Ingeniería del Software II

TP 1 - Parte 2

Docentes a cargo:
Victor Valotto, Cielo Godoy

Alumnas:
Inderkumer, Priscila
Murzi, Ana Sol

22 - SEP - 2025

Trabajo Práctico M1 - parte 2 - IS2: Aplicación de la arquitectura al proyecto

1. **Objetivo**

Aplicar los conceptos de arquitectura vistos en clase (estilos, patrones, vistas, aspectos transversales y atributos de calidad) al proyecto iniciado en Ingeniería de Software I.

2. **Fecha de entrega:**

Semana del 15/09/2025 al 22/09/2025

3. **Directivas**

3.1 Atributos de calidad y escenarios

- **Identificar al menos dos atributos de calidad críticos para su sistema (ej. seguridad, disponibilidad, escalabilidad, mantenibilidad).**

  Dos atributos de calidad críticos para nuestro sistema son:
  - Recuperación
  - Confiabilidad
- **Repasar el escenario de calidad por cada atributo, especificando estímulo, respuesta y medida de la respuesta.**

Escenario de calidad	**Atributo de calidad**	**Estímulo**	**Respuesta**	**Medida de la respuesta**
QA 004.a	Recuperación	Corte de energía en Oro Verde	El sistema se recupera automáticamente sin pérdida de información	Recuperación ≤ 2 minutos sin pérdida
QA 004.c	Recuperación	API de geolocalización no responde	Sistema continúa funcionando sin geolocalización y reintenta conexión	Detección de falla ≤ 10 seg, reconexión automática cada 2 minutos
QA 005.a	Confiabilidad	El usuario aprieta el botón "Datos mascota"	Los datos del dueño de la mascota mostrados son siempre correctos	100% consistencia de los datos
QA 005.b	Confiabilidad	Un vecino reporta una mascota encontrada	El cambio de estado se refleja consistentemente en todo el sistema	100% de actualizaciones de estado exitosas sin inconsistencias

