ChatGPT

Especificación de la Plataforma de Agentes (Portal Inmobiliario)

1. Executive Summary

La **Plataforma de Agentes** será un portal web integrado en nuestro marketplace inmobiliario, destinado a que agentes y brokers gestionen leads, listings y su negocio diario, con una experiencia al nivel de Zillow Premier Agent. El MVP se enfocará en los módulos de uso diario de más alto impacto en retención: **Inbox/Chat unificado**, gestión de **Leads** (pipeline Kanban con estados, notas y tareas), **Calendario de citas/visitas**, y **Administración de Listings** (estado, verificación, actividad). Estos componentes replicarán las funciones esenciales confirmadas en Zillow (p.ej. inbox con mensajes y detalles del lead, filtrado por estado, tareas de seguimiento [1] [2]) y añadirán **micro-interacciones** y feedback inmediato para fomentar hábito de uso.

En la primera release (MVP) se igualarán las capacidades básicas de Zillow: recepción de leads con detalles contextuales, mensajería integrada, cambio de estado de leads y agenda de visitas con recordatorios [3] [1]. Esto sentará las bases para **retención diaria**: los agentes volverán varias veces al día a revisar nuevos leads, responder mensajes y actualizar su pipeline. Subsecuentes releases (V1, V2) desbloquearán monetización y funciones avanzadas: compra de **créditos** y promoción de listings, herramientas de **reporting & analytics** (incluyendo un "Agent Score" similar a *Best of Zillow* [4]), gestión de **equipos** con ruteo de leads y control de desempeño, y servicios premium (p.ej. solicitar fotografía profesional). Cada iteración añadirá nuevos **"triggers" de retorno** (notificaciones, recomendaciones) sin recurrir a dark patterns, sino mediante valor tangible (por ejemplo, alertas de nuevos leads, sugerencias de seguimiento y logros por completar tareas).

Prioridad MVP: Funciones que maximizan el retorno inmediato – Inbox/Chat, Pipeline de Leads con tareas, Calendario de visitas y Listings con actividad – ya que cubren el 80% del flujo diario de un agente. Esto garantiza paridad con Zillow en lo esencial y sienta una base extensible. **Release V1** añadirá Créditos/Billing in-app y Reporting básico (mostrando métricas de respuesta y conversión) para comenzar a monetizar y brindar sensación de progreso. **Release V2** integrará Team management y analytics avanzados, ampliando la plataforma para brokers con equipos y ofreciendo diferenciales de valor (ej. alertas proactivas por IA, integraciones dotloop/CRM externo). En conjunto, la plataforma evolucionará de un CRM reactivo a un **hub proactivo** que constantemente invita al agente a mejorar su desempeño y aprovechar oportunidades, con loops de hábito positivos (tareas claras, recompensas visuales, mejora de métricas personales). Tras el MVP, cada release incrementa retención y LTV: V1 habilita ingresos directos por créditos/upsells, V2 fortalece uso en equipos y agentes top. La especificación a continuación detalla todos estos módulos, interacciones de UX/motion, modelos de datos, librerías OSS seleccionadas y un plan de implementación incremental para lograrlo.

2. Feature Map (Módulos y Funcionalidades)

Módulo	Funcionalidad	Prioridad	Trigger de retorno	Valor para el agente	Complejidad	Dependencias
Inbox & Leads	Inbox unificado (leads + mensajes)	MVP	Sí – *Nuevo lead/mensaje*	Un solo lugar para ver contactos nuevos, mensajes y notificaciones [1] [5].	Media	API leads/ mensajes realtime
Inbox & Leads	Búsqueda y filtros de leads/ mensajes	MVP	No (utilidad continua)	Permite encontrar contactos por nombre, estado o palabra clave [7]; segmenta pipeline por prioridad.	Baja	-
Inbox & Leads	Cambio de estado pipeline (Kanban)	MVP	No (pero mejora retorno indirecto)	Visualiza progreso de cada lead; mover leads entre etapas con drag&drop y feedback inmediato.	Media	Librería DnD (p.ej. DndKit)
Inbox & Leads	Notas internas y etiquetas de lead	V1	No	Documentar info útil (ej. “busca 3H, tiene crédito pre-aprobado”); clasificar leads (frío, caliente).	Baja	-
Inbox & Leads	Tareas y recordatorios follow-up	V1	Sí – *Lead sin respuesta 2h*, *follow-up diario*	Organiza to-do específicos por lead (llamar, enviar email) [9]; recuerda hacer seguimiento a tiempo.	Media	Calendario/ agenda

Módulo	Funcionalidad	Prioridad	Trigger de retorno	Valor para el agente	Complejidad	Dependencias
Chat (Inbox)	Mensajería en tiempo real con leads	MVP	Sí – *Nuevo mensaje*	Comunicación centralizada estilo chat [1]; envía/recibe textos, fotos. Mantiene al agente como punto de contacto único [11].	Media	WebSocket/ infra realtime
Chat (Inbox)	Plantillas rápidas y adjuntos básicos	V1	No (valor en eficiencia)	Respuestas predefinidas (saludo, solicitud de cita) para agilizar; posibilidad de adjuntar PDF/ imagen (brochure).	Media	-
Calendar/ Citas	Calendario integr. (agenda visitas)	MVP	Sí – *Cita confirmada/ hoy*	Visualiza citas agendadas con leads; evita olvidos con recordatorios automáticos. Sincroniza disponibilidad del agente.	Alta	Librería calendario (RBC)
Calendar/ Citas	Confirmar/ Reprogramar/ Cancelar visitas	MVP	Sí – *Cita reprogramada*	Permite gestionar cambios en citas (cliente solicita cambiar hora, etc.) con notificación a ambos lados.	Media	-
Listings	Gestión de listings del agente	MVP	No (pero ligado a actividad)	Editar precio, descripción, fotos; actualizar estado (activo, reservado, vendido). Agente ve todos sus listings en un lugar.	Media	API listings marketplace

Módulo	Funcionalidad	Prioridad	Trigger de retorno	Valor para el agente	Complejidad	Dependencias
Listings	Solicitar verificación de listing	V1	Sí – *Listing verificado*	Envío de documentación para verificar propiedad (ej. título, ID propietario) para obtener sello de verificado (aumenta confianza).	Baja	Workflow verif. (backoffice)
Listings	Feed de actividad del listing	MVP	Sí – *Actividad destacada*	Muestra eventos: nuevas views, saves, consultas recibidas, cambios de precio [13]. Mantiene informado al agente del interés generado.	Media	Tracking front (views/saves)
Créditos/ Billing	Saldo de créditos y recarga (UI)	V1	Sí – *Saldo bajo*	Transparencia en monetización: el agente ve cuántos créditos le quedan para leads/promos, y puede comprar más fácilmente.	Media	Pasarela pago API
Créditos/ Billing	Historial de consumo (ledger)	V1	No (consulta eventual)	Tabla de movimientos de créditos: leads comprados, cargos por destacar listings, etc., con fechas y descripciones.	Baja	-
Team	Gestión de miembros (roles, invitaciones)	V2	No (valor para brokers)	Brokers pueden invitar agentes, asignar roles (Admin, Agente), ver todos en su equipo. Controlan accesos.	Media	API equipos/ roles

Módulo	Funcionalidad	Prioridad	Trigger de retorno	Valor para el agente	Complejidad	Dependencias
Team	Ruteo de leads (reglas por zona/precio)	V2	Sí – *Lead reasignado*	Asigna leads automáticamente al agente apropiado según criterios [14]; garantiza respuesta rápida incluso con equipo grande.	Alta	Motor reglas backend
Team	Pausar agentes (no recibir leads)	V2	No (solo evita triggers)	Permite marcar un agente “de vacaciones” para que no reciba leads temporalmente [14]. Evita leads sin atender.	Baja	-
Reporting	Panel de rendimiento (lead funnel)	V1	No (insight para hábito)	Muestra métricas clave: volumen de leads por periodo, breakdown por tipo/Zona, tasa de respuesta y conversión a cita [15] [16]. Motiva a mejorar con datos concretos.	Media	- (datos de nuestra DB)
Reporting	Customer Experience (feedback)	V2	Sí – *Nuevo feedback*	Similar a “Best of Zillow”: recoge feedback de clientes en distintas etapas [17] [18] y muestra al agente su score de satisfacción. Ayuda a mejorar servicio.	Alta	Integrar encuestas (email/app)

Módulo	Funcionalidad	Prioridad	Trigger de retorno	Valor para el agente	Complejidad	Dependencias
Notificaciones	Centro de notifs in-app	MVP	Sí – *Badge notifs*	Panel centralizado tipo campana, lista eventos recientes (leads, mensajes, sistema). Facilita que el agente no se pierda nada importante.	Media	Infra notifs + WS
Notificaciones	Preferencias & Quiet hours	V1	No (control usuario)	Agente configura qué notifs recibir (email, push) y define horas sin disturbios [19] [20]. Evita frustración por spam o fuera de horario.	Baja	-
Otros	Perfil del Agente (público y privado)	V2	Sí – *Perfil completo*	Página de perfil con info pública (para clientes) y progreso de completitud (guía al 100%). Incluir foto, bio, especialidades, reviews.	Baja	-
Otros	Integraciones externas (dotloop, etc.)	V2	No (feature soporte)	Posibilidad de integrar herramientas: ej. conectar dotloop para iniciar transacciones desde CRM [21], o ShowingTime para calendario. Ahorra tiempo al agente con todo en un lugar.	Alta	APIs externas (OAuth)

Nota: Prioridad *MVP* indica imprescindible en la primera versión. *V1* (~próxima iteración) y *V2* (futuro) indican mejoras graduales. "Trigger de retorno" se refiere a eventos que generarían notificaciones o motivos para que el agente vuelva; *Sí* va acompañado de un ejemplo de evento específico. Complejidad estimada considera solo frontend (asumiendo backends disponibles). UI/Motion notes resume consideraciones de interfaz y micro-interacciones planeadas. Métricas asociadas sirven para medir el éxito de cada funcionalidad (adopción, eficacia o impacto en negocio).

3. Benchmark Zillow: Funcionalidades y Modelo CRM

Zillow Premier Agent (app/CRM) es la referencia principal. A continuación se comparan módulos equivalentes, indicando qué está **confirmado** por fuentes públicas de Zillow y qué son **hipótesis** nuestras (a validar):

- **Inbox (Bandeja de entrada unificada):** *Confirmado:* Zillow tiene un inbox central donde llegan **todos los nuevos contactos, conexiones y mensajes**, listados cronológicamente [1] [5]. Cada lead entra marcado con un icono indicando fuente o siguiente paso óptimo [5]. El inbox permite **buscar y filtrar** por nombre, tipo de lead, estado *My Agent* (exclusividad) y más [7]. En desktop, el inbox se presenta en **tres paneles**: lista de leads/mensajes a la izquierda, chat o detalles de mensaje en el centro, y detalles del lead a la derecha [6]. *Hipótesis:* Zillow probablemente resalta visualmente los leads nuevos/no leídos (bold o indicador). Asumimos que una entrada de inbox corresponde a un canal de comunicación (ej. si un mismo contacto envía email y SMS, aparecen dos entradas) [22] – esto es inusual, tal vez por separar hilos. Nuestro diseño podría simplificar esto unificando por contacto, pero mantendremos separado si diferentes canales requieren atenciones distintas. Validaremos si unificar threads por contacto es viable sin perder contexto del canal.

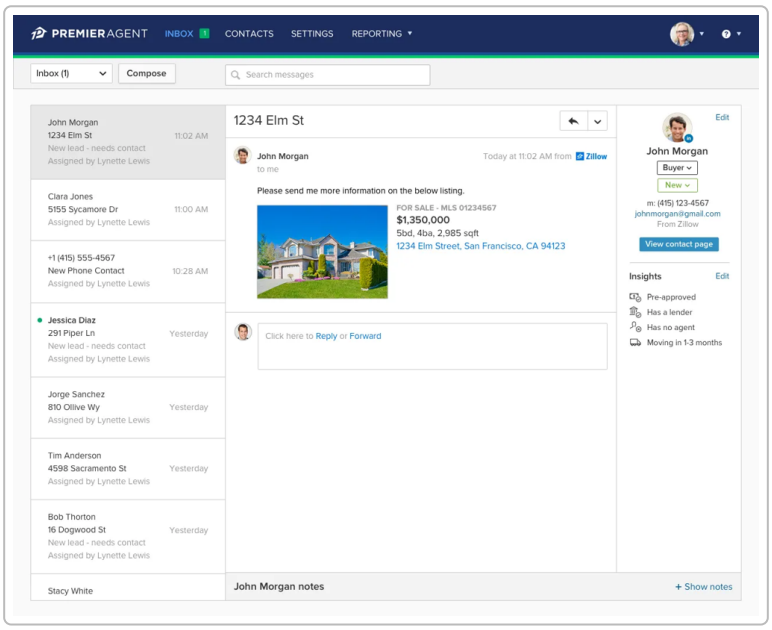

Imagen: Inbox de Zillow Premier Agent en desktop, con lista de leads (izq), mensajes y propiedad (centro), y detalles del lead (der.) [6]

- **Contacts/Leads (CRM de contactos):** *Confirmado:* Zillow ofrece una vista de **lista de contactos** con opciones de filtrado robustas para gestionar el pipeline [23] [8]. Se pueden filtrar por *Time frame* (urgencia de mudanza), estado de búsqueda, y por relación *My Agent* activa, entre otros [24]. Los leads entran con estado inicial "New" automáticamente [25]. Tienen atributos como status de pre-aprobación y ventana de tiempo para moverse [26]. No se menciona explícitamente un **Kanban**, pero Zillow sí enfatiza pipeline y seguimiento; su mobile app muestra pipeline en lista segmentada más que tablero. *Hipótesis:* Implementaremos un **Pipeline Kanban** para visualizar etapas (Nuevo, En contacto, Visita programada, etc.) porque muchos CRM lo usan para arrastrar leads entre etapas de venta. No hay evidencia directa de tablero en Zillow, pero la funcionalidad de cambiar status está confirmada (y Zillow incluso automatiza recordatorios según status) [27]. Validaremos con usuarios agentes si prefieren vista Kanban o lista filtrada;

nuestra MVP incluirá arrastrar para cambiar estado con undo (esto no lo vimos en docs de Zillow, sería una mejora). También asumimos se pueden **asignar etiquetas** o prioridades a leads – Zillow no lo menciona, pero agregarlas podría ser valor añadido (lo trataremos como hipótesis a validar en uso real).

