SIMULACRO OFICIAL — Prueba de Desempeño (Preparación Módulo 6)

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

---

Caso de uso principal

VetTrack, una red de **clínicas veterinarias** con sedes en varias ciudades del país, administra actualmente la atención médica, historial de mascotas y programación de citas en documentos dispersos, hojas de Excel y decisiones manuales del personal.

Esto genera enormes problemas:

- Pérdida o inconsistencia del historial médico.
- Errores al asignar citas.
- Diagnósticos sin trazabilidad.
- Dificultad para validar disponibilidad real de veterinarios.
- Cero autenticación y control de accesos.
- Ausencia de métricas y monitoreo.
- Operaciones imposibles de escalar en múltiples sedes.
- Ausencia de pruebas y cero estandarización técnica.

La dirección de VetTrack ha decidido construir un **Sistema Integral de Gestión Veterinaria**, modular, seguro y escalable, utilizando **Arquitectura Hexagonal**, microservicios y estándares empresariales.

Objetivo de la Prueba

Construir un sistema profesional compuesto por **uno o dos microservicios**, cumpliendo estándares avanzados de arquitectura, seguridad, pruebas y despliegue.

Los servicios principales:

1. appointment-service (Servicio central de gestión veterinaria)

Implementa arquitectura hexagonal, dominio puro, casos de uso, puertos y adaptadores REST/JPA, validaciones, seguridad, métricas, pruebas y contenedorización.

2. vet-availability-mock-service (Servicio simulado de disponibilidad de veterinarios)

Microservicio liviano que responde si un veterinario está disponible en una fecha/hora.

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

Requerimientos funcionales

1) Gestión de Mascotas

- Registrar mascotas con:
- nombre
- especie (PERRO, GATO, AVE, OTRO)
- raza
- edad
- nombreDueño
- documentoDueño
- estado (ACTIVA / INACTIVA)

Validaciones:

- documentoDueño obligatorio
- edad > 0
- mascota ACTIVA para solicitar citas

---

2) Gestión de Citas Médicas

Caada cita incluye:

- mascota
- veterinario
- fecha
- hora
- motivo
- estado (PENDIENTE, CONFIRMADA, CANCELADA)
- diagnóstico asociado (opcional)

Flujo obligatorio:

1. El dueño solicita una cita (estado **PENDIENTE**).
2. El sistema llama a **vet-availability-mock-service** (vía adapter REST), enviando:
    - idVeterinario
    - fecha
    - hora
3. Recibe una respuesta:

{

"veterinarioId": 22,

"disponible": true,

"motivo": "Horario libre"

}

4. Si no está disponible → la cita se RECHAZA automáticamente.
5. Si está disponible:
    - El caso de uso **ConfirmaCita** asigna estado **CONFIRMADA**.
6. Cuando el veterinario atiende la cita:
    - Registra un **Diagnóstico** con:
        - descripción
        - tratamiento sugerido
        - recomendaciones
7. Todo el proceso debe ser **transaccional**.

---

3) Microservicio vet-availability-mock-service

Este servicio debe exponer un endpoint:

POST /availability

Body:

```
{
  "veterinarioId": 22,
  "fecha": "2025-01-15",
  "hora": "10:30"
}
```

Reglas de respuesta:

Debe ser **consistente para cada combinación veterinario-fecha-hora**.
Es decir:

➡️ **Si preguntas 10 veces por el mismo veterinario, misma fecha y misma hora, la respuesta siempre debe ser igual (disponible o no disponible).**
➡️ **Pero si cambia cualquiera de los tres valores → el resultado puede ser distinto.**

Implementación sugerida:

1. Generar un hash combinando los tres valores.
2. Si el hash % 2 == 0 → disponible.
3. Si el hash % 2 != 0 → no disponible.

Respuesta:

```
{
  "veterinarioId": 22,
  "disponible": false,
  "motivo": "Agenda ocupada"
}
```

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_DUENO`
    - `ROLE_VETERINARIO`
    - `ROLE_ADMIN`
- Endpoints:
    - `/auth/register`
    - `/auth/login`
- Control de acceso:
    - DUENO → solo citas de sus mascotas
    - VETERINARIO → citas asignadas a él
    - ADMIN → acceso total

---

5) Validaciones, Errores Estándar y Manejo Global

Obligatorio:

- Validación avanzada con Bean Validation:
    - Validaciones cruzadas (mascota activa, horario válido (no antes de la hora actual), disponibilidad del veterinario)
- 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:
    - Mascota 1–N Citas
    - Cita 1–1 Diagnóstico
    - Veterinario 1–N Citas
- 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 (veterinarios precargados)

---

7) Pruebas Unitarias, Integración y Testcontainers

Obligatorio:

Unitarias (JUnit + Mockito)

- Casos de uso: solicitar cita, confirmar cita, registrar diagnóstico.
- Mock del puerto VetAvailabilityPort.

Integración (Spring Boot Test + MockMvc)

- CRUD de citas
- 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
- citas confirmadas por hora
- fallas de autenticación

---

9) Contenerización y Microservicios

Dockerfile multi-stage

- Etapa build: Maven + JDK

- Etapa run: JRE slim

docker-compose

- appointment-service
- vet-availability-mock-service
- db (Postgres recomendado)
- (opcional) gateway o config-server

Criterios de Aceptación

✔️ Funcional

- Registro y autenticación con JWT
- Citas solicitadas y confirmadas correctamente
- Integración con vet-availability-mock-service

✔️ 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
- Swagger
- Diagramas: arquitectura, casos de uso, relación entre microservicios
- Evidencias de pruebas(reporte % Code Coverage)

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