ChatGPT

Plan Operativo de Remediación y Despliegue

1 Resumen Ejecutivo

Qué está bien

El proyecto **AdoptaFacil** se construye sobre una arquitectura modular con componentes bien definidos: un backend **Laravel** (usando Inertia.js y Vite) y un frontend **React**, más dos microservicios: uno de correo electrónico en **Spring Boot** y un chatbot en **FastAPI**. Las rutas están organizadas y protegidas mediante middleware de autenticación, lo cual permite separar flujos de usuarios anónimos y autenticados. Los documentos incluyen un **checklist de despliegue** con variables de entorno necesarias para cada servicio y pasos de smoke test. Existen scripts para generación de llaves, migraciones y seeding, así como instrucciones para pruebas básicas y health checks. Esta modularidad y las instrucciones de build facilitan el despliegue en contenedores y la futura escalabilidad.

Qué está mal

Los documentos de **Seguridad** y **Deploy** evidencian múltiples vulnerabilidades y omisiones:

- El **endpoint de reestablecimiento de contraseñas** acepta cualquier token sin expiración y sin validar origen. Esto, junto con URLs expuestas de microservicios y tokens sin caducidad, permite secuestro de cuentas.
- La configuración de **CORS** permite cualquier origen para microservicios y no restringe métodos ni credenciales. Además no se implementan cabeceras de seguridad (CSP, X-Frame-Options, HSTS) ni rate limiting.
- Las **sesiones no están cifradas** y las cookies carecen de atributos `Secure` y `HttpOnly`. El modo debug y credenciales están activados en producción.
- No existe configuración para **Azure**: no hay Dockerfiles ni pipelines; los servicios se ejecutan en modo local y almacenan imágenes en disco local. La cola y el cache se implementan con la base de datos (poco escalable) y no se ejecutan jobs. Tampoco hay configuración de correo SMTP ni de variables necesarias para el mail service.
- No se valida ni sanitiza correctamente el **upload de archivos**, permitiendo subir archivos maliciosos o con extensiones prohibidas. Tampoco hay middleware para restringir rutas de administración.
- Se omiten **logs y métricas**; los endpoints de health check son incompletos y no hay estrategia de rollback ni migraciones automáticas. También existen problemas de rendimiento como **consultas N+1** y ausencia de índices.

Qué bloquea producción (P0)

1. **Inexistencia de despliegue Azure**: No hay Dockerfiles, pipelines ni configuración de App Service o contenedores. Sin ello, el MVP no se puede ejecutar en la nube.
2. **Seguridad crítica**: Tokens de reestablecimiento sin expiración, sesiones sin cifrar, cookies inseguras, CORS abierto, ausencia de rate limiting y cabeceras.
3. **Credenciales expuestas y debug**: Variables de entorno hardcodeadas y debug activado.
4. **Persistencia y colas**: El almacenamiento de imágenes en disco local y el uso de base de datos para colas y cache son inaceptables en producción, especialmente en Azure.

5. **Falta de configuración del mail-service**: Sin SMTP ni Google OAuth los usuarios no pueden restablecer contraseñas.
6. **Validación de archivos y rutas**: Subidas de archivos sin validación y ausencia de middleware para administración.

Riesgos si se despliega sin cambios

Desplegar sin corregir los puntos anteriores podría permitir robo de cuentas mediante el endpoint de reset, ejecución de ataques de cross-site scripting, subida de malware y exfiltración de datos. El CORS abierto y sesiones sin cifrar facilitarían ataques de CSRF y secuestro de sesión. La falta de observabilidad y configuración de colas podría causar pérdida de mensajes o emails no enviados. Además, el debug y credenciales expuestas comprometerían todo el entorno. Finalmente, al no existir despliegue en Azure, no habría persistencia para la base de datos ni escalabilidad, por lo que el sistema se caería ante carga moderada.

2 Decisiones de Alcance (MVP vs. Completo)

Se proponen dos rutas para llegar a producción con el mínimo riesgo. Ambas rutas asumen que se usarán servicios gestionados de Azure (App Service para contenedores, Azure Database for PostgreSQL, Storage Blob y Azure Service Bus) para reducir costos operativos.

Ruta A – MVP estricto

Desplegar solo lo indispensable para permitir el registro y autenticación de usuarios y el flujo principal de adopción. **Se desactivan o mockean los microservicios** (mail service y chatbot) y cualquier funcionalidad no esencial.

Pros:

- Permite llegar a producción más rápido y con menor superficie de ataque.
- Reduce complejidad técnica al eliminar dependencias de microservicios. Se sustituye el mail service por un correo SMTP sencillo (p. ej. SendGrid) y se desactiva el chatbot.
- Facilita validar el producto con usuarios reales, enfocándose en registro, login y adopción.

Contras:

- Algunas funcionalidades (correo personalizado, chatbot de adopción) quedarán deshabilitadas o simuladas, lo que puede afectar la experiencia.
- Requiere re-arquitectar ligeramente para que el backend de Laravel gestione directamente los envíos de email (usando mails nativos de Laravel) mientras se integra el microservicio en el futuro.
- Menor escalabilidad inmediata; el sistema principal tendrá que asumir la cola de emails hasta migrar al microservicio.

Recomendación: Empezar por esta ruta para cumplir el objetivo de MVP, validando seguridad y despliegue. El microservicio de mail y el chatbot se pueden integrar en versiones posteriores.