- **Lead Details & Insights:** *Confirmado:* Al abrir un lead, Zillow muestra en desktop un panel de detalles con toda la información del contacto (número, email, requisitos) y campos clave como **pre-aprobación hipotecaria y plazo para moverse** [26]. Además, si el lead viene por un listing específico, hay un panel con los detalles de esa propiedad (fotos, mapa, precio) y botón para contactar al listing agent [28]. Zillow provee **recomendaciones de seguimiento** en el panel de mensajes del lead [29] (posiblemente sugerencias tipo "Responde rápido con…", aunque no se detalla). *Hipótesis:* Es probable que Zillow ofrezca botones rápidos para llamar, enviar email o texto desde la vista del lead (dado que menciona que se puede responder por texto, email o llamada directamente [30]). Supondremos esos accesos directos son parte del diseño (los incorporaremos en nuestro Inbox/Lead panel). También Zillow destaca la sección *Insights* en mobile: un feed en tiempo real de cambios en el comportamiento del comprador (ej. "tu lead X incrementó su actividad de búsqueda", "empezó a ver otras zonas") [31]. Esto es **confirmado** en mobile [32] y en desktop posiblemente integrado en la ficha del lead. Nuestra plataforma tomará esa idea: en la ficha del lead habrá un pequeño feed "Actividad del cliente" mostrando, por ejemplo, propiedades vistas/salvadas en nuestro portal (si consentido), para que el agente entienda mejor sus intereses [33]. Como hipótesis avanzada, consideraríamos utilizar esos datos para **recomendaciones automáticas** (Zillow menciona que su app puede usar IA para mandar recomendaciones de casas vía texto [34] – feature interesante pero fuera de MVP; lo anotamos para explorar en futuro).

- **Tasks & Reminders (Tareas):** *Confirmado:* Zillow CRM tiene una sección de **Tasks** donde el agente puede crear tareas con tipo, descripción, contacto asociado, fecha de vencimiento y alerta [35]. En móvil se segmentan en pendientes y completadas, en desktop se conmutan vía dropdown [36]. Marcar una tarea como completa es fácil (checkbox) [10]. Además, para *Team leads*, existen **Team Reminders** automáticos que el líder configura para recordar a sus agentes que actualicen leads estancados [27] [37]. Por ejemplo: "Si un lead lleva >30 min en estado New, recordar al agente que lo atienda" [27], o "si un lead con time-frame inmediato lleva X días sin contacto, pedir nota" [27]. Estos recordatorios se envían como push notifs cada 24h hasta completarse [38]. *Hipótesis:* Implementaremos tareas manuales en MVP (similar a Zillow) y quizás un simple reminder personal ("seguir up mañana") por lead. Los **recordatorios automáticos de equipo** los planificamos para V2, sujetos a tener gestión de equipos. Validaremos si nuestra base de agentes (muchos individuales o equipos pequeños) requiere esa automatización desde temprano. En MVP podríamos incluir un "nudge" si un lead nuevo no cambia de estado tras X horas (in-app), sin configuraciones complejas. Es un plus no mencionado públicamente de Zillow salvo en contexto de equipos, pero creemos valioso incluso para usuarios individuales (hipótesis: los agentes valoran que el sistema les recuerde leads olvidados; cuidado de no ser molesto – daremos control en settings de notificaciones).

- **Insights / Performance (Reportes):** *Confirmado:* Zillow ofrece reportes de desempeño dentro de **"Performance Reports"**, accesibles en desktop desde menú Reporting [39]. Incluye varios reportes:

- **Lead Report:** Análisis de volumen de leads por periodo, breakdown por tipo de lead, ZIP y rango de precio [40], y muestra la **answer rate** (tasa de respuesta) del agente, que Zillow usa para garantizar distribución equitativa de leads [41].

- **Customer Experience Report:** recopila **feedback real de los clientes (connections)** y muestra con qué frecuencia conviertes conexiones en citas presenciales y clientes finales [42]. Este es el sistema *Best of Zillow*, con métricas CSAT y Work-With Rate: Zillow envía encuestas 24h, 15 días y 45 días después de conectar con el lead para medir satisfacción y si planean seguir contigo [43] [44]. Agentes con alta puntuación son reconocidos como "Best of Zillow".
- **ROI Report:** relaciona gastos en Premier Agent con ventas cerradas, permitiendo ver ROI global y por ZIP (dónde conviene invertir más) [45].
- **Team Report:** para líderes de equipo, mostrando pipeline consolidado y desglose de leads por agente y estado, incluyendo tasas de respuesta de cada miembro [46].

Hipótesis: Nuestro MVP incluirá un **dashboard básico** con métricas clave (leads recibidos en mes, tasa de respuesta, conversión a visita), pero no tendremos encuestas automatizadas ni ROI complejo al inicio. Tomamos inspiración del Customer Experience Report de Zillow para nuestro concepto de **"Agent Health Score"**, pero inicialmente lo basaremos en métricas internas (p.ej. rapidez de respuesta, % leads convertidos a visitas, perfil completo) en vez de encuestas externas, para evitar complejidad operacional. Más adelante (V2) podríamos añadir un módulo de feedback del cliente (quizá enviando encuestas NPS sencillas tras cierto tiempo) – eso replicaría la idea de Best of Zillow pero requeriría volumen y controlar spam, así que es una hipótesis a validar cuando tengamos suficientes leads generados por el sistema. En monetización, no tendremos un ROI report real sin una plataforma de ads robusta; en lugar de eso mostraremos quizás "Clientes cerrados vs Inversión en créditos" manualmente. Validaremos con usuarios si valoran ver ROI de sus gastos en la plataforma.

- **Listing Management:** *Confirmado:* Zillow permite a los agentes (que también son listing agents) **ver y editar sus listings** publicados en Zillow/Trulia [47] (en mobile esa opción está bajo Settings). Pueden actualizar detalles y añadir ventas cerradas recientes a su perfil [48]. También Zillow muestra **stats de cada listing**: views, saves, etc. (en Premier Agent no se detalla en texto, pero Zillow ofrece al propietario del listing esas stats; suponemos el agente también las ve). *Hipótesis:* Implementaremos un módulo "Listings" donde el agente ve todas sus propiedades listadas en nuestra plataforma, con indicadores de visitas, favoritos, y mensajes recibidos sobre ese listing (un mini-dashboard por listing). Zillow no tiene "actividad del listing" visible en Premier Agent (no mencionado), pero creemos que **sí** la tiene: de hecho, Zillow envía un email semanal a propietarios/agentes con estadísticas de views. Supondremos que en el portal hay un feed de actividad (Zillow en su blog general menciona un "owner dashboard"). Incluiremos en cada ficha de listing un feed de eventos (ej. "20 usuarios vieron tu propiedad hoy", "3 usuarios la guardaron esta semana"), para dar motivos de volver. Además, añadiremos **solicitud de verificación**: Zillow marca agentes verificados, pero también listings verificados (en algunos mercados) – no vimos referencias directas excepto dotloop auto-actualizando ventas y solicitando reviews [49]. Dado que nuestra plataforma planea un modelo freemium con verificación de listings, lo incluiremos (hipótesis basada en necesidades locales de confianza).

- **Team Management:** *Confirmado:* Zillow Premier Agent para equipos ofrece herramientas de **lead routing avanzado**: reglas según tipo de lead, rango de precio, ubicación para asignar automáticamente al agente disponible correcto [50]. Permite también **pausar** agentes (no recibir leads temporalmente) [14], y como vimos, config. de recordatorios de equipo. Los líderes pueden ver el desempeño de cada miembro (answer rates, volumen) [46]. *Hipótesis:* Estas funciones avanzadas las planificamos para V2. Para MVP, asumimos la mayoría de nuestros usuarios iniciales serán agentes individuales o pequeños equipos, así que el enfoque estará en la experiencia de un solo agente. Sin embargo, diseñaremos la arquitectura (rutas, data model) con la noción de "Agency/Team" desde ya, para no hacer refactor masivo luego. Probaremos con un par de agencias piloto antes de lanzar Team features completas. También es posible Zillow permite *compartir leads* o reasignarlos manualmente; lo consideraremos (un líder podría

reasignar lead A de Agente1 a Agente2). Lo marcaremos como hipótesis hasta confirmar la necesidad.

- **Reviews & Ratings:** *Confirmado:* Zillow incentiva a pedir **reviews al cliente**: tras cerrar transacciones (sobre todo vía dotloop), se puede solicitar al cliente dejar una reseña en el perfil público del agente [49]. Las reseñas públicas son importantes en Zillow (ayudan a generar nuevos leads orgánicos). También internamente, las encuestas *Best of Zillow* son privadas para mejoras. *Hipótesis:* Nuestro portal permitirá al agente ver sus reseñas públicas (moderar si la plataforma lo permite) y su calificación promedio. No implementaremos desde inicio la recolección activa de reseñas (posiblemente enviarlas manualmente por agente). En reporting quizá mostraremos "Nº de reviews en tu perfil" para incentivar a solicitarlas. Dado que las reviews en Zillow son un factor de ranking en su directorio, supondremos que en el nuestro también lo serán, pero eso es backend/algoritmo externo al portal en sí.

- **Notificaciones (multi-canal):** *Confirmado:* Zillow permite configurar notificaciones por **email, SMS y push móvil** para nuevos leads [51]. El agente puede elegir horarios para SMS (quiet hours) [51] [52]. En Premier Agent móvil hay push inmediatos para leads y recordatorios [53]. *Hipótesis:* Nuestra plataforma web integrará un **centro de notificaciones in-app** para real-time (similar a una app web), y enviaremos emails para eventos críticos (lead nuevo, mensaje nuevo), con opción de opt-out. SMS/push requerirían app móvil nativa o PWA push – lo posponemos hasta tener app nativa. Implementaremos **"Quiet hours"** en settings para no enviar notifs nocturnas, asumiendo los agentes aprecian eso (Zillow lo tiene). Validaremos en pruebas si los agentes quieren recibir notifs 24/7 o limitar. Seguir la transparencia de Zillow aquí nos parece clave para confianza del usuario.

En síntesis, **Zillow Premier Agent** nos sirve de blueprint en funcionalidades y mejores prácticas de CRM inmobiliario: pipeline completo de lead a cierre, datos compartidos (insights de navegación del comprador), énfasis en rapidez de respuesta y satisfacción. Nuestra plataforma buscará **superarla** en interactividad (más feedback visual, más personalización local) y en integrar loops de hábito saludables (gamificación ligera tipo tareas/logros). Donde Zillow cuenta con recursos (p.ej. encuestas NPS automáticas, IA para recomendaciones), nosotros iteraremos hacia eso con prudencia: primero datos básicos y triggers manuales, luego automatización guiada por AI una vez haya suficientes datos usuarios (hipótesis a validar con el crecimiento). Muchas funcionalidades "V2" listadas son aspiracionales tomando como guía el ecosistema Zillow; priorizaremos lo confirmado y mediremos qué innovaciones propias aportan valor real antes de invertir en ellas.

4. Arquitectura de Información y Navegación

Sitemap y Rutas Principales: La plataforma de agentes residirá bajo la URL base `*/agents/*` dentro de nuestra SPA. La jerarquía de páginas propuesta es:

- `/agents` – **Overview/Dashboard** general (resumen de métricas del día, tareas, highlights de actividad). Inicio por defecto tras login de agente.
- `/agents/inbox` – **Inbox/Conversaciones**, listado de leads y mensajes (similar a bandeja de entrada).
- `/agents/leads` – **Leads/Contacts**, vista tipo CRM de todos los leads con filtros avanzados (lista completa, posiblemente redundante con Inbox pero con más campos tipo Excel).
- `/agents/calendar` – **Calendario**, agenda de citas/visitas programadas (vista semanal/mensual).
- `/agents/listings` – **Mis Listings**, listado de propiedades del agente con estado y acciones.

- `/agents/credits` – **Créditos & Facturación**, muestra saldo, opciones de compra y historial de uso.
- `/agents/team` – **Equipo**, gestión de miembros (solo visible si el usuario es líder de equipo o broker con sub-agentes).
- `/agents/reports` – **Reportes/Insights**, panel de métricas de desempeño, funnel de ventas, etc.
- `/agents/settings` – **Configuración**, incluye perfil de agente, preferencias de notificación, integraciones conectadas, etc.

Además, existen sub-rutas o páginas de detalle emergentes: - `/agents/lead/:id` – Detalle de un lead específico (podría mostrarse como modal o panel en /inbox, o en página separada según diseño responsive). - `/agents/listings/new` – Formulario para crear nuevo listing (si se permite). - `/agents/appointment/:id` – Detalle de cita (ej. confirmar asistencia, reprogramar). - Sub-secciones de Settings: `/agents/settings/profile`, `/agents/settings/notifications`, `/agents/settings/team` (o integrado con team route), etc., según convenga organizar.

Todas estas rutas estarán protegidas (solo accesibles para usuarios con rol agente). Usaremos **React Router** con un layout padre para `/agents/*` que renderiza el marco general (sidebar, header) común, y luego cada subruta en el área de contenido.

Layout y Navegación UI: Adoptaremos una estructura clásica de aplicación de productividad: - **Sidebar lateral** persistente (en desktop) con las secciones principales (Inbox, Leads, Calendar, etc.). Íconos claros (ej. ícono de correo para Inbox, calendario para Calendar, gráfico para Reports). La sidebar permite **colapsar** en icon-only para maximizar espacio si el usuario desea. - **Topbar** superior con: saludo/nombre del agente, ícono de notificaciones (campana), quizás acceso rápido a buscar (una caja de búsqueda global para leads/listings) y menú de perfil (logout, switch modo oscuro). En topbar también podría mostrarse el **saldo de créditos** con icono de moneda/bolsa y link a recargar [54] (dado que monetización es importante). - **Contenido principal**: área derecha/adaptable donde se muestran las páginas según la sección seleccionada.

El diseño es **responsive**: - En **desktop** (>=1024px), sidebar visible, contenido panel, idealmente mostrando múltiples paneles a la vez en Inbox (como la imagen de Zillow desktop arriba). - En **tablet/mobile**, la sidebar se convierte en un menú tipo drawer desplegable (hamburger). La navegación primaria quizá se replantee en tabs inferiores en mobile (Zillow Premier Agent app usa bottom nav con Inbox, Contacts, Tasks, More [55]). En nuestra web mobile, podríamos usar un tabbar fijo inferior para 4-5 secciones clave (Inbox, Calendar, Listings, More) para emular experiencia app. Alternativamente, un menú hamburguesa completo. Decidiremos tras wireframes móviles; lo importante es que sea usable en el teléfono ya que muchos agentes trabajan sobre la marcha. - Páginas como Inbox en mobile se volverían multipaso (lista leads -> tap abre pantalla de chat detalle, etc.), posiblemente usando rutas anidadas o modales fullscreen.

Command Palette & Shortcuts: Implementaremos un **command palette universal** (tecla rápida `Ctrl+K` o `⌘K`) para usuarios avanzados. Esto permitirá buscar rápidamente cualquier sección o acción: e.j. “ir a lead [nombre]”, “crear cita nueva”, “ver calendario hoy”. Usaremos una librería como *cmdk* para construirlo, que provee un menú modal con búsqueda rápida [56] [57] . La paleta mejora la navegación para power users y brinda una sensación moderna (muchas apps web de productividad la incluyen). Además, definiremos **atajos de teclado** adicionales: - Ej: `g + i` (go inbox), `g + c` (go calendar), `g + l` (go leads), similares a atajos de Gmail/GitHub. - Atajos dentro de vistas: en Inbox, quizá `j/k` para moverse arriba/abajo en lista de leads, `enter` para abrir, `esc` para cerrar detalle. - Soporte de atajos se hará con una librería (p.ej. react-hotkeys) para gestionar accesibilidad y colisiones.

Todos los atajos serán opcionales y documentados en un *?* help o dentro de la paleta (cmdk puede listar comandos disponibles). Se cuidará que no interfieran con inputs normales. Y respetaremos `prefers-reduced-motion` y accesibilidad (paleta y atajos deben funcionar con lector de pantalla apropiadamente, e.g. cmdk se comporta como combobox accesible [58]).