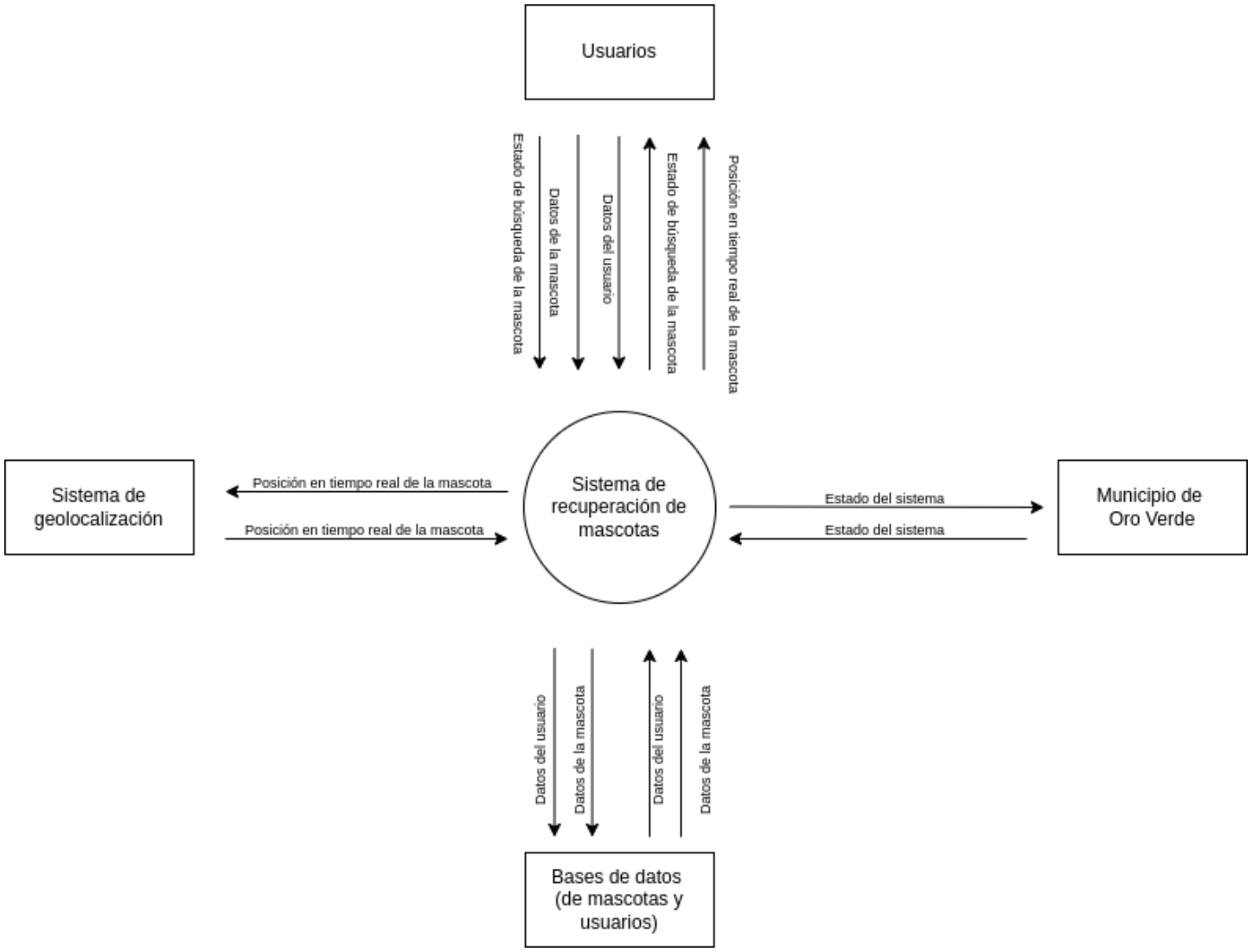
3.2. Vistas de arquitectura

- Documentar la arquitectura de su proyecto usando el modelo de vistas C4.

- Incluir los diagramas correspondientes al modelo

a. Contexto del sistema

Nos basamos en los trabajado en Ingeniería del Software I y utilizamos el diagrama de contexto realizado:

