ChatGPT

Plan integral para la implementación de PriceWaze

1. Departamento de Desarrollo (Dev Dept)

Arquitectura general: PriceWaze adopta una arquitectura **full-stack** moderna con un frontend web (React/TypeScript) y un backend cloud. El frontend (ej. Next.js) ofrece la interfaz inmobiliaria, mientras que el backend integra una solución de **IA multi-agente** para análisis inteligente. En particular, PriceWaze utiliza **CrewAI**, un sistema de agentes especializados que colaboran entre sí [1]. Cada agente tiene un rol definido en el dominio inmobiliario: por ejemplo, un *Market Analyst* analiza la zona y mercado, un *Pricing Analyst* estima el valor justo de la propiedad, un *Negotiation Advisor* sugiere estrategias de oferta, un *Legal Advisor* ayuda con contratos, y un *Coordinator* orquesta las interacciones [1]. Esta separación en múltiples agentes permite un diseño modular y escalable, donde cada componente de IA se encarga de una tarea específica dentro del *workflow* [2] [3]. El backend expone estos análisis a través de APIs REST (por ejemplo, endpoints para análisis de precios, asesoría de negociación, generación de contratos, etc. [4] [5]), construidos con un framework como **FastAPI** en Python.

Integración con la aplicación: El frontend y backend se comunican mediante estas APIs. Al visualizar una propiedad, el cliente web puede invocar al backend para obtener el análisis de pricing o sugerencias. Por ejemplo, existe un evento disparador en la app cuando un usuario abre la página de una propiedad (`onPropertyViewed`), que activa el cálculo de alertas de PriceWaze y muestra una tarjeta de copiloto AI con la información relevante [6]:

```typescript
// Evento al ver una propiedad
onPropertyViewed(user_id, property_id) {
  evaluateAlerts(user_id, property_id);    // calcular alertas correspondientes
  pushCopilotCard(user_id, property_id);   // mostrar tarjeta del Copiloto AI en la UI
}
```
[6]

Este flujo asegura que *"el sistema habla primero"*, es decir, la IA provee de inmediato información y alertas al usuario sin que este tenga que pedirlas explícitamente [7]. Además de eventos en la app, se implementan **jobs en segundo plano** para recalcular insights de precios cada noche y en momentos clave (cambios de precio, nuevas comparables, ofertas/contraofertas) [8]. Esto garantiza que los datos de análisis estén siempre actualizados sin sobrecargar al usuario con cálculos lentos.

Base de código y herramientas: El proyecto utiliza TypeScript en el frontend y Python en el backend, manteniendo buenas prácticas de organización. Por ejemplo, en el backend cada agente AI está definido en módulos separados (p. ej. `agents/market_analyst.py`, `agents/pricing_analyst.py`, etc.), con orquestadores en una carpeta `crews/` que coordinan a múltiples agentes para casos de uso completos

(análisis completo, negociación, contratos) [2]. Esta arquitectura modular facilita el desarrollo concurrente por diferentes desarrolladores (cada uno puede enfocarse en un agente o componente). Se emplea **Supabase (PostgreSQL)** como base de datos principal, aprovechando sus *features* integradas: autenticación, almacenamiento y *RPC functions*. La conexión se configura mediante variables de entorno seguras (por ejemplo `SUPABASE_URL` y keys de servicio) según las instrucciones del README [9], lo que mantiene las credenciales fuera del código fuente.

Calidad y rendimiento: Desde el inicio se establecen prácticas de calidad. Se escriben **pruebas unitarias y de integración** para las funciones críticas (el repositorio incluye un suite de PyTest y scripts de test automatizados [10]), de modo que cada sprint se pueda verificar la estabilidad de las nuevas características. En cuanto a rendimiento, se optimizan las consultas de datos con índices en la BD y *caching* donde aplique. Por ejemplo, se crean índices geoespaciales y por columnas en las tablas de propiedades, ofertas, visitas, etc., para acelerar búsquedas por ubicación, precio, estado, etc. [11]. Asimismo, se utilizan **triggers SQL** para tareas automáticas en la BD, evitando lógica compleja en la aplicación: por ejemplo, un trigger que registra el historial de precios cada vez que cambia el precio de una propiedad inserta automáticamente un registro en la tabla de histórico [12]. Este enfoque de *event sourcing* mantiene consistencia de datos con mínima intervención de la lógica de negocio.

Uso de IA eficiente: Un principio clave en el desarrollo de PriceWaze es usar la IA de forma óptima y económica. **Los modelos de lenguaje (LLM) solo se utilizan para generar explicaciones en lenguaje natural, no para cálculos numéricos】** [13]. Cálculos como porcentajes de sobreprecio/bajo precio, puntajes de equidad, detección de alertas, etc., se realizan con datos estructurados y fórmulas determinísticas, asegurando exactitud y rapidez. La IA (ej. GPT o DeepSeek) se reserva para tomar esos datos y producir narrativas comprensibles para el usuario (por ejemplo: *"Esta propiedad está 11% debajo del mercado, lo que sugiere una oportunidad"*). Este patrón – datos calculados + IA para texto – minimiza costos de API y latencia, alineándose con mejores prácticas [13]. Además, se implementa logging de las invocaciones a IA en la base de datos (tabla `pw_ai_logs` para registrar input, output, latencia, etc. [14]) con fines de *debug* y auditoría, lo que ayuda a monitorear el desempeño de los agentes en producción.

En resumen, el departamento de desarrollo sienta bases sólidas: arquitectura modular con agentes, integración fluida frontend-backend, base de datos optimizada y lógica inteligente distribuida entre aplicación, base de datos e IA. Todo esto soportado por pruebas, automatizaciones y un enfoque pragmático (ej. no sobrecargar la app con paneles complejos ni prompts innecesariamente largos [15]). El resultado es una plataforma preparada para evolucionar, manteniendo alta calidad técnica y adaptabilidad.

2. Marketing

Segmentación y propuesta de valor: PriceWaze se dirige principalmente a **compradores de bienes raíces** (y secundariamente a vendedores y agentes) que buscan tomar decisiones informadas. La propuesta de valor clave es ofrecer un **"copiloto" inteligente** en el proceso inmobiliario, es decir, no es solo otro portal de propiedades, **"no es una app inmobiliaria; es criterio embotellado"** [16]. Esto significa que encapsula experiencia y análisis experto en una herramienta digital. Desde el punto de vista de marketing, se debe comunicar que con PriceWaze un comprador tendrá *insights* que normalmente obtendría de un asesor experimentado, de forma inmediata y personalizada. Por ejemplo, alertas automáticas de sobreprecio o oportunidad, explicación clara de la historia de precio de la propiedad, y recomendaciones de oferta óptima.

Estudios de mercado y tendencias: El equipo de marketing debe apoyarse en datos de mercado para resaltar la relevancia de PriceWaze. Actualmente, la adopción de IA en bienes raíces está creciendo exponencialmente. El mercado global de **PropTech con IA** se valoró en unos $222 mil millones en 2024 y se proyecta que alcance ~$975 mil millones para 2029 (CAGR ~34% anual) [17]. Esto indica un enorme apetito por soluciones tecnológicas en el sector inmobiliario. Incluso los consumidores ya muestran interés: un estudio reciente encontró que **82% de los americanos usan herramientas de IA (como ChatGPT) para obtener información del mercado de vivienda** [18], lo que demuestra que el público está listo para asistentes inteligentes. Sin embargo, también reconocen la importancia del toque humano (62% sigue confiando principalmente en su agente) [19]. Por ello, la estrategia de marketing debe posicionar a PriceWaze como *complemento* que potencia al agente y al comprador, no como reemplazo. Se puede enfatizar que la plataforma brinda transparencia y datos para tomar mejores decisiones, alineado con la expectativa actual de los consumidores de tener **información en tiempo real y transparente** en cada decisión importante [20].

Análisis competitivo: En mercados desarrollados existen ya ejemplos de IA aplicada: por ejemplo, Zillow integró su buscador con ChatGPT para permitir consultas conversacionales y facilitar el paso **"de la búsqueda a la compra"** [21]. Grandes brokerages de bienes raíces también lanzan asistentes IA internos; Douglas Elliman anunció su "Elli AI" para ayudar a agentes con análisis de mercado y estrategias de pricing [22]. Esto valida la dirección de PriceWaze – la industria cree que la IA **"definirá el futuro del sector"** en palabras del CEO de Elliman [23]. Sin embargo, en el mercado local (República Dominicana y Caribe) no hay todavía una herramienta comparable enfocada en comprador. PriceWaze tendría así **diferenciación de primer jugador** (*first-mover advantage*) en ofrecer un copiloto inmobiliario inteligente adaptado al contexto local. Además, la economía dominicana presenta condiciones favorables: es la de más rápido crecimiento en LATAM en las últimas dos décadas [24], con alto nivel de adopción móvil y digital (94% de hogares con móvil, amplia cobertura 4G/5G) [25] [26]. Sectores como el PropTech están *"ganando terreno"* en el país rápidamente [27]. El mercado inmobiliario local se encuentra en fase de maduración con proyecciones de alza de precios residenciales entre 7-12% en 2025 [28], lo que aumenta la necesidad de herramientas de análisis para compradores y hace a PriceWaze muy oportuno.

Estrategia de branding y mensaje: El branding de PriceWaze debe inspirar **confianza** y **modernidad**. Nombre y logo ya comunican "navegador de precios" (Waze -> mapas, Price -> precio), lo cual es intuitivo. En marketing se reforzará esa idea de guíar al usuario por el camino del precio justo. Sugerimos un eslogan claro, p. ej.: "*Tu copiloto inteligente para comprar vivienda con criterio*". Los mensajes clave a comunicar en campañas: - **Transparencia:** PriceWaze revela si un precio está por encima o debajo del mercado con datos claros (ej. *"Esta propiedad está 15% sobrevalorada, quizás podrías ofrecer menos"*). - **Empoderamiento:** El usuario toma decisiones informadas respaldadas por datos y análisis ("*Toma ventaja de oportunidades ocultas que solo un experto detectaría*"). - **Facilidad:** Todo integrado en una experiencia amigable: en lugar de estudiar informes complejos, el usuario recibe *insights* en lenguaje sencillo y visual (3 bullet points, semáforos de oferta, etc.).

Apalancando *consumer psychology*, PriceWaze debe mostrar beneficios en términos que resuenen emocionalmente. Por ejemplo, notificar una *"oportunidad silenciosa"* (propiedad subvalorada) genera **FOMO** (miedo a perder la ganga) y urgencia de actuar [29], mientras alertar de *"sobreprecio emocional"* ayuda al comprador a no dejarse llevar por sesgos y negociar mejor (apela al deseo de seguridad de estar haciendo una compra inteligente [30]). Los estudios de psicología del comprador indican que los compradores buscan **reafirmación de que toman la decisión correcta** [30] y perciben valor según puntos de referencia simples

[31]. Por eso, PriceWaze proporciona puntos de referencia claros (ej. anclajes de precio justo [32]) y explicaciones en términos llanos, para brindar esa tranquilidad.