En cuanto a **estructura del contenido** dentro de páginas: - Inbox: podría reutilizar componentes de Leads (lista) y Chat (detalle). Podríamos implementar Inbox como una variante de Leads list plus el panel chat. Alternativamente, Inbox es su propia página tri-panel como Zillow desktop (recomendado en desktop). En mobile, Inbox route primero muestra lista; seleccionar un item navega a something como `/agents/inbox/123` que renderiza chat en pantalla completa. - Leads: quizá en desktop no se usa mucho si Inbox ya lista todo; pero la diferencia es que Leads page podría ser más extensa (mostrar todos campos en tabla, apta para exportar, etc.). Lo tendremos para agentes que quieren ver todos contactos independientemente de mensajes. - Calendar: se integrará con leads (cada cita enlazada a un lead) y con listings (posible open house scheduling). - Settings: subdividida o en una sola página con secciones con anchors.

Responsive y Mobile-first: El diseño será mobile-first usando Tailwind CSS. Garantizaremos: - **Tablas y listas** se puedan scroll lateral en mobile si exceden ancho, o adaptaremos a tarjetas stacked. - Paneles: en mobile se transforman en pasos modales o páginas apiladas (ej. en Inbox, de lista a detalle). - Interacciones de arrastrar (drag & drop) en mobile: se debe probar usabilidad – quizás ofrezcamos menú de cambiar estado en cada card como alternativa, ya que drag en mobile puede ser engorroso. - Touch targets suficientemente grandes (mín. ~48px). - Evitar *hover-only* features en mobile (usar `:active` o toques prolongados si necesario).

Probaremos en distintos tamaños (iPhone SE small, smartphones grandes, iPad, desktop 1080p y monitores grandes). Usaremos componentes responsivos de shadcn/Radix que ya manejan muchos detalles (ej. Drawer for mobile menus, Dialogs for modals).

En suma, la información se organizará en secciones lógicas, con navegación lateral para eficiencia, y flujos contextuales (ej. desde un lead se puede programar cita sin salir usando un diálogo). La arquitectura debe minimizar clics para acciones comunes: por ejemplo, desde Inbox (lista de leads) se debe poder marcar estado o agregar nota sin tener que ir a Leads page separada; incorporaremos acciones inline. Este enfoque “inseparable tasks” evita la fragmentación que haría el usuario rebotar entre secciones.

5. Motion System y Micro-interactions Spec

Diseñaremos un **sistema de motion** coherente en toda la plataforma, inspirado en los principios de Material Motion y las directrices de Radix UI, pero ajustado a nuestro tono “interactive + alive”. Documentamos a continuación las guías y ejemplos:

Principios de Animación:

- **Rapidez y Utilidad:** Las animaciones deben ser breves (generalmente < 300ms) y no bloquear la interacción. Sirven para comunicar estado (éxito, error, cambio) o guiar la atención, nunca para adornos innecesarios. El agente valora la velocidad, así que optamos por **animaciones rápidas (150–200ms)** para acciones frecuentes (hover, clicks) y quizás **algo más largas (300ms)** para transiciones de página o modales, dando sensación de suavidad sin retraso.

- **Propósito:** Cada motion tiene una razón clara: resaltar un elemento recién añadido/cambiado, confirmar una acción, desplazar la vista de forma que el usuario mantenga contexto. Evitaremos animaciones aleatorias o en bucle que distraigan.
- **Consistencia:** Usaremos un conjunto de **tokens de motion** uniforme. Por ejemplo, la duración estándar de **fade/scale** será 150ms para micro-interacciones; para transiciones de vista ~300ms; easing predeterminado similar a `ease-out` (rápido al inicio, suave al final) para entradas, y `ease-in` para salidas.
- **Accesibilidad:** Respetaremos `prefers-reduced-motion` – si el usuario lo tiene activado, minimizaremos animaciones (las no esenciales se desactivan, las esenciales se acortan/simplifican, sin parallax ni movimientos grandes). Además, toda información comunicada por motion (ej: un error shake) debe también darse en texto/visual no animado para que nadie se la pierda.

Motion Tokens (valores recomendados):

- **Durations (ms):** Instant feedback 100–150ms (hover highlights, button presses); Transitions pequeñas 200ms (dropdowns, tooltips); Transiciones de pantalla/dialogo 300ms; Animaciones más notorias (como confeti) quizás 400–500ms en total pero componiendo varias partículas.
- **Easing:** Utilizaremos curvas suaves: `easeOutQuad` o similar para la mayoría (equivalente CSS `cubic-bezier(0.25, 0.1, 0.25, 1)`), y tal vez `spring` ligero para elementos de entrada (utilizando Framer Motion's spring presets para overshoot sutil en confeti o success pops). En general: Entradas = ease-out, Salidas = ease-in, Movimientos continuos = linear o ease-in-out según caso.
- **Transform Distances:** Por ejemplo, elementos que aparecen desde fuera de vista: entrarán desplazados ~8px y se acomodan (pequeña traslación Y o X). Hover elevaciones: traducir y=-2px o scale 1.02. Drags: elemento arrastrado puede elevarse 5px en z (simulado con shadow).
- **Opacity & Blur:** Fades from 0 to 100% opacity se usan para aparecer/desaparecer casi todo (tooltips, modales). Podemos usar *motion blur* sutil en swipes (ej. toast que se desliza out con blur). Sin embargo, los efectos de blur intensivo podrían afectar perf en dispositivos bajos – limitarlos a cosas raras (fondo de modal blur 2px para foco).
- **Elevation Shadows:** No es exactamente motion, pero definimos que cuando un elemento se anima a primer plano (ej. un menú desplegable), aplica shadow elevación que se incrementa en animación (de shadow pequeña a grande simultáneo con movimiento).

Animaciones por Componente UI:

- **Botones:** Feedback inmediato al hacer hover (desktop) con un sutil cambio de fondo/scale: duración ~100ms. Al click, usaremos un efecto pressed: por ejemplo, reducir ligeramente el botón (scale 0.98) durante unos 50ms con bounce back. Botones importantes (CTA) al hover pueden tener un leve *elevate* (shadow intensifica). Framer Motion nos permitirá quizás usar `whileHover` y `whileTap` para estos estados.
- **Tabs / Navegación:** Cambiar de tab hará moverse un **indicador** (underline bar) con animación deslizante de la posición anterior a la nueva en ~200ms (easing ease). El contenido de tab puede hacer un fade o slide corto horizontal para indicar cambio (pero con `prefers-reduced-motion` saltaremos la animación de contenido).
- **Selects / Dropdowns:** Al abrir, el menú aparece con un **fade-in + slide-down** de ~150ms [59]. Usaremos Radix UI Select que tiene animaciones por defecto personalizables – configuraremos `motionPreset="fast"` con transform origin top. Cierre con fade-out 100ms. Items highlight on arrow navigation with a quick background fill (no delay).
- **Tooltips:** Aparición retardada (~300ms delay tras hover) para no molestar, luego fade-in rápido (100ms). Usaremos Radix Tooltip with its animation props.

- **Dialogs/Modals:** Al abrir un dialog modal, aplicaremos un **scale-in** ligero desde 95% a 100% + fade-in, duración ~200-250ms, easing out. El overlay de fondo aparece con fade ~150ms. Al cerrar, lo inverso (fade-out + scale 100% a 95% to shrink). Esto da una sensación de zoom leve al contenido del modal, que es amigable. Drawer side-panel similar: slide desde la derecha (e.g. para Notifications center o Menu mobile) con amortiguación suave.
- **Drawers (slideover panel):** Animar posición X de 100% a 0% en 250ms ease-out; overlay fade-in concurrently.
- **Cards y Hover States:** Tarjetas de lead/listing al hover en desktop harán un **elevate**: shadow intensifica y quizá un muy sutil translate Y -2px. Duración 150ms. En mobile sin hover, podemos implementar touch feedback distinto (ej. active state highlight).
- **Skeleton Loading & Shimmer:** Cuando carguen datos (e.g. lista de leads inicial, gráficos), mostraremos **skeletons** – barras/rectángulos con animación de *shimmer* (gradiente moviéndose de izq a der continuamente) para indicar carga [59] . Usaremos preferentemente un pequeño util en Tailwind + CSS Keyframes para shimmer ~1.5s loop (prefers-reduced-motion: disable shimmer, just solid grey).
- **Content Reveal:** Una vez llegan datos, podemos animar la aparición de la lista: por ejemplo, staggered fade-in de items (cada item de lista apareciendo con 30ms de diferencia). Esto da fluidez inicial sin ser muy notorio.
- **Empty States animados:** Para pantallas sin datos (ej. “No leads todavía” o “Sin tareas pendientes”), incluiremos ilustraciones o iconos con un micro movimiento sutil (por ejemplo, un icono de buzón parpadeando suavemente o subiendo y bajando 5px en loop lento). Muy leve y con prefers-reduced-motion: static. Alternativamente, podemos usar una pequeña animación Lottie loop en empty states para dar calidez.
- **Toasts y Snackbars:** Notificaciones tipo toast (ej: “Lead guardado con éxito”) aparecerán desde **abajo a la derecha** (en desktop) o arriba (mobile), con una animación combinada: fade-in + translateY. Duración ~100ms para que se vean rápido. Pueden auto cerrarse en ~4s con un progress bar sutil. Si se deslizan (swipe) para cerrar, seguiremos la interacción con la posición del dedo (Radix Toast soporta swipe to dismiss [59]).
- **Progress Bars:** Por ejemplo, en carga de un archivo o en progreso de perfil, la barra se llenará con animación de ancho y cambio de color si completo. Para micro-recompensas, al terminar podríamos lanzar pequeñas confetti animadas: confeti es un **micro-interaction celebratorio** útil tras hitos (perfil 100%, o primer lead convertido a cita). Usaríamos una lib de confetti (canvas) con 5-10 partículas color azul/verdes, animadas en 500ms cayendo; bien calibrado para no saturar (y disabled con reduced-motion).
- **Icons y micro-estados:** Podríamos animar iconos de favorito (corazón latiendo al marcar) o envío de mensaje (avión de papel moviéndose minimal). Ej: al dar click “enviar” mensaje, podemos rotar el icono de avión 45° y lanzarlo unos px hacia arriba con fade (simbolizando envío) – duración < 500ms, con after-state mostrando un check breve. Estas sutilezas hacen la UI sentirse viva.

Estados y Transiciones de Datos:

- **Optimistic Updates:** Muchas acciones se reflejarán inmediatamente en UI antes de la respuesta del server (optimistic). Ejemplos:
- Al cambiar estado de lead arrastrándolo a otra columna, el lead **desaparece de la columna original y aparece en la nueva con highlight** al instante, sin esperar confirmación. Si el server luego falla, mostraremos un mensaje de error y revertiremos (el card volverá atrás con una animación de *shake* leve para indicar que no se pudo).
- Al enviar un mensaje en el chat, mostraremos el mensaje en la conversación inmediatamente (en gris claro o itálico indicando “enviando…”), quizás con un spinner pequeño, y luego al

confirmar marcamos como enviado (cambiamos a texto normal). Si falla, ese burbuja se pone roja o muestra "!" y permite reintentar.
- Al marcar una tarea como completa, la movemos a la sección completados con un tachado animado y fade.

Estas **confirmaciones visuales** deben ser claras: usaremos *toasts* para algunas (ej. "Lead actualizado ✓" que se desvanece solo) o cambios de estilo (ej. lead card flash verde 1s de fondo al guardar nota exitosamente). La idea es evitar que el agente dude si su acción funcionó.

- **Errores recuperables:** Si ocurre un error (no internet, server error) en una acción:
- Botones mostrarán estado error temporal: p.ej. un botón de guardar cambia su texto a "Error" con color rojo y vibra ligeramente (animación de **shake horizontal** 10px 2 veces en 0.5s) para llamar la atención [59] . Luego vuelve a estado normal con texto original, permitiendo reintento.
- Mensajes error inline (texto rojo) pueden *deslizar fade-in* cuando aparece y fade-out si desaparece tras corrección.
- Notificar error general: un toast rojo con icono ⚠ slide-up.
- En general, los shakes suaves indican "algo salió mal / hay un problema que debes atender", como un campo obligatorio no llenado.

Ejemplo: si intentar agendar cita sin elegir hora, el picker field puede resaltar borde rojo y **vibrar** brevemente. Esto da feedback instantáneo además del mensaje "Selecciona hora".

- **Indicador "Guardando..." (Save state):** Para ediciones en línea (ej. cambiar precio de listing), implementaremos un **indicador de auto-guardado** discreto: podría ser un icono de guardado en la esquina que aparece girando mientras hay cambios pendientes y desaparece cuando todo está sincronizado. No queremos que el usuario deba hacer click "Guardar" manual siempre. Este indicador debe transmitir confiabilidad: por ej. un texto "Guardando..." en gris que cambia a "Guardado ✓" por 1 segundo y luego se oculta. Animación: usando Framer Motion, podemos transicionar la opacity del texto. Si `prefers-reduced-motion`, igual lo mostramos sin flicker excesivo.

prefers-reduced-motion y Performance:

Cumpliremos la media query `prefers-reduced-motion: reduce`: - Desactivaremos las animaciones no esenciales: shimmer de skeleton, confeti, transiciones de navegación (podemos saltar directamente al estado final), animaciones decorativas de iconos. - Animaciones críticas (feedback de botón, modales) las simplificaremos: en lugar de scale+fade modales, quizás solo fade instantáneo. O mantenemos alguna pero reduciendo tiempo a ~100ms sin desplazamiento para evitar mareo. - En Framer Motion, podemos leer `shouldReduceMotion` del contexto (existe hook) y condicionar variantes más sencillas.

En cuanto a **performance**: - Todas las animaciones se harán con transform y opacity principalmente (evitando propiedades que causan *layout reflow* pesado como top/left, width animado – en su lugar usamos translate/scale). Así logramos mantener 60fps incluso en dispositivos modestos. - Limitaremos el *layering* excesivo: muchas animaciones simultáneas pueden saturar, así que escalonaremos sutilmente (stagger) en cargas pero no más de ~5-6 elementos concurrentes animando en todo momento. - Utilizaremos **Framer Motion** para secuencias complejas (aprovechando su animation loop en RAF), pero también combinaremos con CSS transitions para cositas simples (hover CSS es suficientemente bueno). Evaluaremos caso a caso; Framer nos da consistencia y control, pero usaremos la prop `layout` para animar reordenamientos sin calcular manual. - Evitar thrashing: animaciones de listas largas (por ej. confeti generando 100 elementos DOM) pueden ser costosas; usaremos ~10 partículas max y removerlos del DOM al terminar (lo podemos hacer con canvas confetti mejor). -

Probaremos con performance profiling para asegurar que las animaciones no causan jank en scroll o input. Por ejemplo, el shimmer se hará con CSS background property para evitar JS overhead.

En resumen, el sistema de motion buscará añadir **claridad, inmediatez y una capa de "pulido" visual** a la interacción. Los agentes deben sentir la aplicación ligera y moderna, donde cada acción tenga una reacción visible que les confirme lo que pasó. Esto aumenta la confianza en la herramienta (saben que se guardó, que se envió, etc.) y aporta placer sutil en el uso (micro-recompensas animadas por completar algo). Al mismo tiempo, mantendremos opción de minimizar motion para quienes lo necesiten, y priorizaremos siempre la velocidad percibida (mucho de esto sucede en <0.3s).