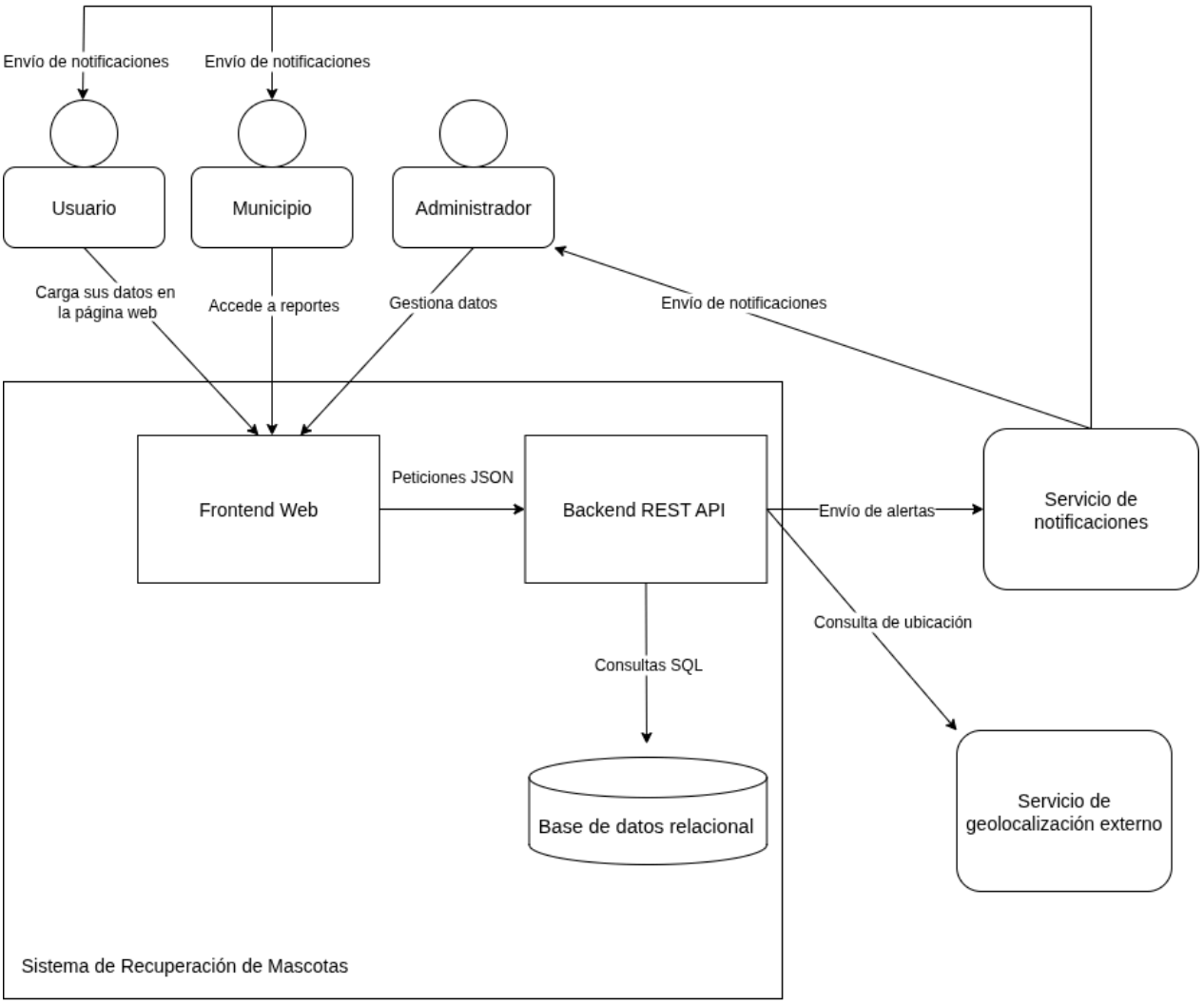


b. Contenedores: [Diagrama contenedores]

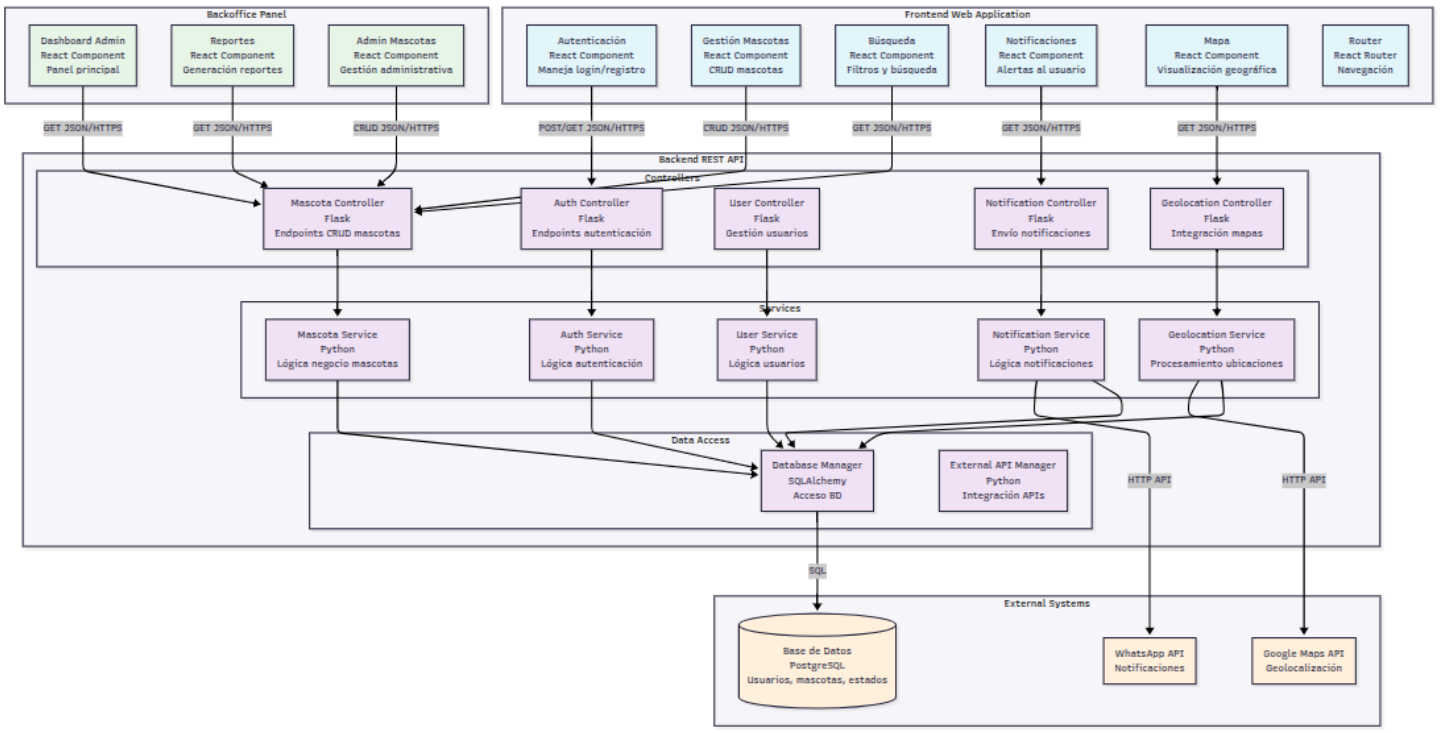
- Frontend Web (React navegador del usuario).
- Backend REST API (Python Flask, maneja lógica de negocio).
- Base de datos relacional (PostgreSQL, guarda usuarios, mascotas, estados).
- Servicio de geolocalización externo (Google Maps API).
- Servicio de notificaciones (WhatsApp).