Canales y tácticas: Se usará una estrategia de marketing digital multicanal. Dado el alto uso de redes sociales para contenidos inmobiliarios (90% de consumidores usan alguna red para este fin [33]), PriceWaze invertirá en contenido educativo en redes (ej. mini-vídeos explicando cómo interpretar si una propiedad está bien precio, usando datos reales anonimizados). También se puede lanzar un blog con estudios de mercado locales utilizando insights agregados de PriceWaze (posicionando la marca como experta en real estate analytics en RD). SEO: optimizar para búsquedas tipo "¿Cuánto ofertar por casa en [zona]?" o "precio justo m2 Santo Domingo". **Colaboraciones**: acercarse a **agencias inmobiliarias** locales y ofrecer demos, posicionando a PriceWaze como una herramienta que pueden usar con sus clientes para diferenciarse (esto crea evangelizadores en la industria). Incluso se puede ofrecer un período gratuito o plan freemium para agentes, animándolos a integrar PriceWaze en su flujo de trabajo de asesoría.

Proyecciones: Con la estrategia adecuada, PriceWaze puede capturar rápidamente usuarios early-adopter (tech-savvy, 25-45 años, buscando inmuebles). Un objetivo podría ser alcanzar X miles de usuarios en el primer año en RD, para luego expandir a mercados similares en la región. La tendencia global apunta a que quien ofrezca **datos en tiempo real y transparencia en transacciones** ganará la preferencia del público [34]. PriceWaze, al poner **"el proceso al desnudo"** con sus alertas y análisis, está alineado con esa expectativa, lo que es un poderoso ángulo de marketing: *"Conoce en tiempo real si es el momento de ofertar o esperar – ninguna otra plataforma local te lo dice."*

En resumen, marketing debe educar al mercado sobre esta nueva categoría de herramienta (copiloto inmobiliario), aprovechar las tendencias de confianza en IA + necesidad de asesoría, y posicionar a PriceWaze como sinónimo de **compra inteligente de propiedades**. Respaldado por datos y un mensaje claro de empowerment, el plan de marketing construirá tanto interés de consumidores finales como el apoyo del ecosistema inmobiliario local.

3. Producción (Despliegue y Operaciones)

Infraestructura de despliegue: Para llevar PriceWaze a producción se utilizará una infraestructura cloud robusta pero flexible. El frontend (React/Next.js) puede alojarse en una plataforma de *serverless hosting* moderna (por ejemplo, **Vercel** u **AWS Amplify**), optimizada para entregar contenido estático y páginas dinámicas con baja latencia global. El backend de IA (FastAPI + CrewAI) se puede contenerizar con Docker e implementar en un servicio de contenedores gestionado (por ejemplo, **AWS Fargate** o **Google Cloud Run**), que permita *auto-scaling*. Esto asegura que en horas pico (ej. lanzamientos de campañas de marketing o subidas de tráfico cuando se envían notificaciones masivas) haya suficientes recursos para manejar tanto las peticiones API como las inferencias de IA sin degradación de rendimiento.

Pipeline CI/CD: Desde temprano se configura una **pipeline de Integración Continua/Despliegue Continuo**. Cada *push* a la rama main puede gatillar tests automáticos y si pasan, desplegar a un entorno *staging* para validación. Una vez aprobado, se promueve a producción. Esta automatización minimiza errores humanos y permite entregas frecuentes. Se adoptan convenciones claras en Git (el proyecto ya sigue *Conventional Commits* para mensajes de commit estandarizados [35]), y *code reviews* obligatorios para garantizar calidad de código antes de fusionar cambios significativos.

Configuración y seguridad en producción: Se gestionan con cuidado las **variables de entorno y secretos**. Claves API (ej. la de DeepSeek u OpenAI para la IA) y credenciales de BD Supabase nunca se exponen en el código; en producción se almacenan en sistemas seguros (como AWS Secrets Manager o las propias env vars de Vercel). Asimismo, se habilitan *alerts* de seguridad: Supabase ofrece mecanismos de **Row-Level Security (RLS)** en la base de datos que se activan en producción para que cada usuario solo pueda acceder a sus datos donde corresponda [36]. Las políticas de acceso están definidas (ej. solo el dueño de una propiedad puede editarla, los demás solo la ven [37]). Esto se probó en stage para garantizar que la lógica de permisos es correcta antes del lanzamiento.

Escalabilidad y rendimiento: En producción se monitorizan métricas clave: uso de CPU/memoria del backend, tiempos de respuesta de las APIs, y latencia de llamadas a la IA (registrada en `pw_ai_logs.latency_ms`). Si se detecta carga creciente, el plan contempla escalar horizontalmente. Por ejemplo, correr múltiples instancias del microservicio de IA detrás de un balanceador de carga para atender más solicitudes concurrentes. La arquitectura stateless del backend (gracias a que la persistencia está en Supabase) facilita agregar instancias en paralelo. La base de datos se escala utilizando las herramientas de Supabase/Postgres (verticalmente con más CPU/RAM, o habilitando read replicas si fuese necesario para separar carga de lectura intensiva). Importante: dado que PriceWaze hará uso intensivo de lectura (mostrar propiedades, comparables, alertas) es fundamental que las consultas estén optimizadas. Ya en la migración inicial se incluyeron índices GIST para consultas geoespaciales y otros índices en campos de filtro comunes [11]. En producción se verificará constantemente planes de consulta para evitar *table scans* costosas. Si ciertas consultas analíticas (ej. cálculo de fairness score global) resultan lentas en vivo, se considerará precomputarlas offline y almacenarlas para lectura rápida (de hecho, ya se hace recálculo nocturno de insights [8] para este propósito).

Monitoreo y logging: Se implementa un sistema de monitoreo de aplicaciones en producción (por ejemplo, **NewRelic** o **Datadog** o incluso el dashboard de Supabase combinado con Sentry para el front). Esto permitirá recibir alertas en tiempo real de errores (excepciones no manejadas, fallos de infraestructura) y de métricas fuera de rango (p. ej., tiempo de respuesta de endpoint de análisis > X segundos, o tasa de error de API > Y%). Un archivo de *log* consolidado recogerá eventos importantes: cada generación de alerta, cada recomendación de oferta hecha, etc., junto con IDs de usuario (anonimizados) para posibilitar auditoría post-mortem de cualquier incidente.

Preparación de contenido y datos: Antes del *go-live*, el entorno de producción se llena con datos mínimos para utilidad inicial. Se migran las tablas definidas (ver sección DB) y se corre un *seed* script (ya existe uno que genera usuarios y datos de ejemplo [38]) para poblar zonas, algunas propiedades de muestra, etc., de manera que los primeros usuarios que ingresen vean contenido significativo. Adicionalmente, se podría integrar un servicio externo de listings (por ejemplo, en RD podría extraerse listado de propiedades de sitios públicos vía scraping o alianzas) para tener datos reales que analizar. Si se opta por scraping, se configura como proceso backend periódico (**logística de datos**, ver sección de Logística) para nutrir la BD continuamente.

Entorno de producción robusto: Se consideran detalles finales: habilitar **HTTPS** con certificados válidos, verificar que el dominio y DNS apunten correctamente a la web app. También optimizar la entrega de assets estáticos: comprimir imágenes (el logo fue optimizado en un sprint previo, reduciendo su tamaño 60% [39]), minificar JS/CSS, usar *caching* con headers adecuados para contenido que no cambia. En la capa de aplicación, se habilita una CDN para entregar imágenes de propiedades de forma rápida a los usuarios, dado que las fotos pueden ser pesadas.

En síntesis, la fase de producción se enfoca en confiabilidad y eficiencia: infraestructura escalable, despliegue automatizado, seguridad de datos, y monitorización proactiva. Todo ello asegurará que PriceWaze funcione de manera **estable** y **rápida** para cada usuario final, incluso a medida que la base de usuarios crezca.

4. Logística (Operaciones y Flujo de Trabajo)

En el contexto de PriceWaze, **logística** se refiere a la coordinación de todos los procesos y recursos necesarios para que el sistema funcione fluidamente de punta a punta. Implica manejar datos externos, agendar tareas, y asegurar que cada componente (dev, AI, BD, UX) interactúe sin fricción.

Flujo de datos y comparables: Un aspecto logístico clave es obtener y mantener actualizada la data inmobiliaria que alimenta al sistema. PriceWaze requiere datos de **comparables de mercado** (ej. propiedades similares en la zona con sus precios) para generar alertas como sobre/subro precio. Por ello, se implementa un pipeline de datos: puede involucrar integración con APIs de portales inmobiliarios locales o scraping automatizado. Idealmente se establecen **alianzas con fuentes** (brokers o MLS locales) para recibir listados actualizados. En ausencia de eso, un *crawler* puede recorrer sitios web populares para extraer nuevos listings diariamente. La información recolectada (precios de venta, m², ubicación, etc.) ingresa a la BD (tablas `pricewaze_properties` y `pricewaze_zones`). Se diseñan jobs automáticos para esto, que corren en horarios de baja carga (ej. de madrugada) para no interferir con el uso diurno. Tener datos de mercado robustos es fundamental para que las comparaciones de PriceWaze sean precisas.

Tareas programadas (cron jobs): Como mencionamos, PriceWaze realiza ciertos **cálculos en lote nocturnos**. Por ejemplo, recalcular el *fairness score* de todas las propiedades cada noche [8], entrenar modelos simples (si hubiera algún componente de ML interno basado en datos históricos), o enviar notificaciones resumidas. Se configura un scheduler (puede ser el servicio de *Cron* de Supabase o un cron job en la instancia backend) que desencadena un proceso cada 24h. Este proceso iterará por propiedades y actualizará la tabla `pw_property_insights` con datos nuevos (por ej., si surgieron nuevas ventas comparables que afectan el valor de mercado estimado). También puede ejecutar limpieza de datos (marcar alertas como expiradas o resolvidas si ya no aplican).

Un ejemplo de **lógica de alertas** que opera con datos actualizados: la alerta *"Zona en inflexión"* se dispara si en un barrio se detecta aumento de demanda y de visitas [40]. Para evaluar esto, el job nocturno podría calcular métricas de cada zona (p. ej., cuántas vistas tuvieron las propiedades de esa zona en la última semana vs la anterior) y si un umbral se excede, marcar internamente esa zona en tendencia alcista (H3). Luego, al evaluar cada propiedad, se sabrá si pertenece a una *zona en inflexión* y generar la alerta correspondiente. Todo este proceso muestra cómo la **logística de datos** y procesamiento por lotes permite que, cuando el usuario interactúa, la app ya tenga el conocimiento listo para mostrar (p. ej. "Zona en inflexión: esta área está ganando demanda rápidamente"). La tabla `pw_alerts` almacenará qué alertas (de los 7 tipos posibles) están activas para cada usuario-propiedad [41] [42], de modo que la app pueda consultarlas rápido.

Integración de eventos en tiempo real: Además de los procesos programados, la logística abarca los **eventos en tiempo real** que suceden en el flujo. Ejemplos: cuando un usuario hace una **oferta** por una propiedad, eso debería inmediatamente reflejarse en el sistema – posiblemente marcando una alerta para el comprador (si su oferta es muy baja comparada con patrones históricos, se podría disparar la alerta *"oferta subóptima"* [43]) o para el vendedor (si la oferta está cerca del patrón ganador, alertar *"oferta*

prometedora", etc.). Para manejar esto, se implementan *webhooks* o triggers en la BD. De hecho, hay triggers SQL que podrían ser aprovechados: por ejemplo, cuando se inserta una nueva oferta en `pricewaze_offers`, un trigger podría evaluar automáticamente las condiciones y poblar `pw_alerts` con la alerta correspondiente. Alternativamente, se envía un evento al backend Python, donde los agentes (p. ej. el Negotiation Advisor) analizan la oferta en contexto y generan la alerta con mayor sofisticación. La decisión entre lógica en BD vs lógica en app se toma según complejidad: condiciones simples (comparar números) pueden ir en el trigger de BD; análisis complejos (considerar mensajes adjuntos, historial de negociaciones) es mejor que los maneje el agente de negociación en Python. La logística consiste en orquestar estos mecanismos para que **nada "se quede en el aire"**: cada cambio importante en datos produce las acciones subsiguientes automáticamente.