6. Flujos Críticos (User Flows)

A continuación describimos los flujos más importantes punto a punto, considerando variantes y edge cases:

A) Registro / Onboarding de nuevo agente:

1. **Sign Up inicial:** El agente (no existente en sistema) llega a una pantalla de registro especial para agentes. Ingresa datos básicos: nombre, email, contraseña. *Edge:* si el email ya existe como usuario cliente, ofrecer conversión a cuenta agente con pasos extra (o usar login).
2. **Verificación Email:** (Suponemos necesaria por seguridad). Tras registro, se envía email de verificación; la UI muestra "Revisa tu email". *Edge:* Opción de reenviar link; si expira el token, manejar error.
3. **Perfil Básico:** Una vez email verificado, el agente entra a un **wizard de onboarding** dentro del portal de agentes. Paso 1: foto de perfil (opcional pero recomendada), licencia inmobiliaria (número, opcional), **zonas de operación** (ej. seleccionar ciudades o zonas geográficas donde trabaja). Probablemente un selector de regiones multi-select.
4. **Especialidades:** Paso 2: elegir tipos de propiedad o cliente (comprador, vendedor, alquiler, comercial, residencial, etc.) – esto nos ayuda a luego rutar leads adecuados. *Edge:* Si no elige, default "generalista".
5. **Disponibilidad y preferencias:** Paso 3: configurar *My Schedule* – rangos horarios disponibles para mostrar propiedades (ej. Lun-Sab 9-18h). Esto alimentará la función de calendario y la lógica de asignar citas en horarios adecuados. También preguntar si quiere *Notificaciones por email/SMS* (opt-in).
6. **Resumen y completar:** Paso final: mostrar un resumen de perfil 80% completo con barra de progreso. Indicar qué falta (ej. "Añade una bio y verifica tu identidad para completar perfil más tarde en Settings"). Botón **"Empezar a usar la plataforma"**.
7. **Post-onboarding:** Se presenta un pequeño tour (tooltips guiados): señalar Inbox ("Aquí recibes nuevos leads"), Calendario, etc., para usuario primerizo. *Edge:* Opción de saltar tour.
8. *Edge Cases:* Si el agente pertenece a una agencia pre-creada (por un broker), podríamos tener un código de invitación en registro que lo vincule automáticamente al equipo. En tal caso, parte del onboarding podría ya venir llena (ej. zonas definidas por la agencia). Nuestro diseño soportará un parámetro inviteToken.

Este flujo asegura datos mínimos para que el agente reciba leads pertinentes y tenga configurado cómo quiere que lo contacten. Validaremos que no sea demasiado largo; preferible en 3-4 pasos. Podríamos posponer la parte de disponibilidad a más tarde si complica el registro inicial.

B) Flujo de Leads → Pipeline → Conversación → Cita → Cierre:

Este es el **flujo principal de trabajo** diario: 1. **Nuevo lead entra:** Un usuario del marketplace solicita información de una propiedad o contacto con agente. El backend asigna ese lead al agente (por zip, turno, etc.). En la plataforma del agente, en tiempo real aparece: - En Inbox: nueva entrada al tope de la lista, destacada (background claro) con icono de doble flecha que Zillow usa para indicar lead Premier [5]. - Se dispara una notificación (toast in-app, y si corresponde push/email). - Edge: si agente está offline, la verá al volver o vía email. 2. **Agente abre el lead:** Hace clic en la entrada. En la UI de inbox, se abre el panel de detalle con: - Mensaje del lead (p.ej. "Hola, me interesa el depto en…") en la vista de conversación. - Panel de propiedad (si venía por listing específico) con foto, precio, link a detalles. - Panel de info contacto: nombre, teléfono, email, requisito (e.g. "busca 2 recámaras"), timeframe ("Quiere mudarse en <3 meses"), pre-aprobado? etc [26]. - Recomendación de seguimiento: p.ej. "Responder en <5 min para maximizar conversión" (puede ser estático). - Estado del lead actual: "New" (por default). - Botones rápidos: "Llamar", "Enviar mensaje", "Enviar email" (estas acciones iniciarán la comunicación preferida; supongamos lead prefirió SMS, entonces aparece un botón "Enviar SMS" destacado). 3. **Primer contacto (conversación):** El agente decide responder via chat in-app (si el lead tiene app/chat) o via teléfono. En nuestro sistema, facilitaremos chat interno (similar a SMS). - El agente escribe un mensaje ("Hola, gracias por tu interés… ¿deseas visitar la propiedad?") y envía. El mensaje aparece en la conversación con marca de tiempo, inicialmente con indicador de enviado (un check vacío, digamos) que se llena al ser entregado. - Edge: Si lead respondió por email, quizá la conversación se etiqueta email; nosotros podemos consolidar todo en un mismo hilo chat (aunque diferentes canales es complejo). Probablemente trataremos todo como "mensajes" indiferenciados si es posible. - Si el lead contesta, el mensaje entrante aparece en el mismo hilo realtime. El agente ve indicador de nuevo mensaje (y suena quizás un ding). - *Hipótesis simplificación:* Todos los leads contactarán vía chat del portal (lo ideal), pero muchos pueden preferir llamada. Si agente hace click "Llamar", fuera de la app hará la llamada, pero podría luego anotar "llamada realizada" manual. Zillow registra llamadas también, pero nosotros en MVP no tendremos VoIP, así que la llamada es offline. - Tras contacto inicial, el agente marca el lead como "Contactado" (puede ser manual o automáticamente al enviar un mensaje – quizás automáticamente se cambia estado a Contactado). 4. **Actualización de estado pipeline:** Ahora el lead está en progreso. Supongamos tras chatear, el agente logra agendar una visita. Entonces: - Agente crea una **cita**: selecciona fecha/hora en el calendario integrado. Puede hacerlo desde el panel del lead (botón "Programar visita"). Eso abre un formulario emergente: fecha (con calendario), hora, método (presencial/virtual), y vinculado al listing de interés. En el lead panel se agrega al timeline "Cita programada para tal día". Estado del lead cambia a "Appointment Set" (etiqueta). - El lead recibe confirmación (email/SMS). - La cita aparece en `/agents/calendar`. - Si por alguna razón el lead no responde inicialmente, podemos tener un Task de follow-up en 1 día creado automáticamente o por el agente. 5. **Reunión/Visita:** Llega el día de la cita, el agente la lleva a cabo. En la plataforma: - Podría haber un reminder al agente ese día en la mañana ("Visita con Juan a las 5pm"). - Tras la hora, se espera input: ¿Se realizó? - El agente marca la cita como *Done* o como *No-show* si el cliente no apareció. Supongamos se realizó. - Estado del lead cambia a "Toured" (visitado), un paso más cerca del cierre. 6. **Conversión a transacción:** Luego de la visita, si el cliente está interesado, el agente puede marcar el lead como "Hot" o crear una tarea "Enviar propuesta". - Finalmente, si el cliente decide comprar/alquilar con el agente, ese lead se convierte en un **cliente cerrado**. En la plataforma, quizá no tenemos un flujo transaccional completo, pero el agente puede marcar manualmente el lead como "Closed – Venta realizada" y asociar qué listing vendió (si aplica). - Esto alimentaría estadísticas (conversion rate). - *Integración dotloop (futuro):* Zillow permite iniciar transacción en dotloop [21]. Nosotros quizás en V2 podríamos integrar un enlace a un sistema de contratos. Pero en MVP, lo manual servirá. 7. **Post-cierre – feedback:** Tras cerrar, el sistema podría sugerir "Pide al cliente una reseña" (mostrar un botón que copia un link público). Y para metric interna, marcar ese lead como *success*. - Ese cliente tal vez se mueve a una sección "Past Clients" dentro de Leads para futuros upsells (Zillow no menciona, pero es una idea). 8. **Edge cases en este flujo:** - Si el lead está duplicado (ej.

mismo cliente envía 2 solicitudes distintas): podríamos detectar y combinar en uno o al menos indicar duplicado. - Si el cliente deja de responder en medio: Se activa trigger “Lead sin respuesta 2 días” => crea tarea de follow-up o envía notificación para recontactar. - Si el agente rechaza el lead (por zona no atendida, etc.): Debemos soportar reasignar lead a otro agente. Zillow probablemente tiene un botón “Rechazar lead” en la app (no visto en textos, pero es común). Nuestro MVP podría simplemente permitir no aceptar (aunque en Premier Agent la idea es que los leads pagados no se devuelven fácilmente). - Cancelación de cita: Agente o cliente pueden cancelar. El agente haría click “Cancelar” en la cita, selecciona motivo (no disponible, cliente cambió plan). Eso notifica al cliente y cambia estado lead quizás de nuevo a “Contactado” o un estado “Cancelled”. - Reprogramación: similar cancel + crear nueva, pero nuestra UI de calendario permitiría arrastrar la cita a otro día/hora directamente.

Este flujo de lead end-to-end abarca varias partes de la plataforma funcionando en conjunto (Inbox, Chat, Calendar, Pipeline, Notifications). Es central garantizar que en cada paso el agente tenga claro qué hacer siguiente (por eso añadiremos sugerencias de “Next step”: ej. tras contactar, un botón “Programar visita”; tras visita, un botón “Marcar como cerrado” o crear tarea de seguimiento).

C) Flujo de Chat Agente-Usuario:

(Foco en UI de mensajería y SLA) 1. **Inicio de chat:** Como en flujo B, el chat puede iniciar porque el lead contactó. Alternativamente, si un agente quiere iniciar la conversación tras recibir el lead con solo info de contacto (tel/email), podría hacerlo enviando el primer mensaje vía SMS/email integrado. Nuestra plataforma ofrecerá plantillas de primer contacto. Ej: Agente abre lead -> campo de texto -> elige una **plantilla** “Mensaje de bienvenida” (como: “Hola {{name}}, gracias por tu interés en {{listing}}, ¿cuándo te viene bien una visita?”). Inserta esa plantilla en el input, agente puede editar o enviar directo. - *Edge:* Si el lead no tiene número de teléfono sino solo email, la plantilla puede ser de email. Idealmente unificamos como “mensaje” y el backend decide enviar por email vs notificación en app. (Hipótesis: integraremos canales pero MVP podríamos solo simular chat independientemente del canal). 2. **Composición & envío:** El agente escribe, quizás adjunta un archivo (botón clip). Adjuntos permitidos MVP: imagen (foto propiedad), PDF (brochure). Al adjuntar, se muestra una vista miniatura en el chat input. - Agente envía -> mensaje aparece en la lista inmediatamente con estado *sending…* (italics, gray). - Backend responde confirmación -> estado cambia a *sent* (normal text). Si hay entrega confirmada (difícil si SMS/email, pero en chat propio podríamos marcar *delivered/read*), podríamos mostrar doble check azul para leído. Zillow menciona *messages from contacts delivered to inbox*, no tanto sobre read receipts, pero podríamos agregarlos en chat interno. - *Edge:* Falla envío (no internet): mensaje se marca rojo “No enviado. Reintentar?”. Permitir tap para reintentar. 3. **Recepción de respuesta:** El lead responde. Si el agente tiene esa conversación abierta, el mensaje entra en tiempo real (WebSocket) al final, con efecto de nuevo (tal vez resaltar brevemente). Si está en otra pantalla, verá un toast “Nuevo mensaje de [Lead]” y/o el icono Inbox con contador. - Notar: Zillow app tiene push para mensajes [60] . Haremos similar con in-app notifs. 4. **Indicadores de escritura:** Podríamos omitirlo en MVP, pero si chat es en tiempo real (tipo webchat), un “…está escribiendo” sería nice. No es confirmado Zillow (posiblemente no tienen chat bidireccional en web, suele ser SMS relay). Lo dejamos como mejora futura. 5. **SLAs y follow-ups:** Si lead no responde en X tiempo, generamos alerta: “No tienes respuesta de Juan hace 2 horas. Puedes enviar un mensaje de seguimiento.” Esto como notificación o tarea. Zillow hace algo mediante Team reminders (líder puede obligar agente a actualizar estado), pero en individual, podríamos implementar un recordatorio. - Esto sería un cron/trigger en backend. En UI, manifestado como notificación de tarea “Pendiente: Dar seguimiento a Juan”. 6. **Notas internas:** Durante chat, el agente puede querer anotar algo que el cliente no ve (p.ej. “Cliente mencionó tiene que vender su casa primero”). Podríamos tener un campo “Nota privada” visible solo al agente. Zillow permite *custom notes* [61] . Implementación: un toggle en chat “modo nota” donde mensajes se marcan distinto (fondo amarillo) y no se envían al cliente. Simpler: campo separado en panel de lead “Notas”. Probablemente haremos lo segundo (notas en detalle lead). 7. **Cierre del chat:** No es un chat que se “cierre” formalmente, pero una vez el lead

avanza a cita, etc., la conversación continúa en mismo hilo hasta que se vuelva inactiva (podrían pasar semanas). - Mantenemos historial accesible siempre. - *Edge:* Si hay muchos mensajes, tendremos scroll infinito con virtualización (use case: chat extenso). 8. **Internos multi-agente:** Si en equipo, otro agente podría continuar el chat si lead se reasigna. Idealmente, la plataforma mantiene el mismo thread visible al nuevo agente. (Zillow rutéa a un agente en pausa al siguiente, no sabemos si comparten la historia – supongo que sí, el lead "connection" es compartido). - Para MVP supondremos un lead solo tiene un agente asignado a la vez. Si cambia, pondremos nota "Lead reasignado de X a Y" en el timeline, pero nueva persona ve los mensajes (ya que guardamos a nivel lead, no agente). - Debemos también permitir que miembros de un equipo vean los chats de leads del equipo? Zillow Team does let team leads see all leads performance, pero tal vez no los mensajes (privacidad). Esto definiremos según políticas. Posiblemente líder sí puede ver para coaching (Zillow Customer Experience feedback sugiere líder puede ver score de agente pero no necesariamente el mensaje exacto). 9. **Tecnología realtime:** Utilizaremos probablemente WebSockets (o supabase/Ably) para entregar mensajes instantáneos. Fallback: poll cada ~10s. Debe sentirse en vivo. 10. **Edge:** Usuario envía mensaje a medianoche -> agente tiene quiet hours hasta 8am -> podemos retener push notificación hasta hora permitida pero igual el Inbox mostrará entrada nueva (no sonará). Respetamos config notifs. 11. **Security:** Debe haber opción de **reportar spam/usuario** si un lead manda contenido indebido. Zillow no lo menciona, pero la consideramos: un botón "Reportar usuario" que notifica al soporte. Edge: En equipos, un admin puede revisar? Fuera de scope MVP posiblemente, pero tener en cuenta.