Ruta B – MVP completo

Desplegar el sistema completo incluyendo el mail service y el chatbot, migrando colas y cache a servicios de Azure, y orquestando todo en contenedores.

Pros:

- Se ofrece al usuario la experiencia completa: emails transaccionales robustos y chatbot de apoyo.
- Permite validar la interacción entre microservicios y ajustar arquitectura distribuida.
- Acelera la transición futura a un producto completo (SaaS), reduciendo deuda técnica posterior.

Contras:

- Aumenta el tiempo de desarrollo inicial y la complejidad de despliegue (tres contenedores, colas externas, etc.).
- Mayores riesgos de seguridad y mayor superficie de ataque al exponer microservicios.
- Dependencia de tokens y OAuth para el mail service; se debe configurar Google OAuth o proveedor de email seguro.

Recomendación: Adoptar esta ruta solo si el equipo puede dedicar tiempo extra a integrar y asegurar los microservicios. De lo contrario, comenzar con Ruta A y planificar la integración progresiva.

3 Backlog Prioritizado

El backlog se divide en prioridades **P0** (bloquea producción), **P1** (estabilidad y observabilidad) y **P2** (mejoras no urgentes). Cada tarea incluye el problema, impacto, rutas afectadas y pasos de implementación.

Tareas P0

ID	Título	Problema / Evidencia	Impacto / Riesgo	Archivos / Rutas afectadas	Pasos de implem
P0-01	**Corregir endpoint de reset de contraseña**	El endpoint permite tokens sin expiración ni validación; tokens expuestos permiten secuestro de cuentas.	Puede permitir que un atacante restablezca cualquier cuenta; compromete integridad.	`routes/web.php`, `AuthController`, `PasswordResetController`, tabla `password_resets`.	1. Añadir colum `expires_at` a 2. Generar toke (por defecto 60 único. 3. Validar y proteger ruta 4. Enviar emails firmado y token

ID	Título	Problema / Evidencia	Impacto / Riesgo	Archivos / Rutas afectadas	Pasos de implem
P0-02	**Configurar CORS y cabeceras de seguridad**	CORS abierto en microservicios y sin cabeceras CSP.	Permite llamadas desde cualquier dominio, exposición a XSS y clickjacking.	`app/Http/Middleware/HandleCors.php`, config `cors.php`, configuraciones de Spring y FastAPI.	1. Definir lista bl dominios (p. ej. producción y loc solo métodos ne `POST`, etc.). 3. cabeceras `Content-Secu` `X-Frame-Optic` `X-Content-Typ` `Strict-Transp` 4. Añadir rate lir tamaño de cuer
P0-03	**Cifrar sesiones y cookies seguras**	Sesiones no cifran datos y las cookies carecen de `Secure` y `HttpOnly`.	Riesgo de secuestro de sesión y robo de credenciales en entornos HTTPS.	Configuración `config/session.php`, `config/cookie.php`; archivos de Spring y FastAPI.	1. Establecer `SESSION_DRIV` usar Azure Cach Habilitar cifrado (`encrypt=tru` cookies con `se` `http_only`, `same_site=st` Establecer expir y renovar token
P0-04	**Eliminar credenciales hardcoded y desactivar debug**	Variables de entorno y credenciales están en código; modo debug activado.	Puede exponer secretos y mostrar información sensible en errores.	`.env.example`, `config/*.php`, `application.properties`, `settings.py`.	1. Extraer todas a variables de en 2. Revisar repos eliminar claves h SMTP, OAuth). 3 `APP_DEBUG=fa` `APP_ENV=prod` Configurar archi propiedades de para leer variab
P0-05	**Migrar colas y cache a servicios de Azure**	Se usa base de datos para colas y cache y no se ejecutan jobs.	Desempeño limitado y riesgo de bloqueo; sin colas no se enviarán emails ni tareas asincrónicas.	`config/queue.php`, `config/cache.php`, jobs y workers; configuración de Spring.	1. Seleccionar Az para colas y Azu Redis. 2. Configu `QUEUE_CONNEC` `CACHE_DRIVER` Crear jobs para 4. Configurar wo contenedores.

ID	Título	Problema / Evidencia	Impacto / Riesgo	Archivos / Rutas afectadas	Pasos de implem
P0-06	**Almacenamiento persistente para imágenes**	Las imágenes se almacenan en disco local.	Al escalar contenedores se pierde la información; no hay redundancia.	Configuración de filesystem `config/filesystems.php`, código de uploads.	1. Configurar dis Azure Blob Stora `filesystems.` Actualizar lógica guardar archivo generar URLs fir imágenes existe
P0-07	**Configurar mail-service o reemplazarlo (Ruta A)**	Sin SMTP ni OAuth el mail service no envía correos; tokens expuestos.	Los usuarios no pueden reestablecer contraseñas; tokens circulan sin caducar.	`mail-service/src/main/resources/application.properties`, rutas de integración en Laravel (`.env` y servicios).	1. Ruta A: desac microservicio de Configurar `MAIL_MAILER=` servicio externo 2. Ruta B: mante pero configurar secret) y rotar to expiración. 3. Ac `MAIL_HOST`, `M` `MAIL_USERNAM` `MAIL_PASSWOR` `MAIL_FROM`.
P0-08	**Agregar middleware de administración y validación de archivos**	No existe middleware para proteger rutas de administrador y la validación de archivos es débil.	Usuarios no autorizados pueden acceder a funcionalidades sensibles; se pueden subir archivos maliciosos.	`routes/web.php`, `app/Http/Middleware`, controladores de carga de archivos.	1. Crear middlew `AdminMiddlew` verifique rol y ap `admin/*`. 2. Añ de tamaño, tipo extensión en up renombrar arch nombres. 3. Des contenido (p. ej. imágenes).
P0-09	**Configurar despliegue en Azure y CI/CD**	No hay Dockerfiles ni pipelines.	El MVP no puede ejecutarse en la nube y no hay reproducibilidad.	Crear `Dockerfile` para Laravel, Spring Boot y FastAPI; archivos `azure-pipelines.yml` o GitHub Actions; scripts de despliegue.	1. Escribir Docke para cada servic dependencias, e 2. Crear `docker-compo` entorno local y `terraform` pa (opcional). 3. Co para construir im desplegar en Az para contenedo Containers) o AK variables y secre