Gestión de la carga de la IA: Dado que PriceWaze depende de llamadas a IA externas (LLM), se toman medidas logísticas para asegurar rendimiento consistente. Por un lado, se implementa un sistema de **cola (queue)** para las peticiones a la IA en caso de alta concurrencia. Si 100 usuarios solicitan simultáneamente análisis, en lugar de disparar 100 hilos que llamen a la API de OpenAI (riesgo de *rate limit*), las solicitudes pueden encolarse y procesarse con un número fijo de trabajadores, garantizando no exceder cuotas. Como parte de este sistema, se podrían priorizar ciertas peticiones (por ejemplo, dar mayor prioridad a respuestas interactivas del usuario en el chat vs a tareas batch menos urgentes). La **latencia** de cada respuesta de IA es crítica para UX; por ello, se monitoriza y cualquier respuesta que tarde más de X segundos puede activar un fallback (ej., mostrar un mensaje "Analizando datos…" con un *spinner* y permitir reintentar). Desde el punto de vista de operaciones, se debe también manejar las **claves de API** de IA: disponer de claves de respaldo o de proveedores alternos (p. ej., si OpenAI está saturado, usar DeepSeek u otro) para evitar downtime de la funcionalidad inteligente.

Soporte y mantenimiento: La logística post-lanzamiento incluye tener un plan de soporte al usuario. Se habilita un canal (chat o email) para que usuarios reporten problemas o hagan preguntas. Internamente, se asigna responsabilidad a alguien del equipo (quizás rotativo) para atender estas consultas, lo que alimentará también el backlog de mejoras. Se mantendrá un **registro de incidencias** y métricas de uso. Por ejemplo, cuántas veces al día se muestra cada tipo de alerta, o en qué punto del *funnel* de uso los usuarios abandonan (si muchos entran pero pocos usan la función de negociación asistida, quizás requiere mejoras en UX o en comunicación de valor). Estas métricas operativas guiarán ajustes logísticos: puede requerirse re-balanceo de la carga (más recursos a cierta hora pico), optimizaciones de consulta, o educación al usuario para adoptar funciones poco usadas.

Escalabilidad operativa: A medida que PriceWaze crezca (más usuarios, posiblemente expansión geográfica), la logística debe adaptarse. Por ejemplo, si se lanza en otro país, habrá que incorporar nuevas fuentes de datos locales y quizás ajustar las reglas (los "7 alertas" pueden requerir calibración distinta según mercado). Se planifica con anticipación la capacidad de agregar nuevas zonas/regiones en la base de datos (se usaron UUIDs y estructuras flexibles, como la columna JSONB `narrative` para poder soportar variaciones [44]). También se mantiene la **localización**: el contenido generado por IA debe poder entregarse en el idioma del usuario. En RD es español, pero si se expande a un mercado anglosajón, la IA debería cambiar a inglés. Dado que los agentes pueden producir lenguaje en cualquiera de los dos (DeepSeek modelo bilingüe, etc.), se implementa un parámetro de localización en las solicitudes de IA o se mantiene un perfil de idioma por usuario.

En resumen, la logística de PriceWaze se asegura de *engrasar* todas las piezas del motor: datos fluyendo de fuentes externas a la BD, triggers y jobs coordinados para actualizar insights, eventos en tiempo real

generando reacciones inmediatas, colas administrando recursos de IA, y monitoreo continuo de todos los eslabones. Este enfoque proactivo garantiza que la experiencia final sea consistente y que el sistema pueda escalar sin cuellos de botella operativos.

5. Interfaz de Usuario (UI)

La interfaz de PriceWaze debe ser **intuitiva, moderna y centrada en la información clave**. Se toma inspiración de referentes como Zillow en cuanto a familiaridad de diseño, pero adaptando a las necesidades locales y a la integración de la capa de IA. A continuación describimos los componentes y pantallas principales de la UI, siguiendo el diseño propuesto:

- **Botón Copiloto AI fijo:** En todo momento, el usuario tiene acceso al asistente. Se incorpora un botón flotante (por ejemplo, circular con ícono de robot o "AI") visible sobre las vistas principales: mapa de propiedades, página de detalle de propiedad, sección de ofertas [45]. Este botón permanece fijo en la esquina de la pantalla para fácil acceso. El estilo es consistente con el branding (por ejemplo, usando el gradiente cyan-emerald de la marca como fondo del botón para destacar). Al pulsarlo, abre la ventana del Copiloto.

- **Pantalla 1 – Copiloto (Overlay Chat):** Al activar el copiloto, aparece un overlay lateral o modal con el asistente AI [46]. **Diseño:** Un header claro que indica *"Tu copiloto está activo"* [47], seguido de una serie de **tarjetas automáticas** que el sistema genera como alertas destacadas. La UI de estas tarjetas es tipo notificación/card: por ejemplo, una tarjeta puede mostrar un ícono de advertencia ⚠ junto al título *"Oportunidad silenciosa"* y un texto breve: *"Esta propiedad está 11% bajo comparables similares."* [48], con quizás un botón o link *"Ver por qué →"* para profundizar. Estas tarjetas se listan una debajo de otra, priorizando las más importantes (la severidad puede indicarse con color de borde: rojo para alta, amarillo media, etc.). **Input de chat:** Debajo de las tarjetas, la UI ofrece un campo de texto opcional donde el usuario puede escribir una pregunta o indicación al copiloto (ej: *"¿Por qué dices que está bajo comparables?"*). Este campo se muestra sutilmente para no abrumar (ya que el sistema provee info proactivamente, quizá el usuario no necesite preguntar inicialmente). El diseño general es limpio, sin ruido, con tipografía legible y usando los colores de marca solo para acentos (no saturar de color de fondo). Este panel de copiloto puede ser ocultable para que no bloquee la visualización de la página detrás cuando el usuario quiera ver la propiedad en detalle.

- **Pantalla 2 – Historial de Precio (en la vista de Propiedad):** En la página de detalle de una propiedad, PriceWaze añade una sección dedicada a **insights de precio** en lugar de un simple "score frío" numérico [49]. La UI aquí presenta varios bloques:

- Un **Fairness Score** visual, probablemente en forma de indicador tipo velocímetro o un badge textual con color. Por ejemplo, *"+15% sobre mercado"* en color rojo indica que el precio listado está 15% por encima del valor de mercado estimado [50]. Si estuviera por debajo, podría mostrar "-10% vs mercado" en verde, etc. Esto permite al usuario entender de un vistazo la situación de precio.
- **Historia explicada:** Debajo o al lado, se muestran bullet points que resumen la historia de pricing de la propiedad [51]. Por ejemplo:
    - *"Vendedor redujo el precio 2 veces"*
    - *"Demanda estable en la zona"* [52]. Estos bullets provienen de la narrativa generada (campo JSONB narrative en BD) y la UI los presenta con íconos correspondientes (quizá un ícono de trending down para reducciones de precio, un ícono de group/personas para demanda, etc.).

- **Acción sugerida:** Un call-to-action destacado, por ejemplo en un cuadro con fondo tenue, que diga *"☞ Oferta sugerida: -12%"* [53] . Esto le indica al comprador qué oferta podría ser razonable dado el análisis. Ese CTA podría ser un botón que al hacer clic inicia directamente el flujo de hacer una oferta con ese porcentaje (prellenando un formulario de oferta con ese número, por ejemplo), facilitando al usuario seguir la sugerencia.

El estilo de esta sección equilibra información textual con visual. Se podría usar un pequeño gráfico de línea para historial de precio (mostrando cómo fue cambiando en el tiempo), complementando los bullets explicativos. Es vital que esta sección esté bien integrada en la página de propiedad, sin distraer de los datos básicos (fotos, descripción) pero visible lo suficiente ya que es un valor agregado único.

- **Pantalla 3 – Exploración Inteligente:** Esta es una vista de búsqueda/generación de recomendaciones asistida por IA [54] . En la UI puede implementarse como una sección dentro de la página de búsqueda o como un modal especial. **Prompt guiado:** Se muestra un texto sugerido como placeholder o en un recuadro: *"Muéstrame mejores opciones para mi objetivo"* [55] . Al activar este prompt (por ejemplo, el usuario hace clic en "Explorar" y el sistema automáticamente alimenta criterios del usuario – presupuesto, zona deseada, etc. – a la IA), la UI presenta un **listado de 3 a 5 propiedades recomendadas** [56] . Cada propiedad en esta lista se muestra como una mini *card* con su foto, precio, y un highlight: *"¿Por qué te conviene?"*. Debajo de cada card se pone una breve explicación generada: ej. *"Esta casa coincide con tu presupuesto y ha tenido reducciones de precio recientes, lo que indica un vendedor motivado"*. Esto corresponde al *"Por qué te conviene"* personalizado que la IA entrega. La interfaz debe permitir al usuario refinar: quizás mostrar filtros o permitir editar la pregunta (*"mejoras opciones"* puede cambiar a *"con 3 habitaciones en zona X"*). La clave es que esta exploración se sienta conversacional e interactiva, más que un filtro rígido. Visualmente, se podría resaltar cada recomendación con el color de marca en títulos o bordes, manteniendo consistencia estética.

- **Pantalla 4 – Negociación Asistida:** Integrada en la sección de Ofertas (donde un comprador realiza ofertas y un vendedor recibe/contra-oferta). Aquí la UI añade indicadores en cada oferta o contraoferta listada [57] . Concretamente, al lado de cada monto de oferta, se muestra un **semáforo de viabilidad**: un pequeño círculo o ícono coloreado en **verde, amarillo o rojo** según la calidad de esa oferta. Verde podría significar "buena oferta", rojo "oferta problemática" y amarillo algo intermedio. Junto a este semáforo se despliega un **texto corto de asesoría** [58] . Ejemplo dado: *"Buen número, mal timing"* [58] podría aparecer con semáforo amarillo, indicando que el monto es bueno pero quizá el momento (muy pronto en la negociación o mercado a la alza) no es óptimo. Otra oferta podría mostrar en verde *"Oferta alineada al valor sugerido"* o en rojo *"Muy por debajo del mercado"*. Este feedback inmediato en la interfaz de negociación ayuda a ambas partes a entender la posición de cada oferta sin tener que adivinar. La UI para lograrlo puede ser, por ejemplo, en la lista de ofertas (generalmente se ve algo como: Usuario X ofreció $Y el día Z) agregar una segunda línea con el texto de la AI y un icono. Si el usuario hace hover o clic, podría ampliarse más detalle si disponible (ej. "mal timing" quizá al hacer clic explica "El vendedor redujo precio recientemente, esperar podría ser beneficioso"). Debemos mantenerlo simple para no saturar la lista de ofertas, por eso usar microtextos con colores es acertado.