Relaciones:

Usuarios ↔ Frontend Web ↔ Backend REST API ↔ Base de datos relacional
Backend ↔ API Geolocalización
Backend ↔ Servicio de notificaciones
Municipalidad ↔ Backoffice (panel administrador sobre el mismo backend)



c. Componentes: [Diagrama-componentes]

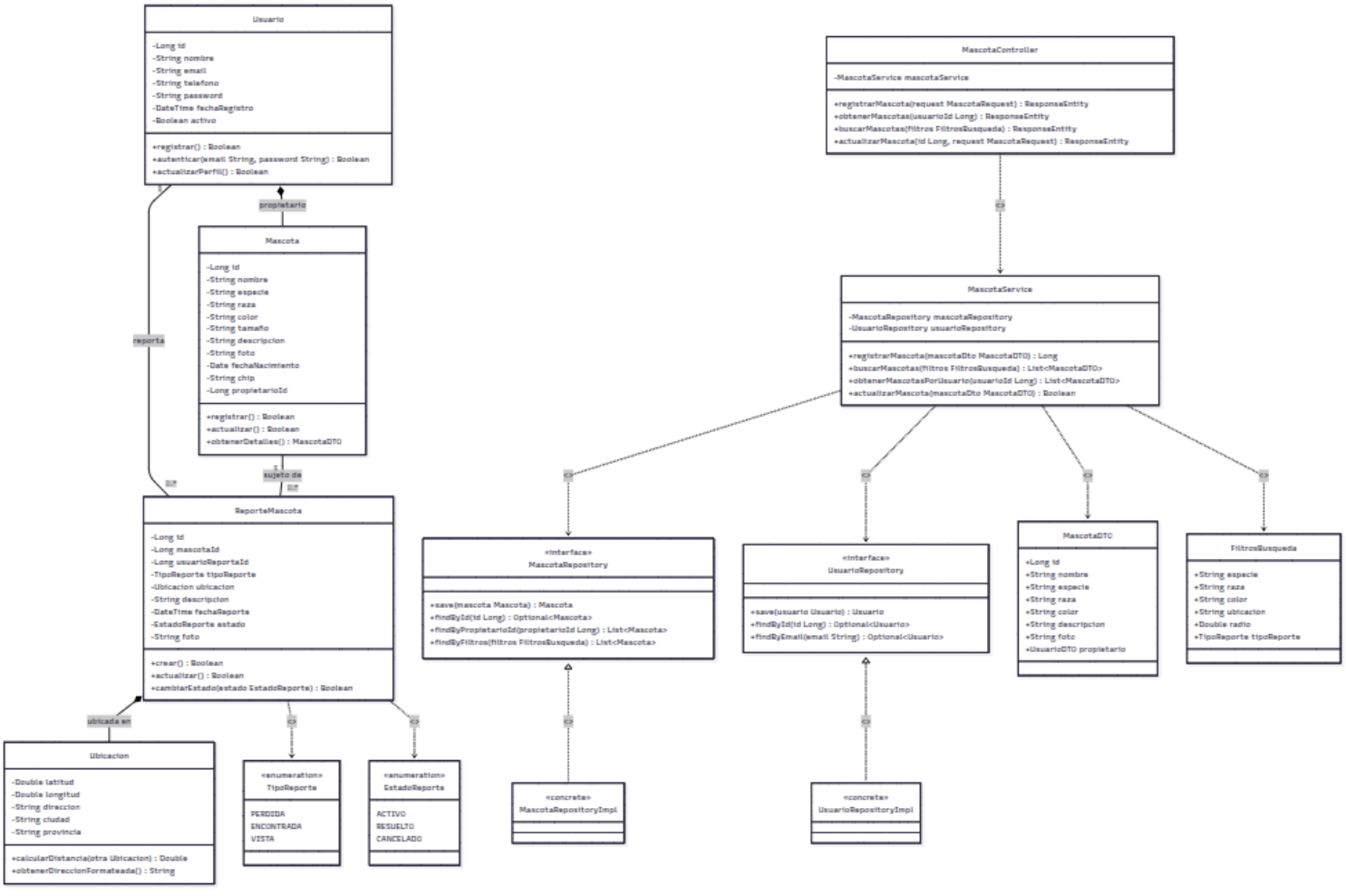


La arquitectura se organiza en tres contenedores principales:

- **Frontend Web Application** contiene los componentes de interfaz de usuario desarrollados en React: Autenticación (manejo de login/registro), Gestión de Mascotas (operaciones CRUD), Búsqueda (filtros y consultas), Notificaciones (alertas al usuario), Mapa (visualización geográfica) y Router (navegación entre páginas).
- **Backend REST API** se estructura en tres capas bien definidas. Los Controllers exponen endpoints REST para autenticación, mascotas, usuarios, notificaciones y geolocalización. Los Services encapsulan la lógica de negocio correspondiente a cada dominio funcional. La capa de Data Access incluye el Database Manager (SQLAlchemy) para persistencia y el External API Manager para integración con servicios externos.
- **Backoffice Panel** proporciona herramientas administrativas mediante componentes React especializados: Dashboard Admin (panel principal), Reportes (generación de informes) y Admin Mascotas (gestión administrativa).

El sistema se integra con una Base de Datos PostgreSQL para persistir usuarios, mascotas y estados. Utiliza Google Maps API para servicios de geolocalización y WhatsApp API para envío de notificaciones automáticas. Los componentes frontend se comunican con los controllers backend mediante peticiones HTTP/JSON. Los controllers delegan la lógica de negocio a los services correspondientes, que a su vez utilizan el Database Manager para operaciones de persistencia. Los services de geolocalización y notificaciones se integran con las APIs externas para completar sus funcionalidades.

d. Código: [Diagrama-codigo]



El diagrama de código representa la estructura estática del componente "Mascota Service" del backend, mostrando las clases, interfaces, atributos, métodos y relaciones UML que implementan la funcionalidad central del sistema.

Las entidades principales del dominio incluyen **Usuario** (con atributos como id, nombre, email, teléfono y métodos para registro y autenticación), **Mascota** (conteniendo información descriptiva como especie, raza, color, chip y métodos para gestión), **ReporteMascota** (que centraliza los reportes de mascotas perdidas/encontradas con ubicación, descripción y estado) y **Ubicacion** (con coordenadas geográficas y métodos para cálculo de distancias).

Los enumerados **TipoReporte** (PERDIDA, ENCONTRADA, VISTA) y **EstadoReporte** (ACTIVO, RESUELTO, CANCELADO) proporcionan vocabulario controlado para clasificar y gestionar el ciclo de vida de los reportes.

Arquitectura por Capas:

- La **capa de presentación** está representada por **MascotaController**, que expone endpoints REST para registro, consulta, búsqueda y actualización de mascotas.
- La **capa de servicio** contiene **MascotaService**, que implementa la lógica de negocio principal, incluyendo validaciones, procesamiento de filtros de búsqueda y orquestación de operaciones complejas.
- La **capa de acceso a datos** define las interfaces **MascotaRepository** y **UsuarioRepository** con métodos para persistencia, consulta por criterios y operaciones CRUD. Estas interfaces son implementadas por clases concretas que manejan la interacción real con PostgreSQL.

MascotaDTO encapsula los datos de mascota para transferencia entre capas, incluyendo información del propietario. **FiltrosBusqueda** estructura los criterios de búsqueda como especie, raza, ubicación y radio geográfico.

El diagrama implementa relaciones UML precisas: **composición** entre Usuario y Mascota (el usuario posee sus mascotas), **asociación** con cardinalidad explícita entre entidades (un usuario puede reportar múltiples mascotas), **dependencia** entre capas arquitectónicas (Controller depende de Service, Service depende de Repository) y **realización** para implementación de interfaces.

La estructura sigue principios de arquitectura limpia con separación clara de responsabilidades, inversión de dependencias mediante interfaces, y encapsulación de la lógica de dominio. Los métodos incluyen parámetros tipados y valores de retorno explícitos, facilitando la comprensión y mantenimiento del código fuente.