ID	Título	Problema / Evidencia	Impacto / Riesgo	Archivos / Rutas afectadas	Pasos de implen
P0-10	**Implementar health checks y monitoreo**	Health checks actuales son incompletos y falta monitoreo.	No se puede detectar caídas ni degradación; impide rollback controlado.	Servicios `/health` en Laravel, Spring Boot y FastAPI; herramientas de monitorización.	1. Agregar endp que verifiquen D dependencias e Configurar `rea` `liveness prob` Integrar logs es (p. ej. Monolog) (Prometheus/Az Definir alertas y

Tareas P1

ID	Título	Problema / Evidencia	Impacto / Riesgo	Archivos / Rutas afectadas	Pasos de implementación	Criterios de aceptación
P1-01	**Implementar logging estructurado y manejo de errores**	No hay centralización de logs ni manejo consistente de excepciones.	Dificulta detectar fallos y realizar trazabilidad.	`app/Exceptions/Handler.php`, configuración de Monolog, Spring Boot logging y FastAPI middleware.	1. Configurar Monolog para enviar logs a Azure Monitor o Application Insights. 2. Crear un middleware que capture excepciones y devuelva respuestas uniformes. 3. Añadir contexto (usuario, petición) a los logs.	Los errores se registran con nivel correcto y contexto; las respuestas no filtran detalles.

ID	Título	Problema / Evidencia	Impacto / Riesgo	Archivos / Rutas afectadas	Pasos de implementación	Criterios de aceptación
P1-02	**Optimizar consultas y evitar N+1**	Se detectaron consultas N+1 y falta de índices.	Afecta rendimiento y escalabilidad.	Eloquent models, controladores, migraciones de índices.	1. Revisar consultas con `Laravel Telescope` / `debugbar`. 2. Implementar `eager loading` (`with`, `load`) en controladores. 3. Crear índices para claves foráneas y campos usados en búsquedas.	Disminuyen las consultas a DB; los tiempos de respuesta mejoran.
P1-03	**Documentar arquitectura y procesos**	No existe documentación técnica ni diagramas.	Dificulta onboarding y mantenimiento.	Repositorio `docs/` o Wiki.	1. Crear documentación que explique componentes, flujos de datos y dependencias. 2. Incluir diagramas (C4) y tablas de configuración de variables. 3. Describir proceso de despliegue y rollback.	Documentación disponible en el repositorio; actualizada y accesible.
P1-04	**Añadir pruebas integrales y smoke tests**	No hay test automáticos; el checklist indica pruebas manuales.	Incrementa riesgo de regresiones.	Tests `Laravel`, `pytest`, `JUnit` según servicio.	1. Escribir tests de integración para los flujos principales (registro, login, adopción). 2. Crear scripts de smoke tests automatizados tras cada despliegue. 3. Integrar pruebas en pipeline CI/CD.	Tests se ejecutan con éxito; fallan si se rompe un flujo.

Tareas P2

ID	Título	Problema / Evidencia	Impacto / Riesgo	Pasos clave
P2-01	**Implementar autenticación de dos factores (2FA)**	Recomendación de seguridad para incrementar protección.	Reduce el riesgo de acceso no autorizado.	Integrar 2FA por correo o app (Time-based OTP) en Laravel; actualizar flujos y UI.
P2-02	**Refactorizar colas para emails masivos**	Actualmente se utilizará Azure Service Bus; en el futuro se pueden separar colas por tipo.	Mejora escalabilidad y permite priorizar mensajes.	Crear colas separadas (`emails`, `notificaciones`) y workers dedicados.
P2-03	**Integrar chatbot en Ruta B con NLP mejorado**	El microservicio de FastAPI se ofrece como MVP completo.	Ofrece orientación al usuario; aumenta la complejidad de despliegue.	Migrar chatbot a un endpoint seguro con autenticación y limitar llamadas; agregar caching de respuestas.

4 Checklist de Producción

Usar la siguiente lista para confirmar que el entorno está listo para producción. Marcar cada casilla una vez completado.