- **Consistencia y branding:** A nivel global, la UI sigue la identidad visual definida. Según la última iteración, se adoptó un **esquema de color gradiente cian-esmeralda** aplicado en elementos clave [59] . Por ejemplo, los botones primarios y títulos presentan este gradiente, que transmite

innovación. La interfaz fue ajustada para parecer familiar a los usuarios: en un sprint se rediseñó el header al estilo Zillow [60] con un logo compacto a la izquierda, búsqueda prominente en el centro y menú de usuario a la derecha, lo que facilita la navegación porque coincide con expectativas del usuario. Los cambios de ese sprint también aseguraron que el **logo** esté presente y optimizado (el logo original se recortó y redujo de tamaño para carga rápida [39]). Se implementaron mejoras en muchos componentes visuales (por ejemplo, badges con gradiente, efectos hover con color de marca, etc. [61] [62]) para dar un look & feel pulido y coherente.

- **Responsividad:** La interfaz está diseñada mobile-first, considerando que muchos usuarios navegan desde móviles. Se aseguran breakpoints adecuados: en pantallas pequeñas, el botón de Copiloto sigue accesible (quizá como parte del menú inferior tipo floating action button), las tarjetas de alertas se vuelven carrusel vertical en pantalla completa, etc. El header y las listas responden bien (el sidebar se colapsa en móvil, se utilizan componentes responsive – Next.js `responsive` o tailwind classes). Un ejemplo: el layout de Dashboard en desktop mostraba sidebar y header, en móvil se adaptó a un menú hamburguesa para la sidebar [63] [64].

- **Accesibilidad:** Se siguen guías de accesibilidad web (WCAG). Colores con suficiente contraste (los gradientes se probaron para que texto blanco encima sea legible). Se agregan etiquetas alt a las imágenes (p. ej., foto de propiedad con alt descriptivo). El aplicativo es navegable con teclado (impotante para usuarios con discapacidades). Las nuevas funcionalidades de AI se integran de forma que no dependan solo del color: por ejemplo, el semáforo de negociación incluye también el texto para distinguir, y los usuarios daltónicos puedan leer *"Buen número"* vs *"Número riesgoso"* en lugar de color únicamente.

En definitiva, la UI de PriceWaze combina **elementos tradicionales de portales inmobiliarios** (búsqueda, listados, fichas) con **componentes innovadores** de IA (tarjetas inteligentes, semáforos, recomendaciones dinámicas). El desafío de diseño se ha abordado manteniendo la simplicidad: pocos clics para acceder a lo importante, visualizaciones claras de datos complejos y estética profesional. Al lograr esto, la interfaz no solo es atractiva sino que incrementa la confianza del usuario en las recomendaciones (una presentación limpia y fundamentada hace que la gente confíe más en la info). Cada pantalla fue concebida para agregar valor concreto al proceso de compra, sin abrumar, que es justo la promesa central de PriceWaze como copiloto.

6. Experiencia de Usuario (UX)

La **UX** de PriceWaze busca proporcionar al usuario una sensación de **seguridad, claridad y control** durante todo el proceso de búsqueda y compra de una propiedad. Dado que incorporamos una capa de IA, es fundamental que la experiencia esté cuidadosamente diseñada para generar confianza y facilitar la comprensión de las recomendaciones. A continuación, se detallan los principios y prácticas UX aplicadas:

Proactividad sin fricción: PriceWaze adopta la filosofía de que el asistente debe brindar ayuda sin que el usuario la tenga que pedir explícitamente. Por eso *"el sistema habla primero"*, mostrando alertas útiles en cuanto se da contexto [7] (por ejemplo, al abrir una propiedad). Este enfoque proactivo aporta valor inmediato y guía al usuario desde el primer momento. Desde la perspectiva UX, esto reduce la carga cognitiva: el comprador no tiene que decidir *qué* preguntar o *qué* analizar, sino que la herramienta ya resalta lo relevante. Sin embargo, es clave implementar esto sin abrumar. La solución ha sido limitar el número de *cards* mostradas a solo las más importantes y con mensajes concisos. Cada tarjeta de alerta se

diseña con máximo una o dos líneas de texto [48], en lenguaje sencillo, para que el usuario las pueda escanear rápidamente. Además, ofrecer *"Ver por qué →"* da control al usuario de profundizar solo en lo que le interese, manteniendo la interacción ligera inicialmente.

Narrativa clara y comprensible: Un valor agregado de PriceWaze es traducir datos complejos a lenguaje natural amigable. En UX se decidió seguir la regla de *"3 bullets"* para explicaciones complejas [65]. Por ejemplo, en la historia de precio de una propiedad, la IA resume la situación en **tres viñetas claras** [52]. Estudios muestran que los usuarios retienen mejor información presentada en listas breves y categorizadas. De hecho, en la implementación se usa un prompt específico a la IA: *"Explica esto a un comprador no técnico en 3 bullets"* [66]. Esto asegura que el tono sea apropiado (evitando jerga técnica o estadística) y que la extensión sea manejable. La UX writing (redacción) se enfoca en **beneficios y contexto**: en vez de decir "Fairness score = 65", se dice "Precio alineado con el mercado" o "15% sobre mercado, podría negociarse a la baja". Este tipo de microcopy orientado a acciones y consecuencias facilita que el usuario entienda qué debe hacer con la info.

Confianza y transparencia: Son pilares de la UX ya que involucramos IA y toma de decisiones financieras importantes. Para fomentar confianza: - Siempre que la IA da un juicio (ej. "esta oferta es riesgosa"), proveemos una justificación o contexto detrás. Por ejemplo, junto al semáforo rojo en una contraoferta, el texto *"Oferta muy baja comparada con últimas ventas"* explica la razón. La transparencia se refuerza permitiendo al usuario *ver los datos* detrás de una recomendación cuando lo desee. Si el usuario hace clic en "Ver por qué", la UX podría mostrar un pop-up con comparables específicos o con el cálculo del fairness (ej. lista de 3 propiedades comparables y sus precios). Esto sigue la tendencia de **real-time transparency** en productos proptech: los consumidores quieren poder verificar por sí mismos los datos [20]. Dar esa opción aumenta la credibilidad del sistema. - Se incluyen avisos de *disclaimer* donde corresponda. Por ejemplo, en la generación de contratos se aclara que son borradores no vinculantes [67], y en general se recuerda al usuario que las recomendaciones son asistencias informativas, no garantías. Esta honestidad en la UX evita expectativas irreales y posiciona a PriceWaze como un consejero honesto.

Control del usuario: Aunque la IA sea proactiva, el usuario siempre mantiene el control final. La UX lo garantiza de varias formas. Primero, las alertas AI son **no intrusivas**: el usuario puede ignorarlas si quiere (por ejemplo, cerrar la tarjeta de alerta). Segundo, si el usuario prefiere explorar manualmente sin la ayuda, PriceWaze no lo obstaculiza; las funciones estándar (buscar, filtrar propiedades) funcionan normalmente. El asistente está "ahí cuando lo necesites". Además, cuando el usuario interactúa con el copiloto (por ejemplo, haciendo una pregunta específica en el chat), el sistema responde pero también **sugiere el siguiente paso** en base a eso, en lugar de forzar un flujo rígido. Por decir, si el usuario pregunta "¿Qué oferta debería hacer?", el copiloto no solo sugiere un número sino que podría ofrecer botones en la interfaz: *"Realizar oferta de X% menos"* directamente. Esto da control (el usuario decide si hacer clic o no) pero facilita la acción siguiente, creando una continuidad natural.

Personalización y aprendizaje del usuario: Una buena UX aprende del usuario con el tiempo. PriceWaze incorpora el concepto de *perfil de comprador* o `pw_user_twin` con rasgos como tolerancia al riesgo, sensibilidad al precio, velocidad de decisión [68]. En la experiencia, esto se refleja en personalizar las recomendaciones. Por ejemplo, un usuario con **tolerancia al riesgo baja** recibirá alertas más conservadoras (priorizando seguridad, quizá más alertas de riesgo oculto), mientras uno con alta tolerancia verá más *"oportunidades"* atrevidas. Esta personalización aumenta la relevancia de la UX. Debe ser sutil; en la configuración de la cuenta, el usuario puede ajustar sus preferencias (ej. "Prefiero inversiones seguras vs. me gusta buscar gangas"). La UX podría incluso gamificar esto al inicio: un breve cuestionario estilo "¿Qué

tipo de comprador eres?" para ajustar el perfil. De esta manera, desde el primer uso se configuran las expectativas y el tono del copiloto acorde al usuario.

Feedback y mejora continua: La experiencia de usuario se refina escuchando a los usuarios. Se habilitan dentro de la UI pequeños momentos para *feedback*. Por ejemplo, después de mostrar una recomendación de oferta, se podría preguntar al usuario "¿Te resultó útil esta sugerencia?" con pulgar arriba/abajo. O permitir calificar las recomendaciones. Este feedback se envía al equipo y también puede alimentar a la IA (por ejemplo, reforzando qué tipo de explicaciones funcionan mejor). Durante pruebas con usuarios (fase beta), se observará cómo interactúan: si ignoran ciertas alertas, quizá esas alertas necesitan re-formularse o presentarse en otro momento. Un hallazgo UX común es *no abrumar al usuario novel*: por eso, podríamos escalonar funcionalidades. Tal vez en la primera propiedad que un usuario nuevo ve, solo le mostramos 1-2 alertas principales para no sobrecargarlo. Ya en usos posteriores, al ver que interactúa cómodamente, se pueden mostrar más datos. Este enfoque progresivo mejora la adopción.

Consistencia en UX cross-platform: Se asume inicialmente una aplicación web responsive. Si en el futuro hay app móvil nativa, la UX debe ser consistente. El flujo de copiloto, las metáforas (tarjetas, semáforos, bullets) se mantienen similares. Esto para que un usuario pasando de web a móvil no tenga que *reaprendender* las interacciones. Las guías de estilo UX se documentan (formato de botones, modales, etc.) para asegurar coherencia.

En resumen, la experiencia de usuario de PriceWaze está diseñada para **empoderar sin abrumar**. Le proporciona al comprador información valiosa en forma digerible, le ahorra trabajo de análisis pero siempre dejándole tomar las decisiones. La interacción con la IA se hace de forma transparente y confiable, creando en el usuario la sensación de estar acompañado por un experto amable durante todo el journey. Una UX bien lograda es la que convierte esa primera impresión de *"¿Será fiable esta recomendación?"* en *"¡Qué bien tener esta herramienta, me siento más seguro!"* – y ese es el objetivo con PriceWaze.

7. Base de Datos (DB)

La base de datos de PriceWaze, implementada sobre **PostgreSQL (Supabase)**, está diseñada para soportar tanto las funcionalidades tradicionales de un sistema inmobiliario como las nuevas características impulsadas por IA. El esquema de datos fue pensado para ser **normalizado, seguro y extensible**. A continuación describimos las entidades principales y sus interrelaciones, junto con consideraciones de diseño:

- **Usuarios y Perfiles:** Se utiliza la tabla `auth.users` propia de Supabase para la autenticación básica, extendida por una tabla de perfiles `pricewaze_profiles` que almacena datos adicionales (nombre completo, teléfono, rol, etc.) [69]. Cada usuario tiene un rol (comprador, vendedor, agente, admin) definido por un ENUM [70] [71]. Esto permite segmentar comportamientos en la app (por ejemplo, un comprador puede ver ciertas alertas, un agente otras). La política de seguridad (RLS) se configura para que un usuario solo pueda modificar su propio perfil [72], manteniendo privacidad.

