Analisis Proyecto

Nota de contexto: el repo es un monorepo con subcarpetas `laravel12-react`, `chatbot-faq-service` y `pet-detail-service`. Predomina TypeScript (≈71%), seguido de PHP (≈25%) y un poco de Python (≈2%).

1) Modelo mental del sistema actual

Visión general:

Un **monorepo** que aloja (al menos) tres piezas:

- **Frontend React/TypeScript** (`laravel12-react` sugiere coexistencia de front con backend PHP; probable SPA que consume APIs). GitHub+1
- **Backend principal Laravel (PHP)** para core de negocio (autenticación, CRUD de mascotas, refugios, adopciones) y servir el front (o API). GitHub
- **Servicios satélite**:
  - `chatbot-faq-service` (probable microservicio de FAQ/AI; por la huella Python podría ser FastAPI o similar). GitHub+1
  - `pet-detail-service` (probable agregador de detalles/fotos/metadata de mascotas, desacoplado del core). GitHub

Flujo de datos (estimado a partir de la estructura):

1. Usuario navega SPA React → llama endpoints Laravel.
2. Laravel orquesta persistencia (DB relacional) y quizás delega a `pet-detail-service`.
3. Para soporte y asistencias, el front o Laravel consume `chatbot-faq-service`.
4. Despliegue actual apunta a Railway (PaaS) para el entorno público. GitHub

Concentración de responsabilidades:

- **Laravel** concentra dominio de adopción, usuarios, reglas de negocio, y probablemente gateway API.
- **React** concentra experiencia de usuario.

- **Servicios satélite** encapsulan funciones específicas no-críticas (FAQ/AI, detalles de mascota).

2) Evaluación arquitectónica

Fortalezas

- **Monorepo**: facilita sincronía de versiones y DX (developer experience).
- **Stack conocido** (Laravel + React): velocidad de entrega y ecosistema maduro.
- **Primer paso hacia servicios** con `chatbot-faq-service` / `pet-detail-service`.

Riesgos y debilidades

- **Acoplamiento front–back** dentro de `laravel12-react`: nombres y estructura sugieren acoplamiento de build/deploy del front con el back (dificulta escalar equipos y pipelines independientes). GitHub
- **Bordes de dominio difusos**: no hay evidencia clara de límites de contexto (p. ej., “Mascotas”, “Refugios”, “Postulaciones/Match”, “Usuarios/Identidad”, “Contenido/Medios”). Esto tiende a un **orquestador gordo** en Laravel.
- **Servicios satélite sin contrato explícito**: no se ven (desde fuera) contratos tipados/versionados (p. ej., OpenAPI). Riesgo de **fragilidad en integraciones**.
- **Observabilidad incierta**: no se aprecian trazas/metrics/logs estandarizados (OpenTelemetry). Dificulta **MTTR** y **SLOs**.
- **Escalabilidad vertical PaaS**: Railway es ágil, pero sin colas/brokers visibles, el sistema luce **request–response síncrono**, sensible a picos (subidas de tráfico por campañas/adopciones).
- **Gestión de medios** (imágenes de mascotas): sin un servicio dedicado de media, aumenta riesgo de almacenamiento desordenado, costos y **hot paths** lentos si se sirven desde el mismo app.

3) Propuesta de rediseño

Te propongo una **evolución a Monolito Modular + eventos**, con posibilidad de abrir microservicios cuando el volumen lo justifique:

3.1. Monolito Modular (DDD-lite)

En el backend principal (Laravel), reestructurar por **módulos de dominio** (bounded contexts):

- **Identity & Access** (usuarios, roles, OAuth/JWT, privacidad).
- **Shelters** (registro/verificación de refugios, KYC liviano).
- **Pets** (alta/edición, estado: publicada, en revisión, adoptada; fotos).
- **Adoptions/Matching** (postulación, evaluación, scoring de afinidad).
- **Content/FAQ** (preguntas frecuentes, conocimiento; proxy al chatbot).
- **Notifications** (email/push/WhatsApp; plantillas y enrutado).
- **Allies & Marketplace** (aliados comerciales, patrocinios, beneficios).

Cada módulo con **capas** claras: `Domain` (entidades/servicios de dominio), `App` (casos de uso), `Infra` (ORM, HTTP, cola), `API` (Controllers), y **contratos** (DTOs) versionados.

Beneficios: mejor mantenibilidad, tests por módulo, ownership por squad y un camino natural hacia microservicios (strangler).

3.2. Event-Driven + Outbox

Adoptar **eventos de dominio** y una **outbox** transaccional (Postgres + worker) para publicar a una cola (p. ej., **RabbitMQ** o **Kafka** lite—Redpanda). Eventos clave:

- `PetListed`, `PetUpdated`, `PetAdopted`
- `ApplicationSubmitted`, `ApplicationApproved/Rejected`
- `ShelterVerified`
- `MediaUploaded`, `FaqUpdated`

Consumidores:

- `notifications-service` (mail/push)
- `search-indexer` (OpenSearch/Algolia)
- `analytics-service` (embudo, cohortes)

- `chatbot-faq-service` (auto refresh de contexto)

Beneficios: resiliencia a picos, **desacople temporal**, reintentos, **idempotencia**.

3.3. CQRS pragmático

- **Reads:** proyección optimizada (Postgres vistas materializadas o OpenSearch) para listados/filtrados por ubicación, tamaño, edad.
- **Writes:** casos de uso en el módulo `Adoptions`.

  Beneficio: latencias bajas en búsqueda y páginas de catálogo.