- [ ] **Seguridad:**
- [ ] CORS restringido a dominios permitidos y métodos necesarios.
- [ ] Cabeceras CSP, X-Frame-Options, HSTS y X-Content-Type-Options configuradas.
- [ ] Tokens de reset con expiración y uso único.
- [ ] Sesiones cifradas y cookies `Secure` / `HttpOnly` / `SameSite`.
- [ ] Rate limiting y protección contra brute force.
- [ ] Validación de archivos subida (tamaño, extensión, MIME) y sanitización.
- [ ] Roles y middleware de administración.
- [ ] **Configuración:**
- [ ] Variables de entorno definidas (`APP_KEY`, `APP_URL`, DB, mail, OAuth, Redis, Service Bus, Blob Storage, etc.).
- [ ] `APP_ENV=production` y `APP_DEBUG=false`.
- [ ] Dockerfiles y pipeline listos y versionados.
- [ ] Configuración del mailer (SMTP o mail service configurado).
- [ ] URLs de microservicios y tokens definidos en variables, no hardcodeados.
- [ ] **Persistencia:**

- [ ] Base de datos configurada en Azure PostgreSQL, con migraciones ejecutadas y seeders aplicados.
- [ ] Colas y cache configurados en Azure Service Bus y Redis.
- [ ] Almacenamiento de imágenes en Azure Blob Storage.
- [ ] **Observabilidad:**
- [ ] Health checks completos para servicios y dependencias.
- [ ] Logs estructurados enviados a Azure Monitor/Application Insights.
- [ ] Métricas básicas (tiempo de respuesta, errores, colas) y alertas configuradas.
- [ ] **Operación:**
- [ ] Script de despliegue automatizado (CI/CD) que construye e instala contenedores.
- [ ] Estrategia de rollback (p. ej. mantener N versiones de la imagen) y migraciones reversibles.
- [ ] Workers de colas activos y escalables.
- [ ] Health checks conectados a liveness/readiness probes en Azure.
- [ ] **Pruebas mínimas:**
- [ ] Smoke test de registro, login, adopción y subida de imágenes.
- [ ] Tests de integración y unitarios ejecutándose en CI.
- [ ] Pruebas de seguridad básicas (OWASP ZAP) y verificación de cabeceras.

5 Plan de Implementación por Pull Requests (PR)

Para agilizar la revisión y asegurar incrementos pequeños, se recomienda dividir el trabajo en varios **PR** secuenciales. Cada PR debe ser revisado y probado antes de fusionarse.

PR1 – Seguridad crítica (P0)

Incluye: – Corrección del endpoint de reset (P0-01), CORS y cabeceras (P0-02), cifrado de sesiones y cookies (P0-03), eliminación de credenciales hardcoded y desactivación de debug (P0-04), middleware de administración y validación de archivos (P0-08).
– Configuración inicial de variables de entorno en `env.example` y removal de secretos.

No incluye: Despliegue Azure ni configuración de colas, storage o mail-service.

Riesgos: Cambios de seguridad pueden romper flujos existentes. Se deben ejecutar pruebas de regresión.

Validación: Ejecutar smoke test local; solicitar reset y verificar expiración de token; revisar cabeceras CORS; revisar cookies. Revisar que no haya secretos en código.

PR2 – Configuración y variables de entorno

Incluye: Migración de colas y cache a Azure (P0-05), almacenamiento de imágenes en Blob (P0-06), configuración de mailer (P0-07) según la ruta elegida, actualización de `.env.example` con variables completas, y externalización de URLs de microservicios.

No incluye: Dockerfiles ni pipeline de Azure; ni health checks avanzados.

Riesgos: La conexión a Azure puede fallar si los permisos no están configurados; se requiere preaprovisionamiento de recursos.

Validación: Ejecutar un trabajo en la cola de prueba; subir imagen y verificar en Blob; enviar correo de reset. Revisar logs de error.

PR3 – Storage persistente y colas (si aplicó ruta B)

Incluye: Ajustes finales de colas y storage en Azure para microservicios; creación de jobs en Laravel; migración de datos existentes.
No incluye: Despliegue ni CI/CD.

Riesgos: Migración de archivos e integridad de datos; pruebas deben asegurar que no se pierdan archivos.

Validación: Comparar número de archivos en Blob vs. local; verificar entrega de jobs.

PR4 – Despliegue Azure / pipeline mínimo

Incluye: Creación de Dockerfiles multistage para cada servicio; `docker-compose.yml` para entorno local; pipeline (GitHub Actions o Azure Pipelines) que construya imágenes y las publique en Azure Container Registry; despliegue en App Service o AKS. Configurar health probes básicos.

No incluye: Monitoreo avanzado ni optimizaciones de performance.

Riesgos: Configuración incorrecta de puertos, variables, o recursos Azure puede impedir el arranque. Costos de Azure deben controlarse.

Validación: Pipeline debe terminar sin errores; los contenedores arrancan y se puede acceder a la aplicación en la URL de Azure. Ejecutar smoke test completo.

PR5 – Observabilidad y pruebas mínimas

Incluye: Health checks completos (P0-10), logging estructurado y manejo de errores (P1-01), pruebas de integración (P1-04) y optimización de consultas (P1-02) inicial. Configurar panel de métricas y alertas.

No incluye: Documentación ni refactors opcionales (P2). 2FA se puede abordar luego.

Riesgos: Las métricas pueden generar overhead; mal configuradas pueden enviar alertas falsas. Pruebas automatizadas requieren tiempo de mantenimiento.