D) Flujo de Calendario y Visitas:

1. **Creación de cita (reserva visita):** Puede iniciar desde:
2. Botón global "+ Nueva Cita" en Calendar,
3. O desde un lead (como en flujo B, sugerido tras conversación).
4. UI: abre un formulario modal "Programar Visita" con campos: Contacto (si se inició desde lead ya viene seleccionado; si no, se puede elegir un lead existente o crear evento personal), Propiedad (opcional si es una visita a listing específico – se puede elegir entre los listings del agente o dirección libre), Fecha y Hora (selector calendario + dropdown horas libres según disponibilidad).
5. Edge: Comprobar conflictos (si ya hay otra cita en esa hora). Si conflicto, al guardar, mostrar error "Conflicto con otra cita de 14:00-15:00" con shake en ese field.
6. Confirmar: Al dar "Guardar", crea evento en calendario, envía notificación al lead (email "Cita confirmada para tal fecha") si aplicable.
7. Animación: toast "Cita creada" y en Calendario la casilla de esa hora parpadea resaltando la nueva cita.
8. **Confirmación por cliente:** Si implementamos confirmación cliente (Zillow posiblemente envía invitación por email), podríamos trackear si el cliente confirmó asistencia (por ahora, suponer confirmado implícitamente).
9. Podríamos tener un campo "Confirmado por cliente: sí/no" con un check manual.
10. **Recordatorios:** El sistema envía recordatorio al lead 1 día antes y/o 1 hora antes (depende integraciones, quizá V1 via email). Al agente, notificación push 1h antes (in-app badge).
11. **Reprogramar:** El cliente pide cambiar hora. El agente puede arrastrar el evento en la vista calendario a otro slot (en desktop) – eso abre un dialog "¿Reprogramar a nueva hora?" -> OK -> notifica cambio. En mobile, entraría a detalle evento (tap en evento abre modal con detalles y botones "Reprogramar" "Cancelar").
12. Reprogramar abre el mismo form con fecha/hora, enviará update.
13. La cita en UI mueve a nueva posición con animación de movimiento (posible con library calendar re-render).
14. Triggers: notificar lead.
15. **Cancelar:** Si el cliente cancela o ya no procede:

16. Agente pulsa “Cancelar cita”. Preguntar motivo (opcional dropdown: Cliente canceló, Vendida antes, etc.).
17. Marca evento como cancelado (podemos tachar en calendario o eliminarlo). Probablemente lo removemos de vista para no confundir, pero almacenamos en lead timeline “Visita del 5 Mayo cancelada”.
18. Notificar lead confirmando cancel.
19. Posible acción: ofrecer al agente reprogramar o seguir en contacto (depende motivo).
20. **No-show:** Si el día llega y cliente no aparece, agente puede marcar “No-show” en el evento (en UI se vería en detalle evento opciones: Completed, No-show).
21. No-show podría crear automáticamente una tarea follow-up (“cliente no asistió, reprograma?”).
22. Este dato alimenta métricas (no-show rate).
23. **Completar visita:** El agente marca la cita como realizada. En UI, quizás quitarla del calendario o marcar completada (depende si queremos mantener eventos pasados visibles; supongo sí, en vista mensual se ven pasados). Podríamos simplemente dejarla normal pero internamente en lead timeline “Visita realizada”.
24. Esto puede cambiar lead status a next stage (e.g. “Toured” como dijimos).
25. **Multi-calendarios:** Si agente pertenece a equipo con calendario compartido, podríamos en V2 tener vista de todos. MVP se enfoca en personal. No obstante, es útil ver disponibilidad de otros en la agencia para agendar en su nombre. Eso es complejo, se dejaría para Team.
26. **Integración externa (ShowingTime):** Fuera de MVP, pero mencionar: Zillow integra con ShowingTime (app de citas). Podríamos en futuro permitir al agente vincular su cuenta de calendarios externos y sincronizar, pero MVP es silo propio.
27. **Mobile UI particularidades:**
    - En mobile, la vista calendario mensual puede ser difícil; quizás damos vista agenda lista (lista de citas próximas).
    - Permitir swipe entre días/semana.
    - Asegurar que crear evento en mobile es sencillo (quizá un botón + que lanza el mismo form).
28. **Edge:** Doble-booking deliberado (agente propone 2 horarios al cliente) – no tenemos concepto de provisional vs confirmada. Si necesitan esto, por ahora que cree 2 eventos y luego borre el no elegido. Un refinamiento futuro puede ser marcar eventos tentativos.
29. **After sale / recurrency:** Si es comprador final, no hay recurrency. Si fuera un arrendamiento periódico, calendario podría tener repetición (Syncfusion lib lo tiene), pero por simplicidad no implementaremos repetición recurrente en MVP.

E) Flujo de Listings: Crear/Editar/Publicar, Verificación, Open House

1. **Crear nuevo listing:** Agente con rol listing agent quiere publicar una propiedad. (Esto puede ser poco frecuente pues muchas veces listings vienen del MLS o así, pero supongamos damos la opción de ingresar manual para promociones).
2. Navega a `/agents/listings`. Ahí hay un botón “+ Nuevo Listing”.
3. Formulario multi-paso (porque muchos campos: dirección, características, precio, fotos). Podríamos integrar un wizard o un solo form largo con secciones plegables.
4. Paso 1: Dirección (autocompletar Google?), tipo (casa, depto), estado (en venta, renta), precio, descripción corta.
5. Paso 2: Detalles: metros, # recámaras, amenities.
6. Paso 3: Fotos: arrastrar o seleccionar múltiples imágenes, usamos uploader con vista previa.
7. Al final, botón “Publicar”.
8. *Edge:* Si tenemos pipeline de aprobación interna, quizás el listing queda en estado “Pendiente aprobación” hasta que un admin verifique (eso dependede políticas).
9. En MVP asumimos se publica directo.

10. Tras crear, listing aparece en la lista con etiqueta “Publicado” y visible en el marketplace para clientes.
11. **Editar listing:** Desde `/agents/listings`, cada row o card tiene un botón Editar. Esto reabre un formulario (posiblemente mismo UI de creación o inline editing).
12. Ej: agente baja el precio: edita el campo precio, guarda.
13. Inmediatamente en UI marketplace se refleja (via backend).
14. Podríamos mostrar “Pendiente de actualizar...” pero generalmente se hace en segundos. Feedback: toast “Listing actualizado”.
15. Edge: Cambios críticos (subir muchas fotos) tardan – usar indicador guardando.
16. **Estados del listing:**
17. Agente puede marcar listing “Pausado” (temporalmente fuera mercado) o “Vendido/Arrendado”.
18. En UI, un combobox de estado. Si Vendido: pedimos info de venta (fecha, precio final, comprador).
19. Este cambio de estado:
    - Lo quita de listados activos públicos.
    - Permite a la plataforma generar lead de cierre (posible coincidencia con un lead actual, si es through us).
20. Representación: listing “Vendido” podría moverse a sección aparte (Histórico) en la lista o marcarlo con badge rojo.
21. **Request de Verificación:**
22. Para dar confianza, ofrecemos un proceso de verificación. Ej: agente sube documentos (escritura, ID) para comprobar que el listing es legítimo.
23. En UI listing, botón “Solicitar verificación”.
24. Abre modal: explica requisitos, botón “Subir documentos”.
25. Agente sube PDF/imagen (ID propietario, etc.), envía.
26. Estado listing cambia a “En verificación” (badge amarillo). Admin revisará offline.
27. Cuando admin verifica, marcamos listing “Verificado” (badge verde).
28. Notificación al agente: “Tu listing X ha sido verificado ”.
29. Esto es similar a, por ejemplo, Airbnb’s verified badges o algunos portales locales que verifican publicaciones.
30. Edge: Si es rechazado (doc inválido), notificar “Verificación rechazada, por favor reintenta”.
31. **Open House scheduling:**
32. Quizá en V2, permitir programar *open house* (visita abierta) en un listing. Esto es un evento en listing visible a público.
33. Agente podría en listing details añadir: “Open House el Domingo 5pm”.
34. Marketplace lo muestra.
35. Internamente, agente vería en Calendar también.
36. MVP omitiremos, pero es un flujo futuro.
37. **Listing Activity Feed:**
38. Dentro de la vista de detalle del listing en portal agente (posiblemente un panel derecho al seleccionar un listing en la lista), habrá un **feed de actividad**:
    - “123 vistas esta semana”
    - “10 usuarios guardaron este listing”
    - “2 nuevas consultas recibidas” (y podría linkear a esos leads).
39. Confirmado: Zillow dice agentes pueden optar a ver qué casas sus clientes vieron/guardaron [33], pero en listing perspective, Zillow al propietario da métricas. No vimos UI agente con feed, pero lo implementaremos.
40. Ese feed se poblará con eventos generados en backend (views, saves contados). Lo mostraremos con timeline-like (fecha y evento).
41. Edge: Poca actividad (0 views) – mostrar mensaje motivador “Comparte tu listing en redes para más visibilidad” en lugar de feed vacío.

42. **Monetización / Boost:**
43. Módulo credits: agente puede gastar créditos para “impulsar” listing. De UI listing, botón “Boost” -> ofrece paquetes (e.g. 5 créditos por destacar 7 días).
44. Si implementado, tras pago restar créditos, y marcar listing como “Destacado” (UI con estrella).
45. Este flujo depende de credits, así que MVP tal vez no, pero tener en cuenta.
46. **Eliminar listing:**
47. Si un agente quiere removerlo (ej. duplicado), puede archivar/eliminar.
48. Lo sacamos del portal público (pero quizá lo mantenemos internamente).
49. UI: botón “Eliminar” -> confirmar -> listing se quita de listado (o se mueve a sección “Archivados” si quisiéramos).
50. Edge: No permitir borrar si tiene leads activos atados (mostrar advertencia).
51. **Edge case:** MLS sync – si un listing viene de fuente externa, posiblemente no editable por agente. En lista podríamos bloquear edición con nota “Sync from MLS”. Fuera de scope por ahora; asumimos manual entry.

F) Flujo de Créditos/Billing:

1. **Consulta de saldo:** En la página Credits, al cargar hacemos fetch del saldo actual (por ej. 50 créditos). Se muestra prominentemente (“Tienes 50 créditos”).
2. Mostrar equivalencia monetaria si aplicable (p.ej. 1 crédito = $1).
3. También link “¿Cómo funcionan los créditos?” (ayuda).
4. **Comprar créditos (Recarga):**
5. Agente hace click en botón “Comprar créditos”.
6. Abre modal o página con opciones de paquete: Ej: 50 créditos, 100, 500 con descuento.
7. Elige una y método de pago.
8. MVP podríamos simular pago completado (asumiendo usamos algún servicio: stripe).
9. Tras pago, actualizamos saldo (optimistically +X, mostrando animación count-up).
10. Registro en ledger: nueva entrada “Compra de 100 créditos - $100” con fecha.
11. Enviamos recibo (email).
12. Edge: Si falla pago (tarjeta declinada), mostrar error en modal “No se pudo procesar, intenta otra tarjeta”.
13. **Historial de consumo (Ledger):**
14. Debajo del saldo, una tabla con movimientos:
    - Positivos: recargas (descripción “Compra vía tarjeta ****1234”),
    - Negativos: usos de créditos, e.j. “Lead Juan Perez - 5 créditos” o “Destacar listing Casa St - 10 créditos”.
    - Cada entry: fecha, descripción, +/- monto, saldo resultante opcional.
15. Permitir filtrar por tipo (carga/gasto).
16. Edge: Si muchos movimientos, paginar o lazy load.
17. **Uso de créditos en acciones:**
18. Por ejemplo, lead marketplace podría costar X créditos cuando el agente lo acepta.
19. En la plataforma, si llega un lead de cierto tipo de pago, podríamos:
    - Mostrar “Costo: 5 créditos” en la notificación del lead.
    - Al hacer click “Aceptar lead” (o tras cierto tiempo automáticamente), deducimos créditos.
    - Confirmamos con toast: “5 créditos deducidos. Saldo: 45”.
20. De igual forma, al usar un boost listing, se deduce.
21. Implementación: Al realizar esas acciones en UI, haremos request al backend para debit, y actualizamos estado local del saldo (con optimistic UI y verificación en respuesta).
22. Si saldo insuficiente:
    - Prevenir la acción: e.j. en lead nuevo, en lugar de aceptar, mostrar modal “Necesitas créditos para este lead. Recarga ahora.” con botón a recarga.

    - O permitir acción y quedar con saldo negativo? Mejor no.
    - UI debería mostrar advertencia si < costo. Quizás un badge rojo en icono de créditos si bajo.
23. Vamos a incorporar triggers: notificación “Saldo bajo (5 créditos restantes)” – esta aparece en centro notifs para incitar recarga (trigger de retorno monetización).
24. **Facturación (Invoices):**
25. Cada recarga genera un comprobante. En UI Credits, pestaña “Facturas” con PDF descargable de cada compra.
26. Podríamos no implementarlo completo en MVP, pero al menos un registro textual.
27. Edge: Empresa requiere factura fiscal – ese flujo manual offline por ahora.
28. **Configuración de consumo:**
29. El prompt menciona “reglas de consumo configurables (UI)”. Quizá se refiere a permitir al agente decidir en qué gastar aut vs manual:
    - Ej. “Auto-aceptar leads hasta X créditos por lead” toggle.
    - O “No gastar créditos en leads de alquiler” etc.
30. Esto suena complejo. MVP no haremos, a menos que interpretemos algo más sencillo: podría ser la capacidad de limitar gasto diario?
31. O quizás un slider “bid” si fuera subasta de leads. Pero no se ha planteado.
32. Por ahora, no implementaremos “reglas”, salvo quizá un ajuste “Aceptar automáticamente leads de <$Y costo” (hipótesis a validar si agentes lo querrían).
33. **Edge: Freemium vs Premium:**
34. Si un agente tiene 0 créditos, puede seguir usando portal para leads orgánicos gratis? Depende modelo. Suponiendo ciertos leads freebies y otros pagos.
35. Debemos manejar 0 saldo: no impedir ver cosas, solo no puede tomar leads premium o boost.
36. Quizá destacar call-to-action recarga en varias partes.
37. No queremos dark pattern de alarmar, pero sí alentar mediante UI sutil (ej. color del saldo cambia a naranja si <10, etc.).
38. **Error flows:**
39. Pago repetido cobrado doble -> likely handled por pago gateway, pero podríamos evitar doble-click multi compra (deshabilitar botón mientras procesando, spinner).
40. Carga de ledger fail -> mostrar error general “No se pudo cargar historial, reintentar”.

