1. DOMAIN LANDSCAPE & STRATEGY

Audit Date: February 10, 2026 **Lead Auditors:** P. Pineda & C. Martinez

Architecture: Modular Monolith **Runtime:** **JAVA 25 (Enabled)**

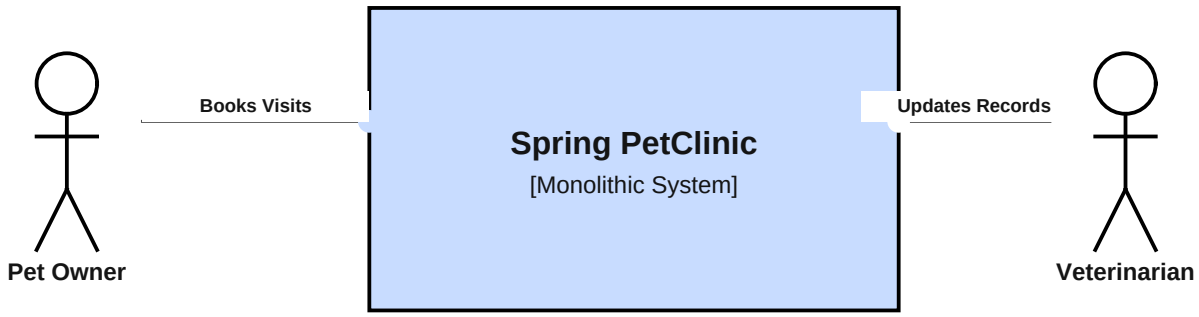
1.1. STRATEGIC DOMAIN CLASSIFICATION

The Spring PetClinic system is not a uniform block of code. Using Domain-Driven Design (DDD) principles, we identified three distinct types of sub-domains operating within the same runtime.

Sub-Domain	Type	Responsibilities & Code Location
Patient Care	CORE	Manages Pets, Medical History (Visits), Owners. (Pkg: .owner)
Clinic Mgmt	SUPPORTING	Manages Vets, Specialties, Shifts. (Pkg: .vet)
Infrastructure	GENERIC	Base Entities, Caching, Error Handling. (Pkg: .model)

System Context Diagram (C4 Level 1)

High-level illustration of how external actors interact with the business 'Core'.

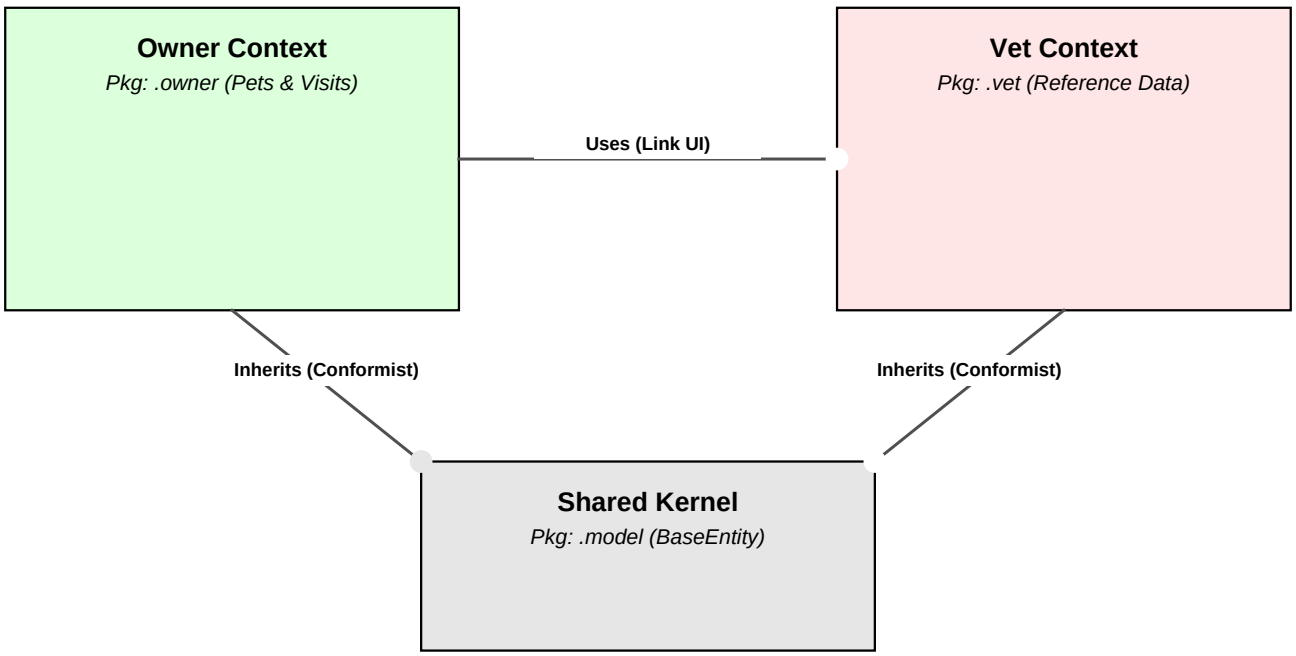


ARCHITECTURAL FINDING

The system acts as a centralized hub. There is no separation between frontend and backend (Server-Side Rendering via Thymeleaf). All business logic resides in a single transactional boundary.