Validación: Health checks devuelven los estados correctos; logs se envían a Azure; tests se ejecutan en CI. Revisar dashboards y alertas.

6 Guía de Ejecución con Copilot (prompts listos)

Los siguientes prompts están diseñados para usarse con **GitHub Copilot** o herramientas de IA de asistencia en desarrollo. Cada uno se focaliza en una tarea P0 o P1. Deben ser copiados y pegados en Copilot para acelerar la implementación.

Prompt para P0-01 – Corregir endpoint de reset de contraseña

Contexto: Proyecto Laravel con tabla `password_resets` y controlador que genera tokens sin expiración. Tokens se envían por email mediante el mailer.

Objetivo: Implementar tokens seguros con expiración y uso único, validar origen y proteger ruta.

Restricciones: No realizar refactors grandes del sistema ni cambiar flujos de negocio; mantener API de autenticación compatible.

Archivos candidatos: `database/migrations`, `app/Models/PasswordReset.php`, `app/Http/Controllers/Auth/ForgotPasswordController.php`, `routes/web.php`.

Implementa: 1. Crear migración que agregue columna `expires_at` a `password_resets` y eliminar tokens viejos. 2. Generar tokens usando `Str::random` y almacenarlos hashados; establecer expiración de 60 min. 3. Modificar controlador para validar que el token no ha expirado ni ha sido usado; invalidarlo tras uso. 4. Proteger la ruta con middleware `signed` y `throttle`.

Criterios de aceptación: Solicitar reset genera correo con enlace firmado; al usarlo dentro del tiempo válido se permite cambiar la contraseña; si se reutiliza o caduca se rechaza.

Tests: Escribir pruebas para generar un token, verificar expiración, probar uso único y asegurar que tokens expirados no permiten reset.

Prompt para P0-02 – Configurar CORS y cabeceras de seguridad

Contexto: El sistema permite cualquier origen en CORS y carece de cabeceras de seguridad. Se usa Laravel, Spring Boot y FastAPI.

Objetivo: Restringir CORS a dominios autorizados, definir métodos y cabeceras permitidos y configurar CSP y otras cabeceras.

Restricciones: No romper funcionalidad de front; mantener endpoints necesarios para microservicios.

Archivos candidatos: `config/cors.php`, `app/Http/Middleware/SecurityHeaders.php` (nuevo), `application.properties` (Spring), `main.py` (FastAPI).

Implementa: 1. Definir lista de dominios permitidos en variable de entorno `CORS_ALLOWED_ORIGINS` y configurarla en `cors.php`. 2. Permitir solo métodos `GET`, `POST`, `PUT`, `DELETE` y credenciales si es necesario. 3. Crear middleware `SecurityHeaders` que añada `Content-Security-Policy`, `X-`

`Frame-Options: DENY`, `X-Content-Type-Options: nosniff`, `Strict-Transport-Security` y `Referrer-Policy`. 4. Configurar en Spring y FastAPI políticas similares usando filtros o middleware.

Criterios de aceptación: Peticiones desde dominios no autorizados son bloqueadas. Las respuestas incluyen cabeceras de seguridad. El front sigue funcionando.

Tests: Usar `curl` desde un dominio no autorizado (Origin header) y comprobar que la respuesta es `403`. Revisar cabeceras de respuesta para asegurar que las políticas están presentes.

Prompt para P0-03 – Cifrar sesiones y cookies seguras

Contexto: Laravel guarda sesiones en base de datos y cookies sin atributos de seguridad; se desea usar Azure Redis.

Objetivo: Configurar sesiones cifradas y cookies con atributos `Secure`, `HttpOnly` y `SameSite=Strict`.

Restricciones: Conservar funcionalidad de login; no cambiar la estructura de la base de datos.

Archivos candidatos: `.env`, `config/session.php`, `config/cache.php`, `config/cookie.php`, `app/Providers/AppServiceProvider.php`.

Implementa: 1. Establecer en `.env` `SESSION_DRIVER=redis`, `REDIS_HOST`, `REDIS_PASSWORD` apuntando a Azure Cache. 2. En `session.php`, habilitar `encrypt` y configurar `connection` a Redis. 3. En `cookie.php`, establecer `secure => env('APP_ENV') === 'production'`, `http_only => true`, `same_site => 'strict'`. 4. Forzar regeneración de sesión al hacer login.

Criterios de aceptación: Las cookies se envían solo por HTTPS y no se pueden leer desde JavaScript. Las sesiones se almacenan en Redis y se cifran.

Tests: Inspeccionar cookies en navegador; intentar leer `document.cookie` (no debe incluir token). Verificar en Azure Redis que se almacenan claves cifradas.

Prompt para P0-04 – Eliminar credenciales hardcoded y desactivar debug

Contexto: Hay claves de API y credenciales en archivos de configuración; `APP_DEBUG` está activado en producción.

Objetivo: Externalizar todas las claves a variables de entorno y asegurarse de que el debug está desactivado en producción.

Restricciones: Mantener compatibilidad con archivos `.env.example`; no exponer secretos en el repositorio.

Archivos candidatos: `.env.example`, `config/*.php`, `application.properties`, `settings.py`.