G) Flujo de Gestión de Equipo:

(Asume un usuario rol “Team Lead” realiza acciones de invitar y configurar routing)

1. **Invitar miembro nuevo:**
2. En `/agents/team`, líder ve lista de miembros actuales. Un botón “Invitar Agente”.
3. Click -> modal “Invitar nuevo miembro”: ingresa email del agente, rol (Agente o Admin).
4. Se envía invitación por email con link de registro.
5. UI muestra en lista como “(Pendiente) [email] - Invitación enviada”.
6. Edge: Si ya existe un usuario con ese email como agente independiente, posible conflicto (podríamos informar “Este usuario ya está registrado; no puede unirse por ahora” o permitir vincular? Depende negocio).
7. Para MVP, suponer agentes solo en un equipo a la vez.
8. **Aceptar invitación (lado del invitado):**
9. La persona hace click en link, se registra (o login si ya tenía cuenta), y el sistema la asocia al equipo.
10. Entonces en Team page del líder, el estado cambia a activo, mostrando su nombre.
11. Edge: Podríamos permitir cancelar invitación (remove pendiente).
12. **Asignar/Modificar roles:**

13. En la lista, líder puede cambiar rol de un miembro (ej. ascender a admin).
14. UI: dropdown de rol junto a nombre, onChange -> confirm y toast.
15. Roles definen permisos (Admin puede ver todos leads? Depende).
16. MVP roles: Admin (equivalente a Team Lead), Agent (miembro regular).
17. Quizá "Asistente" lectura sola (futuro).
18. **Pausar/Reactivar miembro:**
19. Junto a cada agente, toggle "Recibir nuevos leads: ON/OFF" (Zillow *pause* [14]).
20. Líder la puede apagar si un agente va de vacaciones.
21. Efecto: routing omitirá a ese agente en autos.
22. UI: Al pausar, podría aparecer un icono pausa en su fila.
23. *Edge:* Agente mismo debería poder marcar su pausa desde su perfil, pero Zillow lo hace líder.
24. En our design, permitiremos ambos quizá (agente puede solicitarlo, pero líder tiene control final).
25. **Reglas de ruteo:**
26. Sección en Team: "Lead Routing Rules".
27. UI: Tabla de reglas existente. Botón "Crear regla".
28. Form: criterio (tipo lead: comprador/vendedor, zona: seleccionar zonas, rango precio min-max, etc.), asignar a: elegir agente(s). Puede ser round-robin entre varios marcados.
29. También un toggle "Activar" para cada regla.
30. Ej: "Leads con zona = X, asignar a Agente Maria".
31. También prioridad de reglas (orden).
32. Un agente por default recibe todo si no hay regla que lo excluya.
33. MVP podemos simplificar: solo regla por zona y precio, con asignatario uno o cola.
34. Zillow sugiere que tienen por tipo, precio, loc [14], lo seguimos.
35. Una vez config, cuando un nuevo lead llega, backend decidirá siguiendo estas reglas.
36. Pese a ser mayoritariamente backend, pondremos UI para configurarlo.
37. Edge: Reglas conflictivas (dos aplican) – podría asignar al primero en orden. Comunicar orden al usuario en UI arrastrando reglas (draggable list).
38. Sino, advierte si solapan.
39. **Reasignar lead manual:**
40. En listado de leads (quizá en Contacts tab), líder puede reasignar uno a otro agente (ej. si cree que otro lo manejará mejor).
41. UI: en detalles del lead, si líder, mostrar dropdown "Asignar a..." con lista miembros.
42. Al cambiar, el lead aparece en inbox del nuevo agente (y desaparece del anterior salvo que permitamos ver todos).
43. Notificar a nuevo ("Lead Juan te fue asignado por Admin"), notificar al viejo ("Lead Juan reasignado a Pedro" quizás para clarificar).
44. Internamente actualizar ownerId del lead.
45. Edge: Si lead estaba en conversación con anterior, esto se transfiere. Podría ser raro para cliente cambiar de agente sin aviso, pero sucede si uno está ocupado. Zillow lo contempla (pausa y reasigna).
46. **Visibilidad de leads en equipo:**
47. Posible opciones: A) Cada agente ve solo sus leads asignados; líder ve todos. (Zillow likely this). B) Compartir leads (poco probable por privacidad).
48. Iremos con (A): líder tiene toggle/filter "Ver todos leads del equipo" en su Inbox. En la UI, quizá en Inbox hay sección Team con sub-carpetas por agente (no sobrecargar MVP).
49. Más sencillo: líder se puede suplantar agente X en UI para ver su pipeline. O un Report que listan leads y quién los tiene.
50. MVP: no mostrar leads ajenos salvo en Reports.
51. **Team Performance:**

52. En Reports o Team tab, líder ve métricas por agente: # leads asignados, tasa respuesta de cada uno [46] , conversiones, etc.
53. Podría ser una tabla: Cols: Agente, Leads recibidos, % respondidos <5min, Citas programadas, Calificación media (si feedb).
54. Esto se inspira en Zillow Team report [46] .
55. MVP no heavy charts, pero at least some stats list.
56. Edge: rank highlight quien mejor performance.
57. **Chat colaborativo/team:**
58. Si se reasigna un lead, nuevo agente debería ver historiales.
59. No todos en equipo participan en un chat a la vez (no un group chat, es 1 agente vs cliente).
60. Por lo tanto, no hay flujo de "varios agentes responden a mismo lead" al mismo tiempo. Eso lo evitamos reasignando formalmente.
61. Si un agente se va offline, Zillow no menciona pero su sistema de distribución re-intenta. No reproduciremos esa complejidad ahora.
62. **Eliminar miembro:**
    - Líder puede quitar un agente del equipo (p.ej. se fue de la empresa).
    - UI: botón "Eliminar" en su fila. Confirmar.
    - Consecuencia: ese usuario ya no tendrá acceso a leads del equipo. Sus leads pendientes deberían reasignarse (prompt "Reasignar leads de Juan a quién?" al eliminar).
    - O, alternativamente, se puede eliminar y los leads quedan huérfanos para que el líder manual reasigne (pero mejor reasignar en paso).
    - Dado overhead, podríamos obligar a reasignar antes de permitir eliminar.
63. **Edge:** Transferir liderazgo (si broker sale, designa a otro).
    - Podríamos permitir líder cambiar rol de otro a Admin y de él a Agente, con confirmación.
    - Al menos anticipar en DB.
    - No fundamental MVP pero en equipos reales pasa.

En resumen, el flujo de equipo es complejo pero hemos delineado MVP de invitar -> asignar leads -> medir. Confirmado por Zillow: pausa, routing rules y accountability. Muchas configuraciones avanzadas (reglas) son hipótesis que iremos encendiendo según necesidad; nos aseguraremos de instrumentar su uso para ver si los líderes de equipo realmente las usan.

7. Diseño de loops de retorno (Habit Loops "Retorno")

Queremos lograr que los agentes regresen frecuentemente a la plataforma de manera voluntaria porque encuentran valor inmediato o notificaciones relevantes. Definimos a continuación 10 **"Return Triggers"** principales, cada uno con su evento disparador, valor ofrecido al agente, canal/UI donde se presenta y cómo evitamos spam o fatiga:

1. **Nuevo lead asignado:**
2. **Señal:** El sistema recibe un lead nuevo y lo asigna al agente.
3. **Valor para agente:** Oportunidad fresca de negocio (posible comisión). Sabe que responder rápido aumenta chances (lo cual Zillow enfatiza [19]).
4. **UI Surface:** Notificación push/email inmediata ("Tienes un nuevo lead: [Nombre] interesado en [propiedad]"), y dentro de la app un badge en Inbox. Además, listarlo primero con indicador "Nuevo".
5. **Frecuencia:** Depende de cuántos leads genere el marketplace; puede ser varias veces al día o semana. No limitaremos este, cada lead es valioso.
6. **Anti-spam:** No aplicamos quiet hours por default a leads nuevos, excepto si agente configuró no molestar en noche (en cuyo caso podríamos demorar la notificación push hasta mañana, pero el

lead estará en Inbox de todos modos). Nos aseguraremos de no duplicar: un lead = una notificación.

7. *Nota:* Este es el trigger más fuerte; debe ser fiable y con info suficiente para tentar a abrir (ej. incluyendo zona o presupuesto del lead si disponible).

8. **Nuevo mensaje de un lead (no leído):**

9. **Señal:** Un lead existente envía un mensaje/chat nuevo.
10. **Valor:** Requiere atención para mantener la conversación y no perder interés del cliente.
11. **UI:** In-app toast “Mensaje nuevo de [Lead]” si está en otra sección; icono Inbox con contador; push mobile instantánea (“[Lead]: Hola, podemos vernos mañana?”).
12. **Frecuencia:** Depende del intercambio, puede ser varias consecutivas si chat fluido.
13. **Anti-spam:** Si recibe varios mensajes seguidos, agrupar en una notificación (“3 mensajes nuevos de [Lead]”). Y aplicar quiet hours a notificaciones de mensajes no urgentes (posponer de noche, a menos que agente habilite).

14. *Táctica:* Mostrar extracto del mensaje en la notif (contexto). Marcar en UI qué leads tienen mensajes sin leer (bold).

15. **Cita confirmada o actualizada:**

16. **Señal:** Una cita con un lead se confirma (ya sea creada por agente o aceptada por cliente) **o** se reprograma/cancela.
17. **Valor:** Mantiene al agente informado de su agenda; reduce riesgo de olvidos o confusiones.
18. **UI:** Notificación en app (“Cita con [Lead] confirmada para [fecha/hora]”) y añadir al calendario con resaltado. Si cliente confirma vía email link, también notificar.
19. **Frecuencia:** Depende de # citas. Generalmente 0-2 al día.
20. **Anti-spam:** Consolidar en un resumen si hay varias en corto plazo (“2 citas nuevas confirmadas”). Posiblemente suprimir push de confirmación si agente mismo la programó (ya lo sabe), enfocando en confirmaciones de cliente.

21. **Quiet hours:** Citas pueden confirmarse a cualquier hora pero notificación puede esperar mañana si es de madrugada.

22. **Recordatorio de cita próxima:**

23. **Señal:** X horas antes de una cita agendada (ej. 2h antes).
24. **Valor:** Asegura que el agente no la pase por alto, puede preparar material.
25. **UI:** Notificación push “En 2 horas: visita con [Lead] en [dirección]”. Dentro de app, quizás banner en top del calendario/inbox el día de la cita (“Tienes 1 visita hoy”).
26. **Frecuencia:** Por cada cita. Probablemente 0-3 diarios.
27. **Anti-spam:** Si muchas citas, agrupar en notificación del inicio del día: “Hoy tienes 3 citas” (detalladas dentro app).

28. Permitir al agente desactivar o ajustar timing en settings (“Recordarme 1h antes en lugar de 2h”).

29. **Lead sin respuesta tras [N] horas:**

30. **Señal:** Un lead nuevo o en conversación quedó pendiente sin que el agente responda dentro de 2 horas (u otro SLA configurado).

31. **Valor:** Evita que se le pase contestar, protegiendo conversión y su respuesta rápida score.
32. **UI:** Notificación in-app y email (posiblemente): “No has respondido a [Lead] desde hace 3 horas. ¡Responde ahora para mantener interés!”. O se puede presentar como tarea urgente en Tasks.
33. **Frecuencia:** Sólo cuando ocurre, idealmente poco si agente es diligente.
34. **Anti-spam:** En lugar de repetir muchas veces, puede haber un segundo recordatorio a las 24h si sigue sin respuesta, pero no más allá (ya el lead probablemente frío).
35. Se desactiva si el agente marca manualmente que contactó por fuera.

36. Configurable: permitir al agente ajustar estos recordatorios o apagarlos si le molestan (aunque es para su bien).

37. **Tarea pendiente hoy:**

38. **Señal:** Al comenzar el día (8am), revisar tareas con fecha límite = hoy.
39. **Valor:** Ayuda al agente a planificar su jornada y no olvidar follow-ups prometidos.
40. **UI:** Dentro de app, sección Tasks resaltada con icono; notificación push o email “Tienes 2 tareas para hoy: Llamar a X, enviar documento a Y.”
41. **Frecuencia:** Diaria si hay tareas.
42. **Anti-spam:** Si ya se envió una daily summary, integrar ahí (ver trigger 9). Sino, enviar una notificación consolidada en la mañana en lugar de una por tarea.

43. En app, podemos mostrar un badge en ícono Tasks.

44. **Recomendación de Acción (Next Best Action):**

45. **Señal:** El sistema detecta alguna *oportunidad de mejora* para el agente:
    - Ej. “Listing X lleva 30 días sin interés, considera bajar el precio”.
    - O “Han pasado 2 semanas desde la visita con [Lead] y no hay actualización, considera hacer seguimiento”.
    - O “Completa tu perfil al 100% para ganar más confianza”.
46. **Valor:** Aporta asesoría proactiva que puede mejorar resultados (vende antes, recupera lead inactivo).
47. **UI:** Notificación en centro (no necesariamente push inmediata, podría ser en el digest diario). También en dashboard Overview un widget “Recomendaciones” con estas sugerencias, estilo checklist/growth.
48. **Frecuencia:** Event-driven pero moderada: quizás 1-2 por semana.
49. **Anti-spam:** Evaluar relevancia: no dar recomendaciones obvias repetidas.
50. Ej: no notificar “Baja precio” todos los días – tal vez una sola vez hasta que se actúe o se descarte (dar opción “Descartar sugerencia”).

51. Este es un loop para agentes más avanzados, lo pondremos suave en MVP (quizá solo perfil y listing stale).

52. **Notificación de actividad del listing:**

53. **Señal:** Un listing del agente alcanza cierta métrica notable o cambio:
    - “Tu listing *Casa Verde* recibió 50 vistas en las últimas 24h (20% más que el promedio) [33] .”
    - “5 usuarios guardaron *Depto Centro* esta semana.”
    - “*Casa Verde* fue marcada como favorita más que otros en su zona – quizás es buen momento para revisar ofertas.”
54. **Valor:** Mantiene al agente entusiasmado y consciente del interés, motivándole a ingresar a ver detalles/contactar interesados.

55. **UI:** Podría ser un **email digest semanal** con resumen por listing (Zillow envía daily buyer activity email [62]). In-app, en Reports o Overview, un card “Actividad de tus listings”.
56. Para eventos big (p.ej. primer save o súbito spike), se puede push: “¡Tu listing X recibió 10 saves hoy! Considera programar un open house.”
57. **Frecuencia:** Configurable: podríamos enviar **Digest semanal** consolidado (ej. lunes a 9am “Resumen semanal de tus listings: 200 views totales, 5 nuevos saves, etc.”).
58. Y push inmediatas solo para hitos especiales (p.ej. primer save, etc., con cuidado).

59. **Anti-spam:** Digest en lugar de notifs diarias a menos que agente solicite. Permitir opt-out de “marketing” notifs vs leads chat críticos.

60. **Resumen diario/semanal (Digest):**

61. **Señal:** Hora programada (ej. cada día a las 8am, o lunes 8am para semanal).
62. **Valor:** Un snapshot de todo: nuevos leads, tareas hoy, performance de la semana, etc., para que el agente tenga panorama y entre a la plataforma a gestionar. Zillow hace daily digest email de actividad de compradores [62], adaptamos a nuestro contexto general.
63. **UI:** Email preferentemente (ya que es resumen). In-app, mostrarlo en la página Overview (“Buenos días, esta es tu agenda: …”).
64. **Frecuencia:** Depende preferencia: quizás permitir togglear daily vs weekly. Por default, semanal para no saturar, daily si actividad alta.
65. **Anti-spam:** Sólo enviar si hay actividad/pendientes significativos (no enviar “nada nuevo” emails).
66. Y consolidar, para que en lugar de 5 notifs separadas en la mañana (tareas, leads), vaya todo en uno.