- **Propiedades y Zonas:** Tenemos una tabla central `pricewaze_properties` que contiene todos los inmuebles listados [73]. Incluye campos tradicionales (tipo: casa, apto, terreno, precio, metros, características, etc.) y columnas para geolocalización (`latitude`, `longitude` y una columna geometry `location` con PostGIS) [74]. Esto posibilita hacer búsquedas geoespaciales y cálculos de

distancias si se necesitara. Cada propiedad referencia a su **zona** (`zone_id` apuntando a `pricewaze_zones`) y al **propietario/vendedor** (`owner_id` apuntando al perfil) [75]. La tabla `pricewaze_zones` representa barrios o zonas geográficas, con un polígono de área (`boundary` geometry) y datos agregados como precio promedio por m² en esa zona [76]. Un trigger en la BD se encarga de asignar automáticamente la zona a una propiedad nueva basándose en su lat/long: antes de insertar una propiedad, la función `pricewaze_auto_assign_zone()` busca qué polígono de zona contiene ese punto y asigna el `zone_id` correspondiente [77] [78]. Esto reduce errores y hace la entrada de datos más sencilla (el usuario no tiene que elegir zona manualmente). Además, si los límites de zonas cambian o se añaden zonas, con actualizar esa tabla de zonas los nuevos datos se asignarán en adelante.

- **Historial de Precios:** Para rastrear cómo cambia el precio de una propiedad a lo largo del tiempo, existe la tabla `pricewaze_property_price_history` [79]. Cada vez que una propiedad cambia de precio, se inserta un registro con el nuevo precio, precio/m² y timestamp. Esto se logra con un **trigger** en la tabla de propiedades (`pricewaze_properties_price_history`): tras cada UPDATE, si el precio ha cambiado, ejecuta la función `pricewaze_track_price_history()` que inserta el histórico [80] [81]. Gracias a esto, PriceWaze puede contar con datos para la "historia de precio" mostrada al usuario, sin requerir acciones manuales de los agentes o admins para registrar las variaciones. Esta tabla alimenta la narrativa de pricing (ej. *"Vendedor redujo precio 2 veces"* se deriva de contar registros de historial [52]).

- **Ofertas y Negociación:** Las ofertas de compra se almacenan en `pricewaze_offers` [82]. Tiene campos para monto, mensaje adjunto, estado (pendiente, aceptada, rechazada, contraoferta, etc. definido en un ENUM) [83] [84], así como referencias al comprador, vendedor y propiedad involucrados [85]. Importante, hay un `parent_offer_id` que permite encadenar contraofertas haciendo referencia a la oferta previa [84]. Esto modela una negociación como un hilo de ofertas y contraofertas. La lógica de negocio puede usar este campo para agrupar ofertas (p. ej., saber cuál fue la oferta inicial de la cadena y cuáles son réplicas). En cuanto a reglas de acceso, solo el comprador o vendedor involucrados pueden ver una oferta [86], y solo el comprador puede crear una oferta nueva (con check de que `auth.uid() = buyer_id` en la política) [87]. Así, un usuario malicioso no puede ofertar en nombre de otro.

- **Visitas y otras entidades:** Para completar las funcionalidades tradicionales, `pricewaze_visits` registra las visitas programadas a propiedades [88] (quién visita, cuándo, estado – agendada, realizada, cancelada – con un código de verificación para confirmar la visita in-situ). `pricewaze_favorites` guarda las propiedades que un usuario marcó como favoritas [89] (evitando duplicados con una UNIQUE compuesta en user_id+property_id). `pricewaze_notifications` almacena notificaciones generales para usuarios (título, mensaje, tipo) [90], por ejemplo avisos de que su oferta fue respondida, etc. Muchas de estas tablas cuentan con políticas RLS apropiadas (ej. solo el dueño de un favorito puede borrarlo, solo involucrados ven sus visitas agendadas) para mantener la privacidad y seguridad.

- **Tablas específicas de PriceWaze (AI/Insights):** Aquí es donde se extiende el esquema para dar soporte al copiloto y sus análisis:

- Tabla `pw_user_twin`: diseñada para capturar el perfil "psicológico" o preferencial del usuario [91]. Contiene campos como `risk_tolerance` (0-100), `price_sensitivity` (0-100) y `decision_speed` (lento/medio/rápido), que ayudan a personalizar la experiencia. Esta tabla se relaciona 1:1 con el usuario (llave foránea user_id). Inicialmente puede poblarse con valores por defecto o a través de un cuestionario inicial al usuario. Su mantenimiento es ligero (tal vez se actualiza cuando el usuario ajusta sus preferencias).
- Tabla `pw_property_insights`: es central para las características de IA [92]. Cada registro corresponde a una propiedad (llave foránea property_id) y guarda los resultados del análisis automatizado: un *fairness_score* (puntuación de equidad del precio), cuánto porcentaje estaría sobreprecio o bajo precio (columnas `overprice_pct` y `underprice_pct`), y un campo `narrative` tipo JSONB con la explicación estructurada [93]. Un ejemplo de registro podría ser: fairness_score=75, overprice_pct=+15%, underprice_pct=0%, narrative = {"bullets": ["+15% sobre mercado", "Vendedor redujo 2 veces", "Demanda estable"], "suggested_action": "Oferta -12%"} [94] [44]. Estos datos se recalculan periódicamente (por los jobs mencionados) y se muestran directamente en la UI de la propiedad. Es más eficiente tenerlos precomputados en la BD a que calcular todo al vuelo para cada usuario.
- Tabla `pw_alerts`: esta tabla registra las alertas generadas para usuarios y propiedades específicas [95]. Campos principales: `user_id`, `property_id`, tipo de alerta (`alert_type` texto que puede ser uno de los 7 tipos definidos, por ejemplo "Sobreprecio emocional", "Oportunidad silenciosa", etc.), severidad (bajo/med/alto), mensaje descriptivo, y si ya fue resuelta o atendida [96]. El *frontend* consultará esta tabla para saber qué alertas mostrarle al usuario en cada contexto (por ejemplo, al ver la propiedad X, cargar alertas donde property_id = X y user_id = actual). Se puede depurar automáticamente: por ejemplo, una vez que el usuario hace una oferta, una alerta de "bajo precio desaprovechado" podría marcarse como resuelta.

- Tabla `pw_ai_logs`: dedicada a loguear las interacciones con IA [14]. Cada entrada guarda el `user_id`, un campo de contexto (p. ej. "view_property" o "nego_offer"), el *prompt* de entrada enviado a la IA, el *output* devuelto, la latencia en milisegundos y timestamp [97]. Esto es invaluable para depurar y auditar: permite revisar qué respondió la IA en tal caso si un usuario se queja de un consejo erróneo, y ajustar en consecuencia. Dado que puede almacenar texto potencialmente sensible, se maneja con cuidado (posiblemente limpiando PII si se llega a guardar). Esta tabla crecerá con el tiempo, así que quizá se deba implementar rotación o archivado de logs antiguos para no impactar rendimiento.

- **Optimización y seguridad de la BD:** El diseño incluye varias optimizaciones. Múltiples índices fueron creados para acelerar búsquedas comunes: por ejemplo, hay índices GIST para geometrías (para consultas por ubicación), índices por zona, por status, por precio, etc., tal como definido en la migración inicial [11]. Esto permite filtrar propiedades por zona o rango de precio eficientemente en las búsquedas. Por el lado de seguridad, todas las tablas (excepto quizás las de solo lectura pública como zonas) tienen Row Level Security activado [36] con políticas específicas para cada acción [37] [86]. Asimismo, se implementó un trigger en `auth.users` (tabla de Supabase) para crear automáticamente un perfil en `pricewaze_profiles` cuando un usuario se registra [98] [99]. Esto garantiza consistencia de datos (todo usuario autenticado tiene su perfil en nuestro esquema).

- **Extensibilidad:** La estructura permite añadir columnas o tablas conforme se expandan funcionalidades. Por ejemplo, si en el futuro se quiere trackear métricas de mercado a nivel zona (inventario, velocidad de ventas), se puede extender `pricewaze_zones` con campos adicionales o

crear una tabla de estadísticas temporales por zona. El uso de JSONB en `pw_property_insights.narrative` también da flexibilidad para almacenar más detalles de explicación sin necesidad de migraciones de esquema frecuentes. Solo habría que acordar un formato JSON para que frontend e IA lo entiendan. Igualmente, la tabla de `pw_alerts` con un campo `alert_type` textual es flexible para agregar nuevos tipos de alertas en el futuro sin cambiar la estructura (los "7 tipos" actuales [41] podrían crecer a 8,9 si se descubren nuevos patrones, solo se documentarían los nuevos tipos y lógica).

En conclusión, la base de datos de PriceWaze constituye un **fundamento sólido** donde los datos operacionales (propiedades, ofertas, etc.) se entrelazan con los datos analíticos (insights, alertas, perfiles AI) de forma coherente. La arquitectura de datos soporta eficientemente las consultas de la app, mantiene integridad (gracias a foreign keys y triggers), y garantiza confidencialidad mediante RLS. Esto permite que las capas superiores (backend AI y frontend) funcionen con datos confiables y disponibles rápidamente, que en definitiva es crítico para ofrecer una experiencia fluida al usuario final.

8. Mejores Prácticas (Best Practices)

En el desarrollo e implementación de PriceWaze se han seguido y se seguirán una serie de **mejores prácticas** para asegurar calidad, mantenibilidad, y ética en el producto. Estas buenas prácticas abarcan desde la ingeniería de software hasta consideraciones de negocio y usuario. A continuación, enumeramos las más destacadas:

- **Metodología Ágil y Entrega Iterativa:** El proyecto se gestiona mediante *sprints* (ver Plan de Implementación abajo) con entregas incrementales. Cada sprint produce un incremento funcional (MVP, luego mejoras UI, luego módulo de IA, etc.), permitiendo obtener feedback temprano y adaptarse. Se realizan *dailies*, *reviews* y *retros* breves para mantener al equipo sincronizado y mejorar el proceso continuamente. La planificación ágil evita intentar construir todo de una vez, mitigando riesgos y asegurando que el producto siempre esté en un estado utilizable al final de cada iteración.

- **Control de Versiones y Colaboración:** Todo el código reside en GitHub, aprovechando *Pull Requests* para integrar cambios con revisión por pares. Los commits siguen un formato convencional descriptivo (*feat:*, *fix:*, etc. [35]) para generar automáticamente changelogs. Se protegen las ramas principales (main) requiriendo aprobación antes de *merge*, lo que previene introducción de código de baja calidad en la base principal. Además, se utilizan herramientas de integración continua para correr pruebas y *linters* en cada PR, manteniendo la base de código limpia y funcionando.

- **Testing exhaustivo:** Como mencionado, existe una suite de pruebas automatizadas. Se practica **TDD (Desarrollo guiado por pruebas)** en los componentes críticos de negocio: por ejemplo, la lógica que calcula las alertas se implementa primero con casos de prueba que cubren distintas combinaciones (propiedad muy sobrevalorada, subvalorada, oferta aceptable vs muy baja, etc.) para verificar que se disparen las alertas correctas. Igualmente, las funciones de base de datos, como triggers SQL, se validan en un entorno de prueba para comprobar que insertan/actualizan lo esperado (ej., tras actualizar una propiedad, un test verifica que la tabla de historial tenga un nuevo registro). También se incluyen **pruebas de integración** para la capa de IA: simulando una llamada al endpoint de análisis completo y comprobando que coordina los agentes y devuelve un resultado estructurado válido. Por último, pruebas manuales exploratorias se realizan periódicamente,

especialmente enfocadas en la UX (usabilidad, textos entendibles, etc.), con participantes internos o beta testers.