3.4. Frontend: Next.js + BFF

Migrar el front a **Next.js (App Router)** con **SSR/ISR** para SEO (crítico para adopciones). Colocar un **BFF** (API Routes/Edge functions) para agregar datos y cachear.

Beneficios: mejor indexación orgánica, performance percibida, **caché** y separación clara del backend de dominio.

3.5. Media Service

Servicio dedicado para **uploads** (S3 compatible tipo **Cloudflare R2**):

- Generación de **thumbnails** y **blurhash** vía workers.
- URLs firmadas, optimización WebP/AVIF, y CDN.
- Moderación básica (NSFW/violencia) con IA.

3.6. Contratos & Seguridad

- **OpenAPI** para todos los endpoints públicos/privados.
- **API Gateway** (Kong/NGINX + JWT) con **rate limiting** y **circuit breakers**.
- **Policy as Code** (OPA/Conftest) en CI.

3.7. Observabilidad

- **OpenTelemetry** (trazas, métricas, logs) + **Tempo/Prometheus/Loki/Grafana** o Datadog.
- **SLOs** por ruta (p99, tasa de error) y **alertas**.

4) Estrategia de transición (cero traumas)

Fase 0 – Hardening actual

- CI/CD en GitHub Actions con **sonarqube/code scanning**, tests** (Pest en Laravel, Vitest/Jest en front).
- **Feature toggles** (Unleash) para activar el nuevo flujo de adopción gradualmente.

Fase 1 – Modularización interna

- Reorganizar Laravel en **módulos de dominio** sin cambiar endpoints (Branch-by-Abstraction).
- Introducir **eventos de dominio** y **outbox** (publicar a una cola local/embebida primero).
- Separar build de **frontend** del backend (pipelines y artefactos independientes).

Fase 2 – BFF + SSR

- Levantar **Next.js** en paralelo sirviendo **/pets (listado)** bajo toggle.
- Medir SEO y Core Web Vitals. Redirigir tráfico progresivo.

Fase 3 – Servicios satélite

- Extraer **notifications-service** y **media-service** detrás de la cola.
- `chatbot-faq-service`: formalizar **OpenAPI** y contratos de contexto.
- `pet-detail-service`: consolidar como **read model**/aggregator o integrarlo como proyección CQRS.

Fase 4 – Search/Analytics

- Indexación en **OpenSearch/Algolia** desde eventos.

- Tableros de **embudo** (views → postulaciones → adopciones), cohortes por canal.

Fase 5 – Endurecer la periferia

- API Gateway, WAF, **rate limits** y **circuit breakers**.
- Chaos drills y **budgets de error** operativos.

5) Ecosistema recomendado

Backend & dominio

- **Laravel 11 LTS** (Octane para alta concurrencia), **Pest** tests, **Laravel Pint**.
- **Doctrine/eloquent-strict** para entidades vírgenes; **spatie/laravel-data** para DTOs.
- **Symfony Messenger** o **Laravel Horizon + Redis** para colas iniciales.

Mensajería & búsqueda

- **RabbitMQ/Redpanda (Kafka)** + **Debezium** si luego quieres CDC.
- **OpenSearch** o **Algolia** para búsqueda facetada.

Datos

- **PostgreSQL** (particiones por estado/fecha), **PostGIS** para geolocalización.
- **Redis** (cache, rate limit, colas).

Frontend

- **Next.js (App Router)**, **TanStack Query**, **Zod** para validación, **tRPC** (opcional) o OpenAPI client.
- **Tailwind** + **Radix UI**.

Media

- **Cloudflare R2** o **S3** + CDN; **imgproxy** o **Cloudflare Images**.

Observabilidad/SecOps

- **OpenTelemetry** SDKs, **Grafana stack** o **Datadog**.

- **Sentry** para errores.
- **Kong**/**KrakenD** como gateway; **OPA** para políticas.
- **OWASP ASVS** como checklist.

Platform

- Empezar con **Railway/Fly.io/Render**; plan de salto a **Kubernetes** administrado (GKE Autopilot o EKS) cuando el tráfico lo exija.
- **GitHub Actions** (lint, tests, SCA, SAST, DAST ligero), **preview envs** por PR.

Copilot/IA

- **GitHub Copilot** con reglas del repo, **OpenAPI** para autocompletar de clientes.
- **Playwright** + **Copilot** para generación de pruebas de UI.
- `chatbot-faq-service`: **FastAPI** + **LangChain/LlamaIndex** con embedding store (**pgvector** o **Qdrant**) y **retrieval-augmented** de FAQs y políticas.

Cierre: Visión "AdoptaFácil 2.0"

- **Núcleo**: **Monolito modular Laravel** con límites claros (Identity, Shelters, Pets, Adoptions, Content, Notifications).
- **Experiencia**: **Next.js SSR/ISR** con BFF para performance y SEO.
- **Ecosistema**: Servicios asíncronos para **media**, **notificaciones**, **search** y **analytics**, conectados por **eventos** y **outbox**.
- **Operación**: Observabilidad completa (OTel), SLOs, gateway y seguridad por capas.
- **Escala**: Lecturas rápidas (CQRS + OpenSearch), **colas** para picos, y un camino limpio a microservicios cuando un módulo requiera independencia de escala.

Con esto tendrás una base **resiliente, mantenible y preparada para crecer**. Si quieres, te dejo un diagrama de alto nivel y una matriz de toggles/fases aplicados a tu backlog actual.

Fuentes observadas: estructura y lenguajes del repo y URL de despliegue en Railway.