3.3. Aspectos transversales

- Identificar qué aspectos transversales deben incluirse en su proyecto (ej. autenticación, gestión de excepciones, interoperabilidad, notificaciones).
- Explicar en qué parte de la arquitectura impactan más y por qué.

Aspectos transversales		**Parte de la arquitectura donde impactan más**	
Autenticación y autorización	• Registro e inicio de sesión de usuarios • Diferenciación de roles • Acceso restringido a funcionalidades críticas	**Backend**	Porque ahí se validan los datos y se van gestionando los permisos por rol
Gestión de excepciones y errores	• Manejo centralizado de errores en backend • Mensajes claros para el usuario final • Registro de errores	**Backend y Base de datos**	Porque es donde se concentran las operaciones críticas (consultas, reportes, integración con APIs externas). Los errores deben manejarse ahí antes de llegar al usuario
Interoperabilidad	• Integración con servicios externos • Formatos estándar de intercambio de datos	**Backend**	Porque el backend actúa como intermediario entre nuestro sistema y los servicios externos (geolocalización, notificaciones). Se deben transformar datos, manejar protocolos estándar y garantizar consistencia
Notificaciones	• Envío automático de alertas • Posibilidad de notificaciones programadas	**Backend, Servicio de notificaciones y Usuario**	Porque el backend decide cuándo enviar notificaciones, pero el servicio externo es el que finalmente las entrega a los usuarios

Seguridad	• Encriptación de contraseñas • Protección de datos personales de los usuarios	**Frontend web y Backend**	Porque la seguridad atraviesa cada capa: desde el acceso al sistema hasta la persistencia de datos
Auditoría y trazabilidad	• Registro de accesos • Modificaciones de datos • Generación de reportes • Registro de errores para el administrador	**Backend y Base de datos**	Porque se registran accesos, cambios de estado de mascotas y generación de reportes. Permite cumplir con las normativas relacionadas con la privacidad de los datos de los usuarios
Escalabilidad y disponibilidad	• Diseño para soportar más usuarios en el futuro • Posible despliegue en la nube • Backups automáticos de la base de datos	**Despliegue de contenedores y Base de Datos**	Porque el sistema debe estar preparado para más usuarios y consultas. Afecta cómo se diseñan los contenedores y cómo se gestiona la persistencia de los datos
Usabilidad y accesibilidad	• Interfaz intuitiva • Acceso inclusivo para vecinos de todas las edades	**Frontend Web**	Porque es el punto de contacto con los usuarios

4. Entregables

Cada grupo deberá entregar un documento en PDF que incluya:

1. Atributos de calidad seleccionados y escenarios de calidad definidos.
2. Diagramas de arquitectura realizados con el modelo de vistas seleccionado.
3. Análisis de los aspectos transversales más relevantes para su sistema.
4. Conclusión grupal: ¿qué decisión arquitectónica fue más difícil de tomar y qué alternativas evaluaron?

Conclusión grupal

La decisión más desafiante fue **definir la estrategia de comunicación entre servicios** para el proceso de matching de mascotas perdidas/encontradas, debido a su impacto directo en nuestros atributos críticos de **recuperación** y **confiabilidad**.

La complejidad surgió al considerar múltiples aspectos transversales simultáneamente:

- **Gestión de excepciones**: Distinguir entre errores temporales y permanentes
- **Interoperabilidad**: Mantener compatibilidad con APIs externas impredecibles
- **Confiabilidad**: Garantizar entrega de notificaciones sin comprometer performance

Esto nos mostró que las decisiones arquitectónicas requieren un análisis detallado, evaluando efectos en cascada sobre todos los atributos de calidad del sistema.

Algunas alternativas que evaluamos fueron:

1. **Comunicación Síncrona (REST)**: Simple pero con riesgo de punto único de falla. Si WhatsApp API falla, se bloquea todo el matching.
2. **Arquitectura por Eventos**: Máxima resilencia y desacoplamiento, pero mayor complejidad y posibles problemas de consistencia eventual.
3. **Patrón Híbrido**: Comunicación síncrona para operaciones críticas + cola asíncrona para servicios externos.

Optamos por el **patrón híbrido** porque:

- Mantiene simplicidad para operaciones core del sistema
- Garantiza que fallos de servicios externos (geolocalización, notificaciones) no afecten la funcionalidad principal
- Cumple con nuestros escenarios de calidad: sistema continúa funcionando sin geolocalización (QA 004.c) y mantiene 100% consistencia de datos (QA 005.b)