Implementa: 1. Buscar en el código cualquier valor sensible (claves API, SMTP, OAuth) y reemplazarlo por `env('VAR_NAME')`. 2. Actualizar `.env.example` con claves vacías y comentarios que indiquen

cómo configurarlas. 3. Asegurar que `APP_ENV=production` y `APP_DEBUG=false` cuando se despliega. 4. Crear un script de comprobación que falle el despliegue si detecta debug activo o variables faltantes.

Criterios de aceptación: El repositorio no contiene secretos; la aplicación lee todas las credenciales desde variables de entorno; en producción debug está desactivado.

Tests: Revisar el código con un script que busque patrones de claves; ejecutar la app sin `.env` (debe fallar y pedir variables). Simular un error y comprobar que la respuesta no muestra stack trace.

Prompt para P0-05 – Migrar colas y cache a Azure

Contexto: Actualmente se usa la base de datos para cola y cache; no hay workers definidos; se va a usar Azure Service Bus y Redis.

Objetivo: Configurar colas y cache en servicios gestionados, implementar workers y jobs necesarios.

Restricciones: Mantener estructura de jobs actual; no agregar nuevas funcionalidades fuera de las colas.

Archivos candidatos: `config/queue.php`, `config/cache.php`, `app/Jobs/*`, `docker-compose.yml`.

Implementa: 1. Crear un tema/cola en Azure Service Bus y obtener credenciales (connection string). 2. Configurar `QUEUE_CONNECTION=servicebus` (o driver personalizado) y mapear colas a Service Bus. 3. Configurar `CACHE_DRIVER=redis` y apuntar a Azure Redis; actualizar `.env` con `REDIS_HOST`, `REDIS_PORT`, `REDIS_PASSWORD`. 4. Desarrollar al menos un job (por ejemplo, enviar email) y registrar un worker que consuma la cola. 5. Iniciar workers en el despliegue (por ejemplo, proceso supervisado).

Criterios de aceptación: Mensajes enviados se almacenan en Service Bus y se consumen correctamente; la cache funciona; la base de datos ya no se usa para colas.

Tests: Crear mensaje de prueba; verificar en el portal de Azure que aparece y se elimina tras consumo. Revisar cache leyendo/escribiendo valores.

Prompt para P0-06 – Almacenamiento persistente para imágenes

Contexto: Actualmente las imágenes se guardan en el disco local del contenedor; esto no es persistente en Azure.

Objetivo: Configurar almacenamiento en Azure Blob Storage y modificar las cargas para usarlo.

Restricciones: No modificar la lógica de negocio del manejo de imágenes; mantener el interfaz de usuario.

Archivos candidatos: `config/filesystems.php`, `app/Http/Controllers/ImageController.php`, `docker-compose.yml`.

Implementa: 1. Crear cuenta de almacenamiento en Azure y contenedor Blob para imágenes; obtener `AZURE_STORAGE_CONNECTION_STRING`. 2. En `filesystems.php`, añadir disco `azure` con driver `azure` (puede usar paquete `league/flysystem-azure-blob-storage`). 3. Modificar controlador para guardar imágenes en el disco `azure` y obtener URL mediante `Storage::disk('azure')->url($path)`. 4. Actualizar `.env` con las variables de Azure.

Criterios de aceptación: Las imágenes subidas se almacenan en Azure; las URL generadas son accesibles; en caso de escalar contenedores, las imágenes siguen disponibles.

Tests: Subir una imagen y revisarla en el portal de Azure; intentar escalar contenedor y confirmar que la imagen sigue accesible.

Prompt para P0-07 – Configurar mail-service o reemplazarlo

Contexto: El microservicio de mail no está configurado y requiere OAuth; se puede optar por reemplazarlo temporalmente (Ruta A) o configurarlo (Ruta B).

Objetivo: Asegurar que los usuarios reciben correos de confirmación y reset, sin exponer tokens.

Restricciones: No introducir nuevos microservicios; priorizar la funcionalidad básica de correo.

Archivos candidatos: `.env`, `config/mail.php`, `mail-service/application.properties`, `routes/api.php`.

Implementa: 1. **Ruta A:** Desactivar llamadas al mail service y usar `MAIL_MAILER=smtp` con proveedor como SendGrid; configurar variables `MAIL_HOST`, `MAIL_PORT`, `MAIL_USERNAME`, `MAIL_PASSWORD`, `MAIL_FROM_ADDRESS`. 2. **Ruta B:** Obtener credenciales OAuth (Google) y configurar `GOOGLE_CLIENT_ID`, `GOOGLE_CLIENT_SECRET`, `GOOGLE_REDIRECT_URI` en `mail-service`; implementar refresco de tokens y expiración. 3. Actualizar controladores para usar el método de envío correspondiente.

Criterios de aceptación: Los correos se envían correctamente y se reciben; no hay tokens expuestos en el código; los tokens de reestablecimiento incluyen expiración.

Tests: Solicitar reset; verificar que se envía email. Probar ambos caminos (SMTP y OAuth). Revisar que no se filtre información en logs.

Prompt para P0-08 – Agregar middleware de administración y validación de archivos

Contexto: Actualmente cualquier usuario puede intentar acceder a rutas administrativas y se permite subir archivos sin validación adecuada.

Objetivo: Proteger rutas de administrador mediante middleware y reforzar la validación de archivos en los controladores.

Restricciones: No modificar la estructura de usuarios; los roles deben existir o agregarse con migración. No cambiar la interfaz de carga de archivos.