67. Quiet hours irrelevante ya que es en hora laboral típica.

68. **Completa tu perfil 100%:**

    - **Señal:** El perfil del agente está incompleto (menos del 100%). Por ejemplo, no ha subido foto o bio.
    - **Valor:** Perfil completo incrementa confianza de clientes y distribución de leads (si aplicable en ranking).
    - **UI:** Banner en app (“Completa tu perfil para obtener mejores resultados”) persistente hasta cierto punto. Y notificación eventual (“Tu perfil está al 80%. Añade tu licencia y foto para mejorar tu presencia”).
    - **Frecuencia:** Tal vez 1 notificación cada X semanas mientras siga incompleto.
    - **Anti-spam:** No machacar continuamente; quizás al 1 día de registro si no completó, luego a la semana. Y si ignora varias veces, dejar de molestar o esperar un mes.
    - Podría integrarse en digest (“Perfil: 80% completo”).

Centro de notificaciones in-app: Tendremos una sección (p.ej. icono campana en topbar) que al hacer click despliega un panel con lista cronológica de notifs recientes (últimos ~30 días). Cada entrada con tipo (icono), mensaje breve y timestamp. Esto captura los triggers anteriores para consulta central (ej. si se descartó toast, puede verlo ahí). El centro permite marcar como leídas, limpiar todo. Persistirá en DB o local storage.

Quiet Hours y Controles: En Settings -> Notifications, daremos opciones: - “No molestar entre [hora] y [hora]” para push (por defecto quizás 21:00-7:00, configurable). Durante quiet hours: - Los triggers críticos (nuevo lead, mensaje) se podrían seguir enviando quizás como notificación silenciosa para no penalizar (Zillow da opción de programar SMS sólo en horario elegido [51]). Podríamos posponer push

de lead hasta la mañana, pero eso retrasa la respuesta – tal vez mejor enviar pero sin sonido? Este punto es delicado. Quizás permitimos al agente decidir “Enviar notificaciones de leads fuera de horario: sí/no”. Por defecto sí con sonido suprimido. - Toggle on/off por tipo: - Leads/Mensajes (críticos, no se recomienda apagar pero damos check), - Tareas/recordatorios, - Sugerencias marketing (recomendaciones, digest). - Frecuencia digest: Off/Daily/Weekly.

Al dar control, el agente confía más y no percibe spam. Registraremos qué toggles ajustan para calibrar nuestro approach.

En general, **evitaremos dark patterns**: - No enviaremos notificaciones falsas o de urgencia artificial (cada notif tiene un genuino valor). - Daremos info clara en notifs (no solo “¡Hey entra ahora!” sin motivo). - Respetaremos si el usuario cierra repetidamente cierto tipo, tomaremos la señal de reducirlos.

Con estos loops diseñados, esperamos que: - Un agente activo deba entrar varias veces: para leads y mensajes (inmediato), para preparar citas (diario), y por curiosidad de estadísticas o sugerencias (semanal). - Un agente con menos leads al menos reciba el digest semanal y se mantenga enganchado viendo su progreso y eventualmente comprando créditos para obtener más leads.

8. Data Model Mínimo (TypeScript Interfaces) + State Machines

A continuación definimos las interfaces principales en TypeScript que modelan las entidades del portal, junto con las máquinas de estado relevantes:

```
// Identificadores básicos
type ID = string;
type Timestamp = string; // ISO date string

interface Agent {
  id: ID;
  name: string;
  email: string;
  photoUrl?: string;
  phone?: string;
  role: 'Agent' | 'Admin' | 'Broker';  // 'Admin' if team leader or staff
role
  teamId?: ID;    // if belongs to a team/agency
  specialties: string[];  // e.g. ['buyer', 'rental']
  serviceAreas: string[]; // e.g. ['Guatemala City Zone 10', 'Antigua']
  availability?: AvailabilitySchedule; // working hours
  profileComplete: number;  // e.g. 80 (percentage)
  // ... other profile fields like license number, bio, etc.
}

interface Agency {
  id: ID;
  name: string;
  brokerId: ID;   // Agent who is broker/admin
  members: ID[];  // Agent IDs in this team
  routingRules: LeadRoutingRule[];
```

```
}

interface Lead {
  id: ID;
  createdAt: Timestamp;
  assignedTo: ID;    // Agent ID
  teamId?: ID;       // if lead belongs to an agency's pool
  source: 'ListingInquiry' | 'ContactForm' | 'Referral' | 'Other';
  listingId?: ID;    // if tied to a specific property inquiry
  name: string;
  email?: string;
  phone?: string;
  message?: string;  // initial message from lead
  status: LeadStatus;  // pipeline stage
  stageUpdatedAt: Timestamp;
  tags: string[];    // e.g. ['Hot', 'MortgagePreapproved']
  timeframe?: '0-3 months' | '3-6 months' | '6-12 months' | '12+ months';
  budget?: number;
  preApproved?: boolean;
  // Relationship fields:
  conversations: Message[];  // chat history (could be separate store)
  tasks: Task[];
  appointments: Appointment[];
  notes: string;      // internal notes
}

type LeadStatus = 'New' | 'Contacted' | 'Qualified' | 'Appointment Set' |
                  'Toured' | 'Offer Made' | 'Closed' | 'Lost';

interface Message {
  id: ID;
  leadId: ID;
  sender: 'Agent' | 'Lead';
  senderId?: ID;       // Agent ID if sender is agent (null if lead in
general)
  channel: 'PortalChat' | 'Email' | 'SMS';  // channel used
  content: string;
  contentType: 'text' | 'image' | 'file';
  timestamp: Timestamp;
  delivered?: boolean;
  read?: boolean;
  // If contentType = 'file/image', include URL, etc.
  attachmentUrl?: string;
}

interface Appointment {
  id: ID;
  leadId: ID;
  listingId?: ID;
  title: string;         // e.g. "Visita: Casa 1234"
  start: Timestamp;
```

```
  end: Timestamp;
  mode: 'InPerson' | 'Virtual';
  location: string;      // address or video link
  state: AppointmentState;
  notes?: string;
  createdBy: ID;        // Agent who created
}

type AppointmentState = 'Scheduled' | 'Confirmed' | 'Cancelled' |
'Completed' | 'NoShow';

interface Listing {
  id: ID;
  agentId: ID;        // owner
  address: string;
  status: ListingStatus;
  price: number;
  beds: number;
  baths: number;
  // ... more details
  createdAt: Timestamp;
  verified: boolean;
  verificationStatus?: 'Pending' | 'Rejected' | 'Verified';
  viewsCount: number;
  savesCount: number;
  // Not storing list of viewers for privacy, just counts.
}

type ListingStatus = 'Active' | 'Pending' | 'Sold' | 'OffMarket';

interface ListingActivityEvent {
  id: ID;
  listingId: ID;
  type: 'VIEW' | 'SAVE' | 'INQUIRY';  // actions by users on listing
  timestamp: Timestamp;
  // Possibly, user id or lead id if registered, but not needed for agent UI
except count
}

interface CreditLedgerEntry {
  id: ID;
  agentId: ID;
  timestamp: Timestamp;
  type: 'Purchase' | 'LeadCost' | 'BoostCost' | 'Adjustment';
  description: string;
  amount: number;   // positive for purchase, negative for cost
  balanceAfter: number;
}

interface Notification {
  id: ID;
```

```
  agentId: ID;
  type: 'NewLead' | 'NewMessage' | 'TaskDue' | 'Reminder' | 'System';
  title: string;
  message: string;
  timestamp: Timestamp;
  read: boolean;
  relatedId?: ID; // e.g. leadId or taskId if relevant
}

interface Task {
  id: ID;
  leadId?: ID;      // if task is related to a specific lead
  agentId: ID;
  content: string;
  dueDate: string;   // YYYY-MM-DD or Timestamp if specific time
  completed: boolean;
  completedAt?: Timestamp;
  createdAt: Timestamp;
  // Could include reminder settings but for simplicity it's dueDate
}

interface AuditLog {   // For security actions like login, invite accepted
etc.
  id: ID;
  agentId: ID;
  action: string;
  timestamp: Timestamp;
  details?: string;
}

// Team routing rule structure:
interface LeadRoutingRule {
  id: ID;
  teamId: ID;
  criteria: {
    locations?: string[];  // list of areas or ZIPs
    priceMin?: number;
    priceMax?: number;
    type?: 'Buyer' | 'Seller' | 'Rent';
    // could extend with property type etc.
  };
  assignAgentIds: ID[];    // one or multiple agents (round-robin among them)
  active: boolean;
}
```

State Machines:

- **Lead Lifecycle (LeadStatus):** Un lead nuevo entra con estado `'New'` [63]. El agente lo contacta => cambia a `'Contacted'`. Si se califica como oportunidad seria => `'Qualified'` (opcional). Si se agenda cita => `'Appointment Set'`. Tras visita => `'Toured'`. Si el cliente hace oferta o

se prepara propuesta => `'Offer Made'`. Finalmente, si se cierra venta/contrato => `'Closed'`. Alternativamente, si se pierde (cliente desistió o fue con otro agente) => `'Lost'`.

Transiciones válidas (permitidas): - New -> Contacted (enviar mensaje o llamada) - Contacted -> Qualified (agente determina que es un cliente serio) - Contacted -> Appointment Set (directo a cita agendada) - Qualified -> Appointment Set - Appointment Set -> Toured (una vez hecha la visita) - Toured -> Offer Made (si se avanza a negociar) - Offer Made -> Closed (éxito) - (Desde cualquier estado pre-cierre) -> Lost (si se cae la oportunidad)

Validaciones: - No se puede cerrar (Closed) sin haber pasado por alguna etapa intermedia significativa (pero quizá podría un lead New ser Closed si inmediatamente compró otra cosa? Podríamos requerir at least Contacted). - Closed y Lost son terminales; una vez Lost, no se retrocede (a menos que duplicáramos lead). - Podríamos permitir reabrir Lost -> Contacted en caso de resucitar (eso es un edge case).

En la UI pipeline (kanban), permitiremos arrastrar entre columnas en orden lógico pero no saltos muy ilógicos (ej. New directamente a Closed no, a menos que se habilite con confirmación). El estado `'New'` se asigna automáticamente al crear lead [63]. También `'Contacted'` podríamos autoasignarlo cuando agente envía primer mensaje.

- **Appointment (cita) StateMachine:**
- Por defecto al crear = `'Scheduled'`.
- Podríamos marcar `'Confirmed'` cuando cliente confirma asistencia (si trackeamos). Sino, `'Scheduled'` ya implica confirmada.
- `'Cancelled'` puede ocurrir desde `'Scheduled'` en cualquier momento (antes de completarse). Una transición `'Scheduled'` -> `'Cancelled'`.
- `'Scheduled'` -> `'Completed'` cuando la visita se realiza.
- `'Scheduled'` -> `'NoShow'` si llega hora y cliente no llegó (agente marca).
- `'NoShow'` o `'Completed'` podrían transicionar a reprogramación generando nueva Appointment en `'Scheduled'` estado; la vieja queda como estaba.

Basic transitions: - Scheduled -> Completed (normal flow) - Scheduled -> Cancelled (before due) - Scheduled -> NoShow (after due, if agent chooses) - NoShow -> (optionally Completed if client apareció tarde, pero más simple es crear otra).

Validaciones: - No duplicar Confirmado/Completed sin reason. Completed y NoShow son terminal (no further transitions). - Cancelled is terminal.

- **Notification State:** Simple: `read` vs `unread`. Marking read is one transition. Possibly `dismissed` but we can treat as read for now.

- **Team Member State:** Could consider an internal state for agent in team: `Active` vs `Paused`. That is toggled by the pause feature. If paused, leads are not routed. It's less about state machine and more a boolean. But we can model:

- Active -> Paused (via team leader action)

- Paused -> Active (via team leader action or after specified away period)

- **Task State:**

- `completed: false/true`. Transitions:
    - false -> true (when agent checks it off).
    - Completed tasks might still be accessible under 'completed' filter.
    - If needed, could allow undo: true -> false if accidentally marked done (we permit within same session).
- **Credit Balance usage:** Not exactly a state machine but ensure:
- Cannot go below 0 (or if allowed negative, then must handle top-up).
- Purchase increases balance (multiple ways: credit card, promo code).
- Expenditure decreases until 0, then must stop actions requiring credits.
- **Lead Routing logic:**
- Could see as state:
    - `Assigned` vs `Unassigned` (if some leads initially go to team queue unassigned).
    - Zillow seems to auto-assign to agent instantly via rules [50], no pool waiting.
    - We'll implement auto-assign to one agent; if no rule matches, assign to default agent or broadcast? Possibly assign to team leader or round-robin all.
    - Not a user-facing state, more backend logic.

Estos modelos cubren la mayoría de datos del frontend. Durante implementación, podríamos refinarlos al integrar con APIs (e.g. fields rename). Pero son una base suficiente para empezar a codificar componentes con tipos correctos.

9. Librerías OSS recomendadas (Investigación y Selección)

Para cada categoría del stack frontend, evaluamos las 3 principales opciones open source (TypeScript friendly, licencias permisivas) y damos la recomendación final:

A) Data Grid / Tablas (manejo de datos tabulares, filtros, virtual scroll):

1. **TanStack Table (React Table)** – *MIT*, ~27.6k stars en GitHub [64]. Ofrece una librería **headless** altamente personalizable para construir tablas y datagrids en React. **Pros:** 100% TypeScript, enfoque modular (nos da sorting, filtering, pagination, grouping, selección de filas, etc. pero nosotros renderizamos el UI). Soporta **virtualización** de filas nativamente integrándose con TanStack Virtual, ideal para listas grandes [65] [66]. Documentación extensa y comunidad activa (usada en grandes apps). Sin dependencias pesadas. **Contras:** Al ser headless, requiere esfuerzo en estilos – pero esto encaja bien con Tailwind/shadcn ya que podemos aplicar nuestras clases. Su flexibilidad conlleva curva de aprendizaje.
2. *Calidad:* Muy alta, mantenida por Tanner Linsley et al., última versión TanStack Table v8 con soporte a React 18 concurrent.
3. *Accesibilidad:* Depende de nuestra implementación, pero la lib provee mecanismos para aria (ej. indicando rows/cols roles).
4. *TS Support:* Excelente (Type generics for row data etc.).
5. *Rendimiento:* Soberbio con virtualization; no re-render toda tabla en cada cambio (usa hooks bien diseñados).
6. *Ejemplo de funcionalidad:* Sorting multi-col con Ctrl+Click ya soportado, etc [67].

7. **AG Grid (Community Edition)** – *MIT (Community)* [68]. Un datagrid completo con muchas features listas out-of-the-box (edición, agrupación, pivot, export, etc.). **Pros:** Muy poderoso y performant (usa virtual DOM interna altamente optimizada; soporta millones de filas). Buena documentación y presencia en la industria. **Contras:** Pesado en tamaño (~500kb+), estilo por defecto feo y difícil de personalizar completamente a Tailwind aesthetic (usa sus CSS). Además, algunas advanced features solo en versión Enterprise (licencia comercial), pero Community cubre la mayoría (filtros, multi-sort). TS support es decente pero la API es algo compleja/ imponente.

8. *Mantenimiento:* Activo, empresa dedicada.
9. *Riesgo:* Podríamos toparnos con "necesito feature X que es enterprise". Queremos evitar vendor lock de pago, así que considerar cuidadosamente su uso de solo free features.
10. *Uso encaje:* Quizá sobre-dimensionado para MVP (que no tendrá miles de filas). Y incorporar su CSS en un Tailwind project puede ser "isla" aislada.

