - decide **APROBADO** o **RECHAZADO** con motivo
    - actualiza la solicitud
6. Todo el proceso debe ser **transaccional**.

---

3) Microservicio risk-central-mock-service

Este servicio debe exponer un endpoint:

POST /risk-evaluation

Body:

```
{
  "documento": "string",
  "monto": 5000000,
  "plazo": 36
}
```

Reglas de respuesta:

La respuesta debe variar **según el documento**, pero ser siempre **consistente** para ese documento.
Es decir:

➡️ **Un mismo documento siempre debe devolver el mismo score y nivel de riesgo.**
➡️ **Un documento distinto debe generar un resultado distinto.**

Implementación sugerida:

1. Convertir el documento en un seed numérico (hash mod 1000).
2. Generar un score entre 300 y 950 basado en ese seed.
3. Clasificar:
    - 300–500 → ALTO RIESGO
    - 501–700 → MEDIO RIESGO
    - 701–950 → BAJO RIESGO

Respuesta:

```
{
  "documento": "1017654311",
  "score": 642,
  "nivelRiesgo": "MEDIO",
  "detalle": "Historial crediticio moderado."
```

```
}
```

Este microservicio NO usa JPA ni seguridad. Es liviano.

---

4) Seguridad, Roles y Autenticación

Debe implementarse seguridad completa:

- Autenticación con **JWT** (stateless).
- Encriptación de contraseñas con **PasswordEncoder**.
- Roles:
    - `ROLE_AFILIADO`
    - `ROLE_ANALISTA`
    - `ROLE_ADMIN`
- Endpoints:
    - `/auth/register`
    - `/auth/login`
- Control de acceso:
    - Afiliado → solo sus solicitudes
    - Analista → solicitudes en estado PENDIENTE
    - Admin → acceso completo

---

5) Validaciones, Errores Estándar y Manejo Global

Obligatorio:

- Validación avanzada con Bean Validation:
    - Validaciones cruzadas (cuota/ingreso, plazo válido, afiliado activo)
- Manejo global con `@ControllerAdvice`
- Formato de error **ProblemDetail (RFC 7807)**:
    - type
    - title
    - status
    - detail
    - instance

- **timestamp**
- **traceId**

- Logging estructurado
- Personalización de errores JPA / acceso denegado / validaciones

---

6) Persistencia, Lazy/Eager y Transacciones

Implementar:

- JPA + Hibernate avanzado
- Relaciones:
  - Afiliado 1–N Solicitudes
  - Solicitud 1–1 EvaluaciónRiesgo
- Evitar N+1 con `@EntityGraph`, `join fetch` o `batch-size`
- Evaluar solicitud → proceso completo dentro de un **@Transactional**
- Flyway con:
  - V1__schema
  - V2__relaciones
  - V3__datos_iniciales (opcional)

---

7) Pruebas Unitarias, Integración y Testcontainers

Obligatorio:

Unitarias (JUnit + Mockito)

- Casos de uso puros del dominio
- Mock del RiskCentralPort

Integración (Spring Boot Test + MockMvc)

- CRUD de solicitudes
- Seguridad con JWT

Testcontainers

- Base de datos en contenedor
- Pruebas reproducibles

8) Observabilidad: Actuator + Micrometer

Debe exponerse:

- `/actuator/health`
- `/actuator/info`
- `/actuator/metrics`
- `/actuator/prometheus` (si lo deseas)

Métricas clave:

- tiempo de respuesta
- errores
- solicitudes por endpoint
- fallas de autenticación

---

9) Contenerización y Microservicios

Dockerfile multi-stage

- Etapa build: Maven + JDK
- Etapa run: JRE slim

docker-compose

- credit-application-service
- risk-central-mock-service
- db
- (opcional) gateway o config-server

Criterios de Aceptación

✓ Funcional

- Registro y autenticación con JWT
- Solicitudes creadas + evaluadas correctamente
- Integración completa con risk-central

✓ Arquitectura

- Hexagonal pura, con puertos y adaptadores
- Dominio sin dependencias de frameworks
- MapStruct funcionando

✓ Persistencia

- Relaciones JPA correctas
- Transacciones completas

✓ Seguridad

- JWT válido
- Roles aplicados
- Accesos restringidos

✓ Calidad

- Pruebas unitarias + integración
- Testcontainers funcionando

✓ Observabilidad

- Actuator expone métricas
- Logging estructurado

✓ Despliegue

- Dockerfile funcional
- docker-compose operativo

✓ Documentación

- README completo
- Diagramas: arquitectura, casos de uso, relación entre microservicios

---

Entregables

✓ Repositorio GitHub público
✓ Proyecto comprimido (.zip)

✓ Colección de pruebas (Postman u otra) + SWAGGER
✓ README con:

- Descripción del sistema
- Endpoints
- Instrucciones de ejecución local + docker-compose
- Roles y flujo
- Capturas de pruebas
  ✓ Diagrama de arquitectura hexagonal
  ✓ Evidencia de métricas y logs