Archivos candidatos: `app/Http/Middleware/AdminMiddleware.php` (nuevo), `app/Providers/RouteServiceProvider.php`, `app/Http/Controllers/FileUploadController.php`.

Implementa: 1. Crear middleware `AdminMiddleware` que compruebe el campo `is_admin` del usuario autenticado; en caso contrario, redirigir a `/` o devolver `403`. 2. Registrar middleware en `Kernel.php` y aplicarlo en las rutas de administrador. 3. En `FileUploadController`, validar tamaño máximo (p. ej. 5 MB), tipo MIME permitido (`image/jpeg`, `image/png`), y usar `Storage::putFileAs` para renombrar archivos con `uniqid()` y sanitizar el nombre original. 4. Usar librería de sanitización (p. ej. `Intervention/Image`) para limpiar metadatos o contenido malicioso.

Criterios de aceptación: Un usuario sin rol admin no puede acceder a `/admin/*` y los uploads inválidos (extensión o tamaño) son rechazados con mensaje claro.

Tests: Crear usuario sin rol admin e intentar acceder a ruta `/admin/dashboard` (debe fallar). Subir archivo `.php` o mayor a 5 MB (debe fallar) y subir imagen válida (debe subir). Verificar que el archivo tiene nombre único y está desinfectado.

Prompt para P0-09 – Configurar despliegue en Azure y CI/CD

Contexto: El proyecto no tiene Dockerfiles ni pipelines; se debe desplegar en Azure con bajo costo.

Objetivo: Crear contenedores reproducibles y un pipeline de CI/CD que construya e implemente la aplicación en Azure (Web App for Containers o AKS).

Restricciones: Utilizar servicios gestionados de bajo costo (App Service y Azure Database for PostgreSQL). Minimizar cambios en el código.

Archivos candidatos: `Dockerfile` (nuevo), `docker-compose.yml`, `.github/workflows/deploy.yml` o `azure-pipelines.yml`, `README.md`.

Implementa: 1. Crear Dockerfile multistage para Laravel: etapa de build que instala dependencias (`composer install`, `npm install`, `npm run build`), y etapa final con PHP-FPM y Nginx. Incluir `entrypoint.sh` que ejecute migraciones. Para microservicios, crear Dockerfiles basados en JDK (Spring Boot) y Python (FastAPI). 2. Crear `docker-compose.yml` para desarrollo local con servicios de DB, Redis, Service Bus (emulator) y Blob Storage (azurite) para simular producción. 3. Configurar pipeline en GitHub Actions: pasos para loguearse en Azure (`azure/login`), construir imágenes y subirlas a `Azure Container Registry (ACR)`, crear/actualizar App Service con la imagen. Definir variables de entorno secretas en GitHub. 4. Configurar `azure-webapp-deploy` o `kubectl` (si usa AKS) para publicar la aplicación; activar ranuras de despliegue para implementar blue-green/rollback.

Criterios de aceptación: Al crear un commit en `main`, se ejecuta la pipeline y se despliega la versión actual en Azure. La aplicación está accesible en la URL pública y se conecta a la DB y servicios.

Tests: Forzar un commit que provoque despliegue; observar la pipeline; acceder a la URL; realizar smoke test. Hacer rollback usando ranuras si se detecta error.

Prompt para P0-10 – Implementar health checks y monitoreo

Contexto: No existen endpoints de health check completos ni monitorización; sin esto, Azure no puede auto-recuperar instancias y no hay visibilidad del estado.

Objetivo: Añadir endpoints `/health` que verifiquen dependencias, configurar probes y enviar logs/metrics a Azure Monitor.

Restricciones: Mantener simples los checks; no agregar dependencia de librerías pesadas si no es necesario.

Archivos candidatos: `routes/api.php`, `app/Http/Controllers/HealthController.php`, `application.properties` (Spring Boot), `main.py` (FastAPI), `docker-compose.yml`.

Implementa: 1. Crear controlador `HealthController` que verifique conexión a la base de datos, Redis, Service Bus y almacenamiento; devolver JSON con estado y tiempos de respuesta. 2. En Spring Boot, habilitar `actuator/health` con `management.endpoints.web.exposure.include=health` y agregar verificadores de DB y Redis. 3. En FastAPI, añadir endpoint `/health` que retorne estado de la app y dependencias; usar try/catch. 4. Configurar en Docker/Kubernetes probes de liveness y readiness que llamen a estos endpoints. 5. Configurar Monolog/Azure Monitor y exportar métricas de latencia y conteo de errores. Configurar alertas en Azure.

Criterios de aceptación: Health check devuelve `UP` cuando todos los servicios están disponibles y `DOWN` si alguno falla; la aplicación se reinicia automáticamente si falla. Logs y métricas se visualizan en Azure.

Tests: Simular caída de la base de datos y verificar que health check marca `DOWN`; revisar que se genera alerta; observar reinicio automático (según configuración de probes). Revisar que métricas están en Azure Monitor.

Prompt para P1-01 – Implementar logging estructurado y manejo de errores

Contexto: No existen logs centralizados ni manejo uniforme de excepciones.

Objetivo: Configurar logs estructurados y manejo de excepciones para detectar fallos y auditar operaciones.

Restricciones: No introducir dependencias pesadas; seguir buenas prácticas de cada framework.