2. TACTICAL CONTEXT MAP (CURRENT STATE)

This diagram visualizes the actual relationships found in the source code (`src/main/java`). The analysis reveals a **'Shared Kernel'** pattern via the `.model` package.



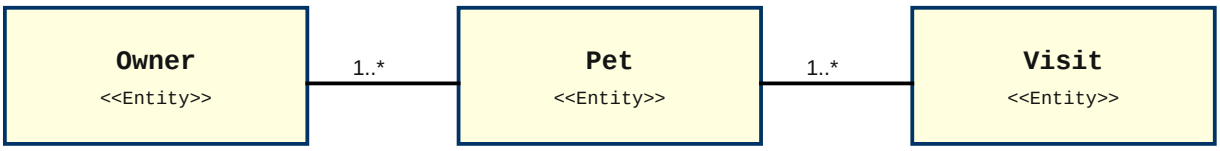
2.1. Coupling Analysis

A **Conformist** relationship exists. The business contexts (Owner, Vet) rigidly depend on the Shared Kernel. Any change in `BaseEntity` forces a recompilation and deployment of the entire system.

Critical Finding: The `Visit` entity is physically located within the `owner` package. This suggests the concept of 'Scheduling' lacks its own space, coupling Appointment logic with Patient Registration.

3. AGGREGATE DESIGN & ENTITIES

Below we detail the internal structure of the 'Core Domain'. The diagram shows how JPA Entities relate in memory today.



3.1. The 'Visit Anomaly'

The current code models the `Visit` as an object owned by the `Pet`. This means to load a Visit, JPA often loads the Pet and the Owner (EAGER fetching or long transaction scope). In an ideal distributed architecture, `Visit` should be an independent Root Aggregate that holds only the `petId`.

> **REFACTORING OPPORTUNITY**
> To enable microservices, we must break the hard link between Pet and Visit. Visits should reference Pets by ID, not by object composition.

4. JAVA 25 MODERNIZATION STRATEGY

The technology stack allows for the use of Java 25 Preview Features. As Staff Engineers, we propose refactoring the current anemic model using the following language capabilities:

4.1. Immutability with Java Records

Currently, DTOs and Entities are mutable (Getters/Setters). Java 25 allows using `records` for immutable data projections, improving safety in concurrent environments.

```
// PROPOSED REFACTORING (JAVA 25)
public record VisitDetails(
    LocalDate date,
    String description,
    @NotNull Integer petId  // Reference by ID, not Object
) {
    public VisitDetails {
        if (date.isBefore(LocalDate.now()))
            throw new IllegalArgumentException("Invalid Date");
    }
}
```

4.2. Domain Control with Sealed Classes

To prevent business logic dispersion, we can restrict the inheritance hierarchy using `sealed interfaces`. This is vital for the Shared Kernel.

```
public sealed interface PetClinicEntity
    permits Owner, Vet, Pet, Visit { ... }
```

This strictly controls which classes can extend the base entity, preventing unauthorized extensions in the future modularized system.

5. DECOUPLING ROADMAP

Based on the Context Map analysis, we define the following execution plan to transition from the Modular Monolith to a scalable distributed system.

Phase	Technical Action	Business Impact
Phase 1	Extract 'visit' package	Decouple Scheduling from Registration.
Phase 2	Break Shared Kernel	Duplicate BaseEntity for team autonomy.
Phase 3	Domain Events	Async communication (Eventual Consistency).
Phase 4	Java 25 Records	Reduce memory footprint and boilerplate.

ARCHITECTURE GOVERNANCE

1. Forbidden cycles between packages (ArchUnit Test).
2. JPA Entities must not leak out of the Service Layer (use Records).
3. New modules must be compiled with --enable-preview (Java 25).

OFFICIAL TECHNICAL APPROVAL

Pablo Pineda
Lead Staff Engineer
Java 25 Architect

Christian Martinez
Lead Staff Engineer
Domain Expert