- **Documentación y Transparencia:** Toda nueva funcionalidad viene acompañada de documentación. En el repo se mantiene un **Sprint Log** [100] detallando cada sprint completado, con su fecha, objetivos y resultados. Esto ayuda al equipo (y a futuros desarrolladores) a comprender el histórico de decisiones y cambios. Asimismo, se documenta la arquitectura (como el README de CrewAI [101] que describe la estructura de carpetas y roles de agentes) y se provee un **Manual de Usuario** (puede ser una página de ayuda en la app) explicando cómo usar el copiloto, qué significan las alertas, FAQs, etc. Esta documentación de cara al usuario es importante para manejar expectativas y educar en las nuevas funcionalidades de IA.

- **Seguridad y Privacidad:** PriceWaze maneja datos sensibles (p. ej., decisiones de oferta, datos personales básicos). Se adhiere a principios de *Privacy by Design*. Solo se recolectan datos necesarios y se protege la información personal: la comunicación está cifrada (HTTPS), las contraseñas nunca tocan nuestros servidores (Supabase Auth se encarga con métodos seguros), y la información de comportamiento del usuario (por ejemplo, qué preguntas le hace a la IA) se utiliza únicamente para mejorar el sistema, nunca se comparte con terceros sin consentimiento. Además, cualquier integración con IA considera la privacidad: si se envía información de una propiedad a un LLM, se evita incluir datos de identificación personal o direcciones exactas del usuario. En los términos y condiciones (a mostrar al usuario en registro) se aclara el uso de IA y se obtienen consentimientos necesarios. También se sigue el principio de mínimos privilegios en la arquitectura: las keys de servicio de Supabase se usan solo en backend seguro, mientras que el frontend usa solo la key pública con reglas RLS para que no pueda extraer más datos de los permitidos.

- **Uso Ético de la IA:** Con PriceWaze ayudando en transacciones importantes, es crucial mantener la IA dentro de límites éticos. Se definen claramente *qué no hace* la IA [15] : por ejemplo, no debe inventar datos ni ofrecer consejos sin sustento en información real. Si el sistema no tiene suficientes datos sobre una propiedad, en lugar de que la IA "improvise", se le indica que admita no tener certeza. También se programan las *prompts* de forma que la IA no haga **discriminación ni sesgos**. Por ejemplo, no se harán recomendaciones basadas en características sensibles (raza/etnia del vecindario, etc., lo cual sería antiético y probablemente ilegal); la IA se concentra en datos de mercado objetivos. Se entrena al modelo (o más bien, se le proporcionan instrucciones) para que su tono sea **imparcial y objetivo**, similar a cómo un asesor profesional actuaría. Cualquier consejo que pudiera tener implicaciones legales (ej. contratos) viene acompañado de disclaimers que animan a consultar a un profesional humano [67] .

- **Mejoras continuas y Best Practices de Código:** A nivel de código, se sigue <u>Clean Code</u>: funciones pequeñas y bien nombradas, componentes React desacoplados (por ejemplo, varios componentes UI fueron creados en módulos reutilizables en una sprint [102] para pagination, slider, popover, etc., fomentando la reutilización y consistencia). Se mantienen convenciones de estilado (linters para JS/TS, formateo con Prettier, PEP8 para Python). Se evita la duplicación de lógica: la regla DRY (Don't Repeat Yourself) se aplica consolidando en funciones utilitarias cualquier cálculo usado en varios lugares (por ejemplo, cálculo de porcentaje de sobreprecio centralizado para ser usado tanto en backend como en frontend si se requiere). También se siguen patrones de diseño donde son útiles: p.ej., el patrón Strategy para los diferentes agentes de IA (cada agente es una estrategia

de análisis especializada), el patrón Observer para notificaciones (escuchar eventos de BD y notificar).

- **Performance y costo:** Otra buena práctica es ser consciente de la eficiencia de las soluciones. Se revisan las complejidades de algoritmos (aunque la mayoría de la lógica es sencilla, se vigila no hacer loops innecesarios sobre cientos de propiedades en la API de manera sin control; en su lugar, delegar a la base de datos que es más eficiente en set-based operations). Con el uso de IA, se monitorean costos de API y se ponen límites: ej., máximo N llamadas de IA por usuario por minuto, y si se excede se cola o se avisa. Se implementa *caching* de respuestas de IA cuando tiene sentido (por ejemplo, si muchos usuarios distintos piden análisis de la misma propiedad en poco tiempo, se puede reutilizar la narrativa calculada en la última hora en lugar de recalcular). Estas prácticas mantienen el producto sostenible económicamente y rápido en respuesta.

- **Accesibilidad e Inclusividad:** Ya mencionado parcialmente en UI/UX, pero se considera también una best practice: asegurarse que la herramienta sea utilizable por la mayor cantidad de personas. Se realizan revisiones de accesibilidad (ej. usando Lighthouse o herramientas automatizadas) para cumplir con estándares. En inclusividad, se eligen textos y ejemplos que no alienen a ningún grupo. Por ejemplo, en la escritura de la IA, se procura un lenguaje neutro y respetuoso. Internamente, en el equipo, se fomenta una cultura de inclusión y se aplica también al producto (por ejemplo, ilustraciones y imágenes de marketing que reflejen diversidad de usuarios).

- **Monitoreo y Respuesta Rápida:** Post-lanzamiento, se sigue la práctica DevOps de "You build it, you run it". El equipo de desarrollo asume la rotación de guardias para atender incidentes en producción. Con las alertas configuradas (errores, tiempos de respuesta), hay procedimientos listos: playbooks que indican qué hacer si la BD se queda sin espacio, o si el servicio de IA empieza a fallar, etc. Esto es parte de la excelencia operativa: anticipar fallas y tener planes de contingencia (por ejemplo, si el servicio de IA está caído, temporalmente degradar la experiencia mostrando solo datos sin narrativa en lugar de dejar de funcionar por completo, y comunicar en un banner “El asistente AI no está disponible temporalmente”).

En resumen, estas mejores prácticas garantizan que PriceWaze no solo se construya bien, sino que **siga funcionando bien y de forma responsable** a lo largo del tiempo. Al adoptar estándares altos en desarrollo, pruebas, seguridad y ética, el equipo se asegura de entregar un producto confiable y respetado tanto por usuarios finales como por stakeholders (inversionistas, socios). Es esta disciplina la que permitirá escalar el proyecto sin comprometer calidad.

9. Plan de Implementación por Sprints (Auditoría Final)

A continuación se presenta un plan detallado de implementación, dividido en sprints quincenales (de ~2 semanas cada uno), que abarca el desarrollo técnico, así como consideraciones de marketing, producción y UX en cada fase. Cada sprint tiene objetivos claros y entregables, de modo que al finalizar todos, PriceWaze habrá pasado de concepto a lanzamiento exitoso. Esta planificación también sirve de **auditoría**, asegurando que cada departamento (dev, marketing, etc.) tenga tareas y que nada quede desatendido.

1. **Sprint 1: Planeación y Fundamentos**
   Objetivo: Sentar las bases del proyecto en todos los frentes antes de codificar en firme.

2. *Desarrollo:* Definir la arquitectura técnica en detalle. Decisiones de stack (confirmar Next.js + Supabase + FastAPI), estructura de repositorio. Configurar el repositorio GitHub e integrar herramientas iniciales (linters, formateo, CI básico). Escribir un **PRD (Product Requirements Document)** inicial que describa funcionalidades core (puede basarse en el doc PriceWaze v1 [103]). Comenzar a implementar el esquema de base de datos en un entorno de prueba: crear migraciones para tablas principales (usuarios, propiedades, ofertas, etc.) [69] [73] e instalar Supabase local para test.
3. *Marketing:* Realizar un pequeño *study* de mercado local (encuestas o entrevistas exploratorias con potenciales usuarios: agentes inmobiliarios, 2-3 compradores recientes) para refinar el mensaje de valor. Desarrollar la identidad visual: logo (si no existe, aunque ya hay indicios), paleta de colores (se optó por cyan-emerald, validar que funciona en distintos medios). Reservar dominio web para PriceWaze y crear cuentas en redes sociales reservando el nombre.
4. *Producción/Infra:* Montar los entornos de desarrollo. Configurar un servidor de staging (por ej., subdominio staging.pricewaze.com) para futuros despliegues. Escribir un plan de gestión de configuraciones: dónde estarán las variables de entorno en cada entorno. No se despliega nada productivo aún, pero se deja lista la pipeline CI y una página "Hola Mundo" simple para verificar el flujo de despliegue hasta staging.

5. *UX/UI:* Crear wireframes de las principales pantallas (mapa/búsqueda, detalle propiedad con sección de insights, panel del copiloto, sección de ofertas). Esto se puede hacer en Figma u otra herramienta. Hacer un **prototipo navegable** básico para alinear a todos en la visión UI/UX. Comenzar a definir el tono de voz de la aplicación y sus textos (microcopys) con marketing.
   Deliverables Sprint 1: Documento de arquitectura aprobado, esquema de BD inicial migrado en dev, prototipo UI en baja fidelidad, backlog priorizado de funcionalidades para próximos sprints, estrategia de marca inicial (logo, colores, eslogan preliminar).

6. **Sprint 2: MVP Core (Listado y Funciones Básicas)**
   Objetivo: Desarrollar la funcionalidad básica de una plataforma inmobiliaria sin IA, que servirá de base para integrar luego el copiloto. Al mismo tiempo, avanzar en marketing preparatorios.

7. *Desarrollo Backend:* Implementar servicios CRUD básicos: endpoints para listar propiedades, ver detalle, crear cuentas de usuario, login (aunque Supabase provee mucho de esto, integrar el SDK en frontend). Desplegar la base de datos inicial en Supabase cloud y probar consultas. Crear algunos *seed scripts* para popular la BD con datos de ejemplo (10 usuarios, varias propiedades, etc. como ya se hizo [104]). Verificar triggers importantes (auto-asignación de zona, historial de precios) funcionan correctamente en entorno real [77] [12].
8. *Desarrollo Frontend:* Maquetar las páginas principales usando los wireframes. Implementar la **landing/landing page** con explicación del producto (para uso de marketing incipiente). Implementar página de **búsqueda de propiedades** con filtros por precio, zona, tipo (consultando la BD vía Supabase orquestrada por el backend). Implementar página de **detalle de propiedad** mostrando información básica (sin aún la sección de IA, que estará vacía o con placeholder "Próximamente análisis de PriceWaze"). Añadir funcionalidad de marcar favoritos y agendar visitas, con botones que llamen al backend (estos features clásicos se pueden terminar en sprints posteriores, pero dejar el esqueleto).
9. *Marketing:* Desarrollar la **landing page** con contenido orientado a early adopters. Por ejemplo, incluir un formulario de "Regístrate para la beta" o "Únete a la lista de espera" para empezar a captar leads interesados. Iniciar presencia en redes: publicar teaser de lo que viene ("Pronto: tu copiloto en bienes raíces"). Generar un pequeño **estudio comparativo** para contenido, e.g. un blog post

"¿Cuánto sobrevaloran su casa los vendedores en Santo Domingo? – Presentamos PriceWaze" usando algunos datos fake o iniciales para llamar la atención.

10. *Producción:* Hacer el **deploy de MVP** (lo que exista hasta ahora) en un ambiente accesible (puede ser beta.pricewaze.com) para que stakeholders puedan verlo y para empezar pruebas de usabilidad. Configurar monitoreo básico (al menos uptime monitoring, para saber si el sitio cae). Ajustar cualquier issue de configuración que surja (CORS, routing, etc.).

