**Propuesta de Arquitectura Ideal: Arquitectura Basada en Microservicios + Frontend Web/Móvil (SPA)**

**Resumen de la arquitectura**

**Tipo:** Microservicios (backend desacoplado por dominios funcionales)  
**Frontend:** SPA (Single Page Application) con React/Vue + versión PWA o móvil  
**Backend:** API RESTful o GraphQL (Node.js, Django, Spring Boot, etc.)  
**Base de Datos:** SQL + NoSQL (combinado)  
**Pagos:** Integración con PayPal REST API  
**Infraestructura:** Contenedores Docker + Orquestador (Kubernetes o similar)  
**Despliegue:** Cloud (AWS, Azure, GCP) con CI/CD  
**Seguridad:** JWT, HTTPS, 2FA, cifrado de datos, políticas RBAC  
**Autenticación:** Auth propia + opción futura de OAuth (Google/Facebook)

**1. Modularidad y Escalabilidad**

* Separar la lógica en microservicios te permite **escalar solo los módulos más usados** (por ejemplo, la tienda en fechas de alta demanda).
* Algunos microservicios que puedes tener:
  + servicio-usuarios (login, registro, perfiles)
  + servicio-mascotas (registro y ficha médica)
  + servicio-citas
  + servicio-veterinarios
  + servicio-tienda (catálogo, carrito, pedidos)
  + servicio-pagos (integración con PayPal)
  + servicio-notificaciones

**2. Facilidad de Integración de Pagos con PayPal**

* PayPal tiene SDKs oficiales y REST APIs que funcionan muy bien en arquitecturas con API Gateway.
* Puedes crear un microservicio específico para pagos que gestione órdenes y respuestas desde PayPal, sin mezclar esa lógica con la de productos o usuarios.
* Soporta pagos, reembolsos, y seguimiento desde el mismo módulo.

**3. Alta Disponibilidad y Resistencia a Fallos**

* Con microservicios en contenedores (Docker), puedes reiniciar o escalar servicios individualmente sin tirar todo el sistema.
* Puedes distribuir la carga con un API Gateway (por ejemplo, **Kong**, **Amazon API Gateway** o **NGINX**).

**4. Desarrollo paralelo y mantenimiento**

* Los equipos pueden trabajar en servicios distintos (uno en historial clínico, otro en la tienda) sin estorbarse.
* Más fácil de actualizar o rehacer solo un módulo si se daña o necesita mejora.

**5. Frontend moderno y multiplataforma**

* Un frontend SPA con React o Vue permite una experiencia fluida e inmediata.
* Puedes empaquetarlo como **PWA** para móviles, o crear una app nativa que consuma la misma API.
* Esto te da soporte para **web, Android y iOS** sin reescribir la lógica.

**6. Base de Datos híbrida**

* **SQL (PostgreSQL, MySQL)**: ideal para usuarios, productos, citas, veterinarios, pedidos.
* **NoSQL (MongoDB, Firebase)**: útil para historial clínico (documental), logs, notificaciones.
* Así aprovechas lo mejor de ambos mundos según el tipo de dato.

**7. Seguridad y cumplimiento**

* Puedes implementar roles, permisos, auditorías y 2FA de forma modular y con políticas centralizadas (RBAC).
* Es más fácil aplicar políticas de cifrado y trazabilidad que cumplan con la **Ley de Protección de Datos Personales**.

**8. Escalabilidad futura**

* Si decides integrar nuevas funciones (seguros para mascotas, adopciones, foros, telemedicina), solo agregas nuevos microservicios sin tocar los actuales.

**Tecnologías recomendadas**

| **Componente** | **Tecnología sugerida** |
| --- | --- |
| Frontend (SPA) | React.js + Vite/Tailwind, o Vue.js + Vuetify |
| Backend API | Node.js + Express / NestJS, o Django REST, o Spring Boot |
| BDD relacional | PostgreSQL / MySQL |
| BDD documental | MongoDB |
| Autenticación | JWT + 2FA + RBAC |
| Contenedores | Docker |
| Orquestador | Kubernetes (o Docker Compose en etapa inicial) |
| Infraestructura | AWS / GCP / Azure (Elastic Beanstalk, App Engine) |
| Gateway API | Kong / NGINX / AWS API Gateway |
| Pagos | PayPal REST API |
| Notificaciones | Firebase Cloud Messaging, Email (SendGrid) |

**Conclusión**

La **arquitectura basada en microservicios** es la mejor opción para tu plataforma porque te permite construir una **solución robusta, modular, segura y preparada para crecer**, con integración natural a servicios como PayPal, notificaciones y autenticación segura.