Context-map Spring PetClinic

Rol: Arquitecto de Software

Enfoque: Domain-Driven Design (DDD) y Modernización de Monolitos

1. Identificación de Bounded Contexts

Spring PetClinic es un sistema monolítico organizado por funcionalidades principales (Owner, Pet, Visit, Vet). Esta organización permite identificar claramente los Bounded Contexts, es decir, áreas del sistema con responsabilidades definidas y modelos propios.

Contexto	Tipo de Subdominio	Responsabilidad	Nivel de Dependencia
Owner & Pet Management	Core Domain	Gestión de propietarios y mascotas	Medio
Visit Management	Supporting Domain	Registro y consulta de visitas médicas	Alto (depende de Owner)
Veterinarian Management	Supporting Domain	Gestión de veterinarios y especialidades	Bajo
System Infrastructure	Generic Subdomain	Configuración, caché, internacionalización	Transversal
Shared Kernel	Compartido	Clases base del modelo (BaseEntity, Person)	Implícito

1.1 Explicación Estratégica

Owner & Pet es el núcleo del sistema y controla la identidad de las mascotas.

Visit maneja el historial clínico, pero depende estructuralmente de Owner.

Vet es mayormente independiente y principalmente de lectura.

System Infrastructure brinda soporte técnico transversal.

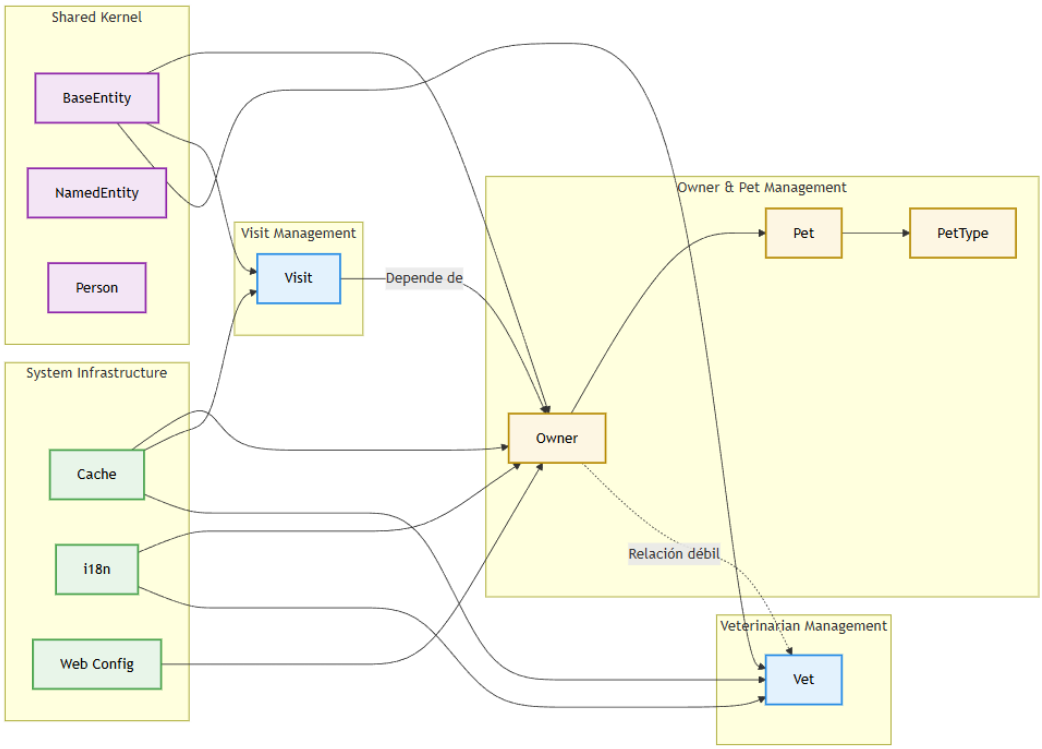
Shared Kernel contiene clases base compartidas entre contextos.

2. Relaciones entre Contextos (Context Map)

Relación	Patrón DDD	Significado
Visit → Owner	Conformist (actual)	Visit usa directamente el modelo de Owner
Visit → Owner (propuesto)	Customer/Supplier + ACL	Visit debe consumir solo contratos (PetId/DTO)
Owner ↔ Vet	Separate Ways	No existe dependencia estructural fuerte
System → Dominios	Customer-Supplier	Infraestructura provee soporte técnico
Todos → Shared Kernel	Shared Kernel	Comparten clases base

Diagrama Conceptual:

Shared Kernel → Owner, Visit, Vet
System Infrastructure → Todos los dominios
Visit → Owner (Dependencia directa actual)
Owner y Vet operan de forma relativamente independiente



3. Observaciones Arquitectónicas Críticas

#	Observación	Qué Significa	Recomendación
01	Owner es el núcleo del sistema	Es el módulo más importante y otros dependen de él	Definir límites claros y proteger su modelo interno
02	Visit depende directamente de Owner	Existe acoplamiento fuerte	Usar solo PetId o DTOs para reducir dependencia
03	Vet es el más independiente	Tiene bajo nivel de dependencia	Extraer primero si se migra
04	Clases compartidas (Shared Kernel)	Código base común entre módulos	Reducir código compartido antes de separar
05	No existe modularización formal	Organizado por carpetas, pero no por módulos reales	Modularizar internamente antes de microservicios

4. Riesgos de Migración

• Mantener Shared Kernel completo puede generar monolito distribuido.

• Separar Visit sin desacoplarlo de Owner puede romper consistencia.

• Problemas transaccionales al dividir bases de datos.

• Migrar sin modularizar primero aumenta el riesgo técnico.

5. Estrategia de Modernización

Fase 1: Modularización interna por contexto (owner, visit, vet, infrastructure).

Fase 2: Definición de APIs internas y contratos claros.

Fase 3: Implementar Anticorruption Layer entre Visit y Owner.

Fase 4: Extracción progresiva comenzando por Vet, luego Visit y finalmente Owner.

6. Conclusión Ejecutiva

Spring PetClinic posee una estructura organizada por dominios que facilita aplicar DDD. El principal punto crítico es la dependencia estructural entre Visit y Owner. La modernización debe enfocarse en formalizar límites, reducir acoplamiento y modularizar antes de avanzar hacia microservicios.