11. *UX:* Realizar tests de usabilidad rápidos con 2-3 usuarios usando el MVP parcial (aunque no tenga IA, ver si pueden buscar propiedades, si la UI les parece intuitiva). Recopilar feedback para ajustes de UX en los flujos básicos (por ejemplo, quizás mejorar cómo se muestran los filtros de búsqueda, o si entienden el botón del copiloto aunque aún no funcione). Documentar hallazgos e incorporar pequeños fixes en este sprint o anotar para siguientes.
    Deliverables Sprint 2: MVP básico desplegado (catálogo de propiedades navegable, accounts funcionales), dataset de demo listo, landing page pública con captación de interesados, primeras publicaciones en redes, reporte breve de feedback UX inicial.

12. **Sprint 3: Integración del Copiloto AI (Versión Alfa)**
    Objetivo: Empezar a incorporar la lógica de IA al sistema. Implementar el backend multi-agente y una versión inicial del copiloto en frontend, con capacidad de generar al menos una recomendación/insight básica.

13. *Desarrollo Backend (AI):* Integrar la biblioteca o framework de **CrewAI** en el proyecto backend. Desarrollar los agentes especializados según el diseño: por ejemplo, programar las clases o funciones para *Market Analyst, Pricing Analyst,* etc., aunque inicialmente pueden retornar respuestas simplificadas o simuladas. Establecer la comunicación con la API de lenguaje (DeepSeek o OpenAI). Probar llamadas unitarias: dado un `property_id` de prueba, obtener que el *Pricing Analyst* calcule fair price (puede usar una fórmula simple por ahora, ej. promedio de zona) y que el *Market Analyst* traiga datos de zona (ej. avg_price_m2 ya almacenado). Implementar el **endpoint** `/api/v1/pricing/analyze` que coordina ambos agentes y devuelve un JSON de análisis [4]. Igualmente, `/api/v1/negotiation/buyer-advice` para oferta sugerida [5], integrando al menos Pricing + Negotiation Advisor. En esta fase alfa, la lógica puede ser simplificada, pero la estructura de llamadas entre agentes y la agregación de resultados via el *Coordinator* deben quedar funcionando. Registrar las llamadas en `pw_ai_logs` para inspección.
14. *Desarrollo Frontend:* Implementar el **panel del Copiloto** real. Conectar el botón flotante de Copiloto para que llame al endpoint `/pricing/analyze` al abrir la vista de una propiedad, reciba el JSON de resultados y renderice una tarjeta de alerta básica. Por ejemplo, si el endpoint devuelve que el precio está 10% sobrevalorado, generar la card ⚠ *"Sobreprecio: ~10% sobre mercado"*. Incluir el link "Ver por qué" que por ahora podría simplemente mostrar un modal con los comparables (puede obtenerlos del backend también). También, incorporar en la vista de propiedad la sección de **historia de precios** llenando datos reales desde la BD (mostrar el fairness_score y bullets que ahora sí provienen de `pw_property_insights` calculados por el backend). Es decir, la UI de la pantalla 2 (Historial de Precio) se conecta con datos reales en este sprint. Para la **negociación asistida**, integrar la llamada al endpoint de negotiation cuando se carga la lista de ofertas de una propiedad: para cada oferta, enviarla si es del usuario actual o contraoferta recibida, y traer un indicador (p. ej., en esta fase podría ser random o simple: si oferta < 90% del precio listar -> rojo, etc., hasta tener la lógica completa). Renderizar los semáforos y textos cortos en la lista de ofertas.

15. *DB:* Poner en práctica las tablas `pw_property_insights` y `pw_alerts`. Ajustar las funciones backend para que al hacer un análisis, inserten o actualicen los registros en estas tablas. Por ejemplo, cuando `analyzePricing` corre, debería upsert en `pw_property_insights` el fairness y narrative resultantes para ese property_id. Para `pw_alerts`, decidir la estrategia: se pueden poblar reactivamente (al consultar el copiloto, calcular alertas en ese momento y devolver) o proactivamente (calculadas antes vía triggers/jobs). En este sprint, quizás más sencillo calcular on-demand: el endpoint de analyze también rellena una lista de alertas lógicas (basadas en reglas definidas [41]) y las devuelve sin guardar. Sin embargo, se puede ya sentar la base de guardar alertas persistentes en BD para cada usuario-propiedad. Implementar triggers sencillos si fácil (ej., cuando se inserta property_insights con underprice_pct < -10%, crear alerta "Oportunidad silenciosa").
16. *Testing y Refinamiento:* Hacer pruebas integrales: elegir varias propiedades de ejemplo, manipular sus datos (poner una claramente cara, otra barata, etc.) y verificar que el copiloto muestra las alertas correspondientes. Medir tiempos de respuesta de la API de análisis – si una llamada a IA tarda demasiado, considerar agregar caching pronto. Recabar al equipo interno probando la alfa del copiloto para ver si las recomendaciones iniciales hacen sentido.

17. *Marketing:* Mientras dev trabaja en la IA, marketing prepara la estrategia de lanzamiento beta. Definir criterios de selección de beta testers (ej. agentes aliados, usuarios que se inscribieron en la lista de espera). Elaborar materiales explicativos: un *pitch deck* breve o video tutorial mostrando cómo usar el copiloto. Iniciar campaña de expectativa en redes: publicar una animación donde se ve una alerta de PriceWaze apareciendo sobre una propiedad (para intrigar a la audiencia sobre la funcionalidad).
    Deliverables Sprint 3: Copiloto AI funcional en entorno de prueba (capaz de generar al menos alertas y un consejo de oferta), backend multi-agente integrado, tablas de insights/alertas poblándose, frontend mostrando datos de IA en panel de copiloto y sección de propiedad. Primer grupo de beta testers identificado, contenido teaser de IA publicado.

18. **Sprint 4: Refinamiento de IA y Funcionalidades Avanzadas**
    Objetivo: Completar la implementación de las 7 alertas lógicas con su plena funcionalidad, afinar las respuestas de la IA (narrativas) y añadir características complementarias (ej. contratos). También, pulir la interfaz según feedback y mejorar performance.

19. *Desarrollo (AI Lógica):* Implementar todas las reglas de alerta según la especificación [41]. Esto implica desarrollar funciones que calculen: absorción de zona (para detectar *sobreprecio emocional*), estacionalidad/mes malo (*timing incorrecto*), detectar subida de demanda (*zona en inflexión*), patrón de oferta ganador (analizar historial de ofertas previas exitosas en BD, si se tiene, para *oferta subóptima*), detectar comparables anómalos (*riesgo oculto*), visibilidad baja + precio bajo (*oportunidad silenciosa*), y condiciones de negociación (ex: oferta sin contingencias vs con muchas cláusulas para *negociación mal planteada*). Algunas de estas pueden requerir más datos; se pueden implementar aproximaciones: ej., *visibilidad baja* si `views_count` de la propiedad < cierto percentil y *precio bajo* si underprice_pct < -X%. Programar estos checks y generar alertas correspondientes almacenando en `pw_alerts`. Puede ser útil crear un **servicio de evaluación** separado (ej: `AlertEvaluator`) que dado un user y property recorra todas las reglas y emita alertas. Llamarlo tanto en eventos (view property) como en un cron.
20. *Desarrollo (AI Narrativa):* Ahora que la lógica está, concentrarse en la calidad de las explicaciones. Ajustar los *prompts* de la IA para formatear bien la salida. Por ejemplo, asegurar que `pw_property_insights.narrative` venga ya en español y con viñetas. Posiblemente usar

algunas pruebas con diferentes modelos o parámetros para ver qué estilo de respuesta es más consistente. También implementar multi-idioma soporte: aunque el mercado inicial es hispanohablante, permitir que si `user.preferred_language` = 'en', las respuestas salgan en inglés (IA lo puede hacer, solo cambiando el prompt). Probar la generación de **contrato borrador** via la IA (endpoint `/contracts/generate` [105]): formatear la respuesta en secciones (partes, precio, fecha de cierre, cláusulas). Asegurar que se guarde en `pricewaze_agreements` la versión generada para registro.

21. *Desarrollo Frontend:* Integrar las funcionalidades avanzadas: módulo de **generación de contrato** (quizá en la página de una oferta aceptada, botón "Generar contrato" que llama al endpoint y luego muestra el contrato en pantalla o descarga PDF). Mejorar la **página de dashboard** para usuarios: por ejemplo, un comprador al entrar a su panel vea un resumen – propiedades vistas recientemente con sus fairness score, alertas pendientes, etc. Esto requiere consultas agregadas, pero la mayoría se puede obtener de BD (por ej., mostrar las últimas 5 alertas no resueltas de `pw_alerts` para ese user). Incorporar la **Exploración Inteligente** (Pantalla 3): permitir al usuario hacer la búsqueda guiada; esto implica un input donde puede escribir en lenguaje natural su deseo, llamando a un nuevo endpoint (posiblemente `/analysis/full` [106] o similar) que use varios agentes para recomendar propiedades. Realizar la lógica backend para este caso: combinar preferencias del usuario + criterios para filtrar propiedades en BD + reordenar por algún "score de conveniencia" calculado por IA. En frontend, mostrar los resultados con la explicación de "Por qué te conviene" debajo de cada card. Esta parte cierra el círculo de funcionalidades IA.
22. *UX/UI Polishing:* Aplicar las mejoras de UI recopiladas: por ejemplo, incorporar los **gradientes de marca** en todos los elementos destacados, como se hizo en Sprint de Branding [59]. Asegurar que el header estilo Zillow responsivo esté implementado [60]. Revisar consistencia de íconos, tamaños de fuente. Incluir animaciones sutiles: por ejemplo, un efecto de *skeleton loading* (ya implementado en componentes UI [107]) mientras se espera respuesta de IA, para que el usuario sepa que algo está cargando. También, añadir *tooltips* o iconos de ayuda junto a conceptos nuevos; ej., un ícono "?" al lado de "Fairness Score" que al pasar diga "Calculado comparando el precio con propiedades similares recientes [108]". Realizar otra ronda de testing con usuarios externos centrado en las nuevas funciones de IA: observar si comprenden las alertas, si confían en ellas, qué dudas surgen. Usar este feedback para afinar textos (ej. renombrar "Fairness" a algo más claro en español como "Equidad de precio").

23. *Marketing:* Preparar el **lanzamiento beta cerrado**. Esto incluye: enviar invitaciones a los beta testers con instrucciones para acceder (enlace, credenciales si se usan cuentas de prueba, etc.). Ofrecer quizás un webinar corto demostrativo para explicarles cómo usar la plataforma. Paralelamente, crear materiales de marketing para público general de cara al lanzamiento público: un video promocional de 1 min mostrando el flujo (antes/sin PriceWaze vs después/con PriceWaze), testimonios de los beta testers iniciales si es posible. Empezar a redactar comunicados de prensa para tech blogs locales anunciando la herramienta, enfatizando la innovación (ej. "Startup dominicana lanza asistente con IA para comprar casas").
    Deliverables Sprint 4: Plataforma casi completa con todas las alertas implementadas y UI refinada, generación de contratos funcional, explorador inteligente activo. Beta cerrada iniciada con ~10-20 usuarios y retroalimentación recogida. Materiales de marketing (video demo, press kit) listos o en borrador avanzado.