Archivos candidatos: `app/Exceptions/Handler.php`, `config/logging.php`, `application.properties`, `main.py`.

Implementa: 1. Configurar en `config/logging.php` canal `stack` que incluya `single` y `azure` (con `monolog/monolog` y `Microsoft Application Insights`). 2. En `Handler.php`, capturar excepciones y devolver mensajes genéricos al usuario; registrar detalles internos en la herramienta de logging. 3. En Spring Boot, configurar Logback con appender a Azure Monitor y definir niveles. En FastAPI, crear middleware de logging y captura de errores. 4. Agregar contexto (user ID, IP, endpoint) en cada log.

Criterios de aceptación: Los logs se envían a Azure y contienen estructura JSON con contexto; las excepciones no muestran detalles sensibles en la respuesta.

Tests: Provocar error; verificar en Azure Monitor que se registra con contexto; revisar respuesta HTTP (debe ser genérica). Probar diferentes niveles de log (info, warning, error).

Prompt para P1-02 – Optimizar consultas y evitar N+1

Contexto: Se detectaron consultas N+1 y ausencia de índices en tablas clave.

Objetivo: Mejorar rendimiento evitando N+1 y creando índices en campos usados en filtros y joins.

Restricciones: Mantener la lógica de negocio; no reescribir toda la capa de modelos.

Archivos candidatos: Modelos de Eloquent, controladores, migraciones.

Implementa: 1. Usar `with()` para cargar relaciones necesarias en las consultas; evitar ejecutar queries dentro de bucles. 2. Revisar queries lentas mediante el profiler (`Laravel Telescope` o `debugbar`); identificar campos que requieren índices. 3. Crear migraciones que añadan índices a claves foráneas y columnas filtradas (por ejemplo, `user_id`, `adoption_status`). 4. Añadir caché de consultas donde aplique (por ejemplo, catálogos estáticos) usando Redis.

Criterios de aceptación: Reducción del número de queries; mejora en tiempos de respuesta. Las pruebas de carga muestran reducción significativa de latencia.

Tests: Ejecutar endpoint que listaba adopciones; medir queries antes y después; confirmar que se han reducido. Usar `EXPLAIN` para verificar uso de índices.

Prompt para P1-03 – Documentar arquitectura y procesos

Contexto: No existe documentación de arquitectura ni procesos de despliegue.

Objetivo: Crear documentación clara y accesible sobre la arquitectura y operaciones del proyecto.

Restricciones: Mantener la información actualizada; no incluir datos sensibles en diagramas.

Archivos candidatos: `docs/architecture.md` (nuevo), diagramas en `docs/img/`, `README.md`.

Implementa: 1. Redactar un documento `architecture.md` describiendo la arquitectura general, flujos de usuarios, microservicios, tecnologías y dependencias externas. 2. Generar diagramas C4 (Context, Container, Component) usando herramientas como draw.io; almacenar imágenes en `docs/img/`. 3. Describir el flujo de CI/CD, las variables de entorno y el proceso de rollback.
4. Actualizar `README.md` con instrucciones resumidas para correr el proyecto localmente y desplegar en Azure.

Criterios de aceptación: La documentación está disponible en el repositorio; los miembros nuevos pueden comprender la arquitectura y desplegar el proyecto siguiendo la guía.

Tests: Pedir a un colega que siga la guía para levantar el proyecto y dar feedback; actualizar según sea necesario.

Prompt para P1-04 – Añadir pruebas integrales y smoke tests

Contexto: Las pruebas actuales son manuales según el checklist; no hay tests automáticos.

Objetivo: Implementar pruebas de integración y smoke tests para detectar regresiones temprano y validar cada despliegue.

Restricciones: No invertir tiempo excesivo en refactors; las pruebas deben cubrir flujos críticos.

Archivos candidatos: `tests/Feature/*`, `tests/Integration/*`, `pytest` para FastAPI, `JUnit` para Spring Boot.

Implementa: 1. Escribir pruebas de integración en Laravel usando PHPUnit para flujos: registro de usuario, login, solicitud de adopción, subida de imagen. 2. En FastAPI, usar `pytest` para probar endpoints de chatbot y health check. En Spring Boot, usar `JUnit` / `MockMvc` para probar endpoints de correo. 3. Crear script de smoke tests que ejecute estas pruebas básicas tras cada despliegue; integrarlo en la pipeline.

Criterios de aceptación: Las pruebas se ejecutan automáticamente en CI y deben pasar en cada PR. Cualquier falla impide el merge.

Tests: Ejecutar las pruebas localmente y en la pipeline; introducir un bug deliberadamente para verificar que fallan.

7 Preguntas mínimas

1. **Servicio de correo:** ¿Desea utilizar un proveedor de correo externo simple (SendGrid/SMTP) para el MVP (Ruta A) o configurar OAuth y el microservicio completo (Ruta B)?
2. **Opción de despliegue:** ¿Prefieren desplegar en **Azure Web App for Containers** (más sencillo y económico) o en **AKS** (mayor control pero más coste)?
3. **Persistencia de colas:** ¿Aceptan utilizar **Azure Service Bus** para colas y **Azure Cache for Redis** para sesiones y cache, o prefieren otra alternativa?

En ausencia de respuesta a estas preguntas, se asumirá la configuración más sencilla y de bajo costo (SendGrid, Web App for Containers, Service Bus/Redis).