24. **Sprint 5: Pruebas Beta y Ajustes Finales**
    Objetivo: Someter PriceWaze a pruebas de fuego con usuarios reales, corregir bugs, mejorar

rendimiento donde sea necesario y preparar la infraestructura para escala. De igual forma, ultimar detalles de marketing para el lanzamiento público.

25. *Atención a Beta:* Durante este sprint, se monitorea intensivamente el uso por beta testers. Se recolectan métricas: cuántos análisis ejecutan, si hay errores en las llamadas de IA (ver logs), qué tan rápido responde todo. Se habilita un canal de comunicación (grupo de WhatsApp o Discord) con los beta testers para recibir sus impresiones en tiempo real. El equipo de desarrollo debe estar listo para **parchear bugs críticos rápidamente**. Ej: si se descubre un fallo en cierta alerta que siempre sale mal, se corrige y despliega una actualización menor a producción beta. También se evalúa la carga: si 20 usuarios simultáneos provocan lentitud, se analiza y optimiza (quizá aumentar tier de supabase, o bajar temperatura del modelo IA si tarda mucho, etc.).
26. *Optimización y Performance:* Con datos reales de uso, identificar consultas lentas (usar el dashboard de Supabase to check query performance). Si la búsqueda geoespacial está lenta, tal vez implementar *materialized view* o ajustar índices. Optimizar el frontend bundling: quitar dependencias innecesarias, hacer code splitting para reducir el tamaño inicial de carga. Verificar memoria y CPU del backend IA: afinar parámetros de la aplicación server (por ejemplo, aumentar gunicorn workers si using, etc.). Simular escenarios de carga más altos con herramientas (JMeter, k6) para 100 usuarios concurrentes, e identificar posibles cuellos de botella.
27. *Seguridad & Audit:* Realizar un **pentesting ligero** o auditoría de seguridad. Revisar que no haya endpoints expuestos sin auth, que las políticas RLS estén correctamente aplicadas (testear que un usuario A no pueda acceder a los datos de B vía supabase queries). Comprobar que los formularios de entrada (login, inputs de chat) saniticen datos para prevenir XSS o inyecciones. Si es posible, involucrar un tercero confiable para hacer una prueba de penetración breve. Implementar correcciones de cualquier vulnerabilidad descubierta.
28. *UX Improvements:* Procesar el feedback cualitativo de la beta. Quizá se descubre que ciertos términos no se entienden, entonces se ajustan. Por ejemplo, si “Oportunidad silenciosa” no se entiende, se cambia a “Oportunidad oculta (precio bajo, pocos lo han notado)”. Agregar en la UI alguna indicación de que **los agentes inmobiliarios humanos siguen siendo importantes** (para alinear expectativas); quizá un texto en el perfil: “Recuerda: PriceWaze te asesora con datos, pero contar con tu agente de confianza siempre suma.” Esto, según feedback, puede ayudar a que agentes no lo vean como amenaza sino como aliado.

29. *Marketing Pre-lanzamiento:* Basado en la beta, recopilar testimonios o casos de éxito: ej, un tester pudo negociar 10% abajo gracias a PriceWaze – con su permiso, usar esa historia en marketing. Crear contenido educativo: un whitepaper o infografía “Los 7 alertas de PriceWaze explicados” para distribuir en redes/blog, que sirve tanto para marketing como para que futuros usuarios entiendan el trasfondo. Afinar el sitio web público para el lanzamiento: actualizar la landing con nuevas screenshots reales del producto en acción, incorporar un botón “Prueba PriceWaze ahora” que estará activo al lanzar. Planificar la campaña de lanzamiento: fecha y hora, qué se publicará (hilo de Twitter mostrando funciones, post de LinkedIn para comunidad profesional, etc.), a quién se contactará (prensa, influencers de bienes raíces).
    Deliverables Sprint 5: Plataforma robustecida tras pruebas, informe de feedback de beta con acciones tomadas, sistema optimizado y seguro para escalamiento inicial. Plan de lanzamiento detallado y materiales finales (historias de usuarios, blog posts) listos. Go/No-Go meeting con equipo para validar que estamos listos para lanzar públicamente.

30. **Sprint 6: Lanzamiento Público y Monitoreo Inicial**
    Objetivo: Desplegar PriceWaze al público general, monitorear su desempeño en vivo con una base creciente de usuarios y establecer un ciclo de mejora post-lanzamiento.

31. *Lanzamiento:* En el día y hora acordados, hacer público el acceso a PriceWaze (quitar restricciones de beta, enviar mails a lista de espera, publicar en redes sociales el anuncio oficial con link). Mantener el equipo en "war room" ese día para responder rápidamente a cualquier incidencia inesperada. Verificar en tiempo real el estado de los servicios durante el peak inicial de curiosos.
32. *Monitoreo en producción:* Usar las herramientas de monitoreo configuradas para observar métricas. Particular foco en: tasa de error de API (debe mantenerse muy baja, <1%), latencia media de respuesta del copiloto (aspirar a que <2s), uso de CPU/DB (asegurarse que no se esté saturando). También monitorear engagement: cuántos registros nuevos por hora, cuántas propiedades vistas, etc., para comparar con las previsiones de carga.
33. *Soporte al Usuario:* Habilitar canales de soporte público (email support@pricewaze o chat integrado) y estar listos para responder dudas. Crear un pequeño **FAQ** en la web a partir de las preguntas que vayan surgiendo recurrentemente durante los primeros días. Si algún bug menor aparece con usuarios reales (siempre pasa algo imprevisto), priorizar fixes rápidos con parches desplegados vía CI/CD.
34. *Escalado progresivo:* Si la demanda resulta mayor a la anticipada (lo cual sería bueno), estar preparados para escalar. Por ejemplo, aumentar el plan de Supabase a más recursos o habilitar la réplica de lectura, subir un nuevo contenedor de backend IA si la CPU llega al 80% sostenido, etc. Gracias al uso de infraestructura flexible, esto se puede hacer en caliente con mínimos cortes.
35. *Feedback e Iteración:* Reunir al equipo a las 2 semanas del lanzamiento para una **retrospectiva de lanzamiento**. Evaluar qué salió bien y qué no. Revisar analíticas: ¿Los usuarios están usando el copiloto? ¿Qué porcentaje de propiedades vistas activan alertas? ¿Hay funciones casi no usadas? Por ejemplo, si notamos que pocos entran a "Exploración inteligente", investigar por qué (quizá no está visible, o los usuarios no entienden que pueden usar lenguaje natural). Priorizar en el backlog las mejoras o features de segunda fase que aporten más valor ahora que hay usuarios reales (quizá un app móvil, o una función para vendedores).
36. *Marketing continuo:* Después del boom inicial, mantener el ritmo de marketing. Publicar casos de uso interesantes que aparezcan. Contactar medios para que cubran la historia (si no se hizo ya): enfatizar la diferenciación (primera IA inmobiliaria en RD). Utilizar el contenido creado (whitepaper, etc.) como lead magnet para atraer más registros. Evaluar métricas de adquisición: qué canal trajo más usuarios (orgánico, referidos, ads si se corrieron). Doblar esfuerzos en lo que funcione y ajustar lo que no.
    Deliverables Sprint 6: PriceWaze lanzado públicamente con éxito, X número de usuarios registrados en primeras dos semanas, monitoreo operativo funcionando sin incidentes mayores. Informe post-mortem del lanzamiento con métricas iniciales y plan de mejoras continuo.

Auditoría Final: A lo largo de estos sprints, se cubrieron todos los aspectos clave: - El **Dev Dept** entregó arquitectura sólida, código de calidad con test, IA integrada eficientemente. - **Marketing** definió la propuesta de valor, creó expectativa y acompañó con contenido educativo, asegurando que el mercado entienda el producto. - **Producción/Logística** implementó la infraestructura y procesos para un servicio confiable, con escalabilidad probada y datos actualizados constantemente (jobs, triggers). - **UI/UX** fue iterada con feedback real, logrando una interfaz intuitiva y una experiencia de usuario que inspira confianza y satisfacción. - **DB** fue bien diseñada, soportando todas las features con integridad y performance. - **Best**

practices se siguieron en cada etapa, resultando en un producto técnicamente robusto y éticamente responsable.

Cada sprint fue auditado contra sus objetivos y ajustado según la retroalimentación, lo que demuestra un proceso de mejora continua. El plan presentado es integral y deja a cada área *"complacida"*: Desarrollo tiene un camino claro, Marketing obtiene un producto vendible y diferenciable, Operaciones cuenta con fiabilidad, Diseño ve plasmadas sus ideas en una UX de alta calidad, y la dirección del proyecto puede ver un panorama paso a paso hasta la meta final. Con esta hoja de ruta, PriceWaze está encaminado a un **lanzamiento exitoso y sostenible**, listo para revolucionar el mercado inmobiliario con inteligencia artificial y una ejecución impecable. [108] [34]

---

[1] [2] [3] [4] [5] [9] [10] [67] [101] [105] [106] README.md
https://github.com/nadalpiantini/pricewaze/blob/ad234038f93140487ff73f7a1f8d532d425ba80e/crewai/README.md

[6] [7] [8] [13] [14] [15] [16] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [65] [66] [68] [91] [92] [93] [94] [95] [96] [97] [103] [108] PRICEWAZE_COPILOT_V1_DEFINITIVO.md
https://github.com/nadalpiantini/pricewaze/blob/ad234038f93140487ff73f7a1f8d532d425ba80e/docs/PRICEWAZE_COPILOT_V1_DEFINITIVO.md

[11] [12] [36] [37] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [98] [99] 20260106000001_pricewaze_initial_schema.sql
https://github.com/nadalpiantini/pricewaze/blob/ad234038f93140487ff73f7a1f8d532d425ba80e/supabase/migrations/20260106000001_pricewaze_initial_schema.sql

[17] Ai In Real Estate Market Report Opportunities 2025
https://www.thebusinessresearchcompany.com/report/ai-in-real-estate-global-market-report

[18] [19] [33] Most Americans Use AI for Housing Market Info | Florida Realtors
https://www.floridarealtors.org/news-media/news-articles/2025/10/most-americans-use-ai-housing-market-info

[20] [21] [22] [23] [34] A Zillow-ChatGPT integration; Elliman launches AI assistant
https://www.realestatenews.com/2025/10/06/a-zillow-chatgpt-integration-final-offer-enters-new-market

[24] [25] [26] [27] Dominican Republic: The Market Startups Keep Underestimating | by Ana Abreu | Nov, 2025 | Medium
https://anaabreur.medium.com/dominican-republic-the-laboratory-startups-are-underestimating-038a7b11ac49

[28] Dominican Republic's Residential Property Market Analysis 2025
https://www.globalpropertyguide.com/latin-america/dominican-republic/price-history

[29] 6 Go-To House Pricing Strategies Used to Sell Real Estate in 2026
https://www.homelight.com/blog/house-pricing-strategies/

[30] The Psychology of Buyer & Seller Decision-Making: What New ...
https://realtyschool.com/the-psychology-of-buyer-seller-decision-making-what-new-agents-need-to-know/

[31] [32] Real Estate Pricing Psychology, Explained | Redfin
https://www.redfin.com/blog/real-estate-pricing-psychology/

[35] [38] [39] [59] [60] [61] [62] [63] [64] [100] [102] [104] [107] SPRINT_LOG.md
https://github.com/nadalpiantini/pricewaze/blob/ad234038f93140487ff73f7a1f8d532d425ba80e/SPRINT_LOG.md