11. **React Data Grid (by rows/Comcast)** – *MIT*, 7.5k stars [69]. Este componente (anteriormente adazzle's) es **feature-rich y personalizable** [70] [71]. Tiene virtualization out-of-box, soporta ediciones de celdas, frozen columns, etc. **Pros:** Es un componente completo pero con un look neutro que se puede estilizar via CSS variables o override. 100% escrito en TS, con foco en rendimiento (virt. de columnas y filas). **Contras:** Menos popular que TanStack en momentum, y UI no integrada con Radix/shadcn (trae su propio minimal styles). Quizá más orientado a app tipo Excel/analítica más que simples listas. Documentación OK con ejemplos.

12. *Activity:* Maintained por Comcast, último commit reciente.
13. *Integration:* If we want a plug and play for a heavy data table (reportes?), podría servir. Pero para cosas simples (lista leads) puede ser overkill comparado con escribir map de array con tailwind.

Recomendación Final - Data Grid: TanStack Table es la opción elegida. Su enfoque headless nos permite integrarlo a nuestra UI kit (shadcn + Tailwind) sin conflictos, y activar solo las features que necesitamos. La licencia MIT y comunidad robusta nos dan confianza. Ya soporta TS first-class [72] y es extremadamente extensible. Para MVP, podemos usarlo inicialmente para la tabla de leads con filtros (por ejemplo, filtrar por status fácilmente) y escalar a vistas más complejas (reportes tabulares, etc.). AG Grid se descarta por su peso y potencial bloqueo de features enterprise; React Data Grid es atractivo pero TanStack ofrece más flexibilidad y un stack unificado (podemos usar TanStack Virtual, etc., del mismo org). Además TanStack Table es altamente probado en escenarios variados (de simple tables a huge grids).

B) Charts / Analytics:

1. **Victory (Formidable)** – *MIT*, ~11.2k stars [73]. Librería de gráficos modular para React (y React Native), basada en d3 internamente pero con API declarativa sencilla. **Pros:** Ya la usamos actualmente; tiene componentes para ejes, líneas, barras, pies, etc. Es completamente open-source y activamente mantenida (Formidable indicates Active status [74]). TS support: Victory recién en versión 35 incluyó definiciones; hay @types/victory también [75]. Buen control estético via props y estilos, y animaciones integradas (animate prop) para transitions. **Contras:** El bundle puede ser algo pesado si importamos todo (pero podemos import sub-packages). Documentación un poco dispersa pero hay tutoriales. En rendimiento, para <100 data points va bien; en charts con miles de puntos, no es tan optimizado como canvas.
2. *DX:* Familiar (ya adoptado en código actual).

3. *Integration:* Victory uses SVG, which meshes well con React and can be styled via props/CSS easily.

4. **Recharts** – *MIT*, ~19k stars (muy popular) [76]. Basado en React + D3, ofrece componentes sencillos (LineChart, BarChart, etc.) con pasar data. **Pros:** Fácil de usar, buen aspecto por defecto, responsive out-of-box. TS support: tiene .d.ts (authors maintain TS defs). Animaciones incorporadas para mount. Documentación decente + muchas respuestas online. **Contras:** Menos bajo control fino: styling a veces limitado a props predefinidas. Conjuntos de datos muy grandes pueden ser lentos (usa SVG). Estuvo un tiempo con mantenimiento lento, pero sigue actual (último release 2023).

5. *Performance vs Victory:* Similares escalas.
6. *Compatibility:* Recharts uses a specific approach, might conflict if mixing with Victory in project (not conflict per se, but duplicative weight if we use both).

7. Muestra issues TS en pasado, pero aparentemente se resolvió.

8. **Nivo** – *MIT*, ~13.9k stars [77]. Conjunto amplio de gráficos (line, bar, pie, heatmaps, geo, etc.), configurables y con bonito diseño predeterminado. **Pros:** Gran variedad; trae animaciones via react-spring; theming global. Genera gráficos SVG, HTML Canvas o incluso API render. TS support: no escrito en TS pero tiene definiciones (comunidad) [78], puede requerir some fiddling. **Contras:** Más pesada, e integrada (menos modular que Victory). Personalización más por theme config que por direct element override. Aprender su API (tiene muchas props).

9. *DX:* Muchos ejemplos en su docs interactive.
10. *Case:* Útil si necesitamos gráficos complejos (calendario de actividades, etc.), pero MVP quizá no.

Otras: **ECharts (Apache-2.0)** – extremely powerful, but using via React wrapper is a bit imperative (and heavy). **Chart.js** (MIT) – proven but canvas-based and requires manual config, good for simple charts but not as React idiomatic out-of-box (though react-chartjs-2 wrapper exists).

Recomendación Final - Charts: Mantener **Victory** por ahora. Dado que ya está en uso, evitamos migración cost (Formidable invests in it, so it's not abandonware [74]). Victory cubre las necesidades identificadas: barras para lead volume, líneas para tendencias, pies for breakdowns. Es compatible TS e integrado con React lifecycles de transición. Además, la personalización estilística se alinea con nuestra app (podemos aplicar colores de Tailwind theme fácilmente). Por ejemplo, podemos crear un theme para Victory con nuestros colores corporate. Migrar a otra lib aportaría similar funcionalidad con potenciales quebrantos. Recharts es tentador por su popularity, pero como ya dominamos Victory en código existente, seguimos con ella y reevaluamos tras MVP si hay limitaciones (Victory vs Recharts performance in heavy analytics – a ser medido, pero improbable ser bottleneck en MVP scale).

C) Chat UI + Realtime (componentes de chat y virtualización de mensajes):

1. **Chatscope Chat UI Kit (React)** – *MIT*, 1.7k stars [79]. Es un conjunto completo de componentes estilizados para chat: ChatContainer, MessageList, Message, TypingIndicator, Input, etc. **Pros:** Rápido de implementar una interfaz de chat profesional sin diseñar todo. Su estilo es neutral/moderno y personalizable via CSS. Incluye scrolling virtuoso (ellos mencionan virtualization). TS support: sí, escrito en TS. Documentación a través de Storybook con ejemplos [80]. **Contras:** Opinión visual – aunque se puede override CSS, viene con su propio layout que puede o no encajar con Tailwind easily (pero podemos encapsular su container). No tiene conocimiento de nuestra state, es solo UI – pero eso está bien.

2. *Riesgo:* Mantenimiento dependiente de chatscope developer (antoniobermuda, responded on Reddit 3y ago), la actividad no es enorme (110 commits) pero issues hay movimiento.

3. Sin embargo, chat UI doesn't change often, so it might be fine as is.

4. **React Virtuoso (for chat lists)** – *MIT*, ~3.6k stars (petyosi)**. Es una librería de** virtualización **de listas optimizada, con soporte explícito para chat (inverted scrolling, maintain scroll position on new messages)** [81] **.** Pros: **Excelente performance con muchos mensajes, fácil integration (wrap <Virtuoso> around message components). It handles dynamic heights.** Contras:** No provee UI style, solo virtualization logic – pero esto es fine. TS support: good.

5. *Para un chat*, combinar Virtuoso (scrolling) + UI either custom or from kit.

6. Possibly redundant if chatscope has virtualization built-in (docs unclear; but likely uses scroll Into View).

7. **Stream Chat React (Component library)** – *License mix: Stream's own, free up to certain usage but not OSS fully.* We skip due license.

8. Also **Sendbird UIKit** similar – closed/premium.

Otras: **Voiceflow react-chat** (likely MIT, tailored for conversational AI, maybe overkill?). **Shadcn's example chat** – shadcn made an AI chat UI using Radix + Tailwind [82], could serve as reference for custom building. - Actually, building a simple chat UI ourselves using Radix ScrollArea + our own message bubbles is possible, but time-consuming to polish (scroll retention, etc. have tricky edge cases). - If chatscope kit works, it saves time.

Recomendación Final - Chat UI: Utilizar **Chatscope Chat UI Kit** combinado con **React Virtuoso** para robustez. Con chatscope, obtenemos un marco de interfaz (burbujas de chat, lista con auto-scroll, input area) que luce bien por defecto [80], y podemos ajustarle el theme CSS para que coincida con Tailwind colors (incluso podrían envolver con Tailwind classes). Su licencia MIT y modularidad son apropiadas. Para garantizar performance en historiales largos, integraremos Virtuoso (chatscope docs indican cómo usar custom scroll, o podríamos manualmente componer Chat + Virtuoso MessageList). Esta combinación nos da: - Rápida implementación UI (minimizando CSS custom), - Eficiencia en DOM (no renderizar 1000 mensajes a la vez).

Adicional: Implementaremos real-time con probablemente **Socket.IO** (MIT) en backend, pero en frontend usaríamos el cliente oficial (MIT). O si usamos Supabase (Apache) for real-time, su JS library. De cualquier forma, para nosotros lo relevante es que la UI soporte updates. Chatscope kit no impide usar cualquier state management para messages.

D) Calendario / Scheduling (UI de calendario):

1. **FullCalendar (React)** – *MIT (core)* [83] . Librería veterana con vistas **mes, semana, día, agenda** y features: drag events, resizing, recurring events (lo básico es MIT, funcionalidades avanzadas como Timeline resource view son parte de premium plugin pero no necesitamos). **Pros:** Muy completa, ya soporta drag & drop de eventos y interactions, localizable, ampliamente usada. Tiene un wrapper for React that lets using JSX for event content [83] . **Contras:** Estilos predefinidos no Tailwind (pero hay themes, e.g. a bootstrap-like theme or we can override CSS). Some learning to configure. The React component is mostly an integration layer to the vanilla JS FullCalendar. TS support: via @fullcalendar packages (written in TS internally).

2. Performance wise, handles decently up to hundreds of events.
3. Large bundle though (couple hundred KB if include all views).

4. But we might not have alternative if want all those interactions ready.

5. **React Big Calendar** (rbc) – *MIT*, 8.6k stars [84] . A React-native calendar component (not RN, but React built). Provides **Month, Week, Day, Agenda** views similar a Outlook/Google Calendar, supports drag and drop (requires an addon with react-dnd). **Pros:** Pure React, so easier to style via CSS (it uses flexbox, you can override class names, someone even made a Shadcn style for RBC [85] !). Has localizer system for date (moment, date-fns etc.). It's widely used for scheduling apps. **Contras:** The default look is somewhat plain. Setting up drag and drop requires adding backend for DnD context. Also, the project though old is still maintained (last commit 2023, v1.10). TS: It has types but not sure if fully converted to TS code, likely not fully but definitely has .d.ts, and community knowledge exists.

6. RBC might integrate more smoothly with our stack because we can apply Tailwind classes if we control components rendering events (their docs say you can customize event rendering).
7. RBC has no premium version, everything open.

8. If we recall, RBC came from jquense (who also wrote React Router and others, reliable dev).

9. **Schedule X** (ilamy's calendar) – *MIT*, relatively new (100+ stars, just reached v1.0) [86] . Built with Shadcn components and Tailwind, offering multiple views, drag & drop, recurrence, etc., basically what RBC/FullCalendar do but modern. **Pros:** It's designed to integrate easily with Tailwind/shadcn (likely classes out-of-box), and written in TS. Could be easier to customize to our UI style. Has resource calendar now too. **Contras:** New project, minimal adoption, possible bugs. Could lack some polish vs older libs. But developer seems active and excited.

10. Given it's v1.0 and they've specifically marketed it for modern stack, it might be a dark horse.
11. However, trusting a < 150 stars library for a critical component might be risky; if issues arise, less community support.

Otras: **DevExtreme React Scheduler** (MIT by DevExpress) – integrated with MUI, not desired due external heavy dependency, and maybe not pure MIT for everything. **React Calendar** by dhtmlx & others are paid. **Kalend** (MIT) – an open source calendar with day/3-day/week views (found on npm: 455kB, last updated 2022, ~200 stars). Could consider but RBC seems more proven.

Recomendación Final - Calendario: React Big Calendar (RBC) es la opción recomendada. Motivación: - Familiar UI pattern (like Google Cal) que los usuarios reconocen. - Componente maduro, MIT, sin funcionalidades bloqueadas. - Integrable con Tailwind (podemos override CSS or find that Shadcn preset that was made [85]). - Soporta drag & drop para reprogramar nativamente (con a small setup). - TS support adequate (we might need some @types but fine). - FullCalendar es excelente pero integrarlo en React adds complexity (it's essentially an imperative lib under the hood). RBC siendo puramente React encaja mejor con our state approach (we manage events as state arrays, pass to RBC). - RBC's performance for our scale (tens of events per week) is more than enough. - RBC localizer easily ties to date-fns (we prefer not to include moment to keep bundle smaller, date-fns is tree-shakable).

Nos aseguraremos de probar RBC dragging con our use case (should allow moving an Appointment event easily). RBC's license MIT ensures no legal strings. We'll keep an eye on Schedule X (ilamy's) as it evolves; si RBC styling resultara muy costoso, podríamos pivot más adelante, pero por MVP vamos a lo seguro.

E) Forms / Validation:

1. **React Hook Form (RHF)** – *MIT*, ~44k stars (muy popular) [87] . Forms manejados con hooks sin re-render excesivo, altamente performant (usa refs). **Pros:** Muy flexible (controla campos no controlados, integración con UI libs), excelente performance con grandes forms. TS support robusto (Generics for form values). Talla muy pequeña (~9kb). Gran ecosistema: resolvers para integrarse con **Zod**, **Yup**, etc. **Contras:** API puede parecer más bajo nivel comparado a Formik (no automágica two-way binding, se hace register() manual). Pero su approach imperativo is a plus for performance. Documentation es buena.
2. RHF also easily handles dynamic fields, field arrays, etc.
3. A11y: It doesn't do built-in UI, so that's on us (but easier to manage with our design system).
4. Maintained activamente (v7 stable, v8 rumored in future).
5. **Formik** – *MIT*, ~32k stars. Form management via state and context, popular historically. **Pros:** Intuitivo para principiantes (like formik <Formik initialValues> wrap, use handleChange, etc.), good docs. Has an ecosystem (Formik UI, etc.). **Contras:** Performance: re-renders the form on every change by default, can be heavy in large forms. Newer patterns (like RHF) have overtaken it. TS support: Formik v3 might be fully TS but current v2 has types. It's fine but not as DX friendly in TS as RHF+Zod in terms of type inference for values.
6. We likely moved away from Formik due performance.
7. Not needed since our forms aren't extremely complex, but we anticipate possibly large forms (listing form, multi-step), RHF scales better.
8. **React Hook Form + Zod (combination)** – Actually RHF itself doesn't validate, it leaves validation to either HTML validation or integrates with schema libs via resolvers. The recommended approach nowadays: define a schema with **Zod** and use `@hookform/resolvers/zod` to validate on change/submit.
9. **Zod** – *MIT*, ~23k stars. Typescript-first schema validation [88] . **Pros:** define schema in TS, infer type automatically. Great DX and error message structure. We can use it for forms (synch or as RHF resolver). Also can use same schemas on backend if needed.
10. Versus **Yup** – MIT, older, not TS-first (infers types but not as clean, and doesn't catch certain issues at compile time like Zod can).
11. Zod is actively maintained, widely adopted in TS community, so prefer it.
12. Another plus: we can define our data models interface and Zod schema together to ensure consistency.

Other: **Final Form** (MIT, by Farcic) – good but Formik and RHF overshadow it now. **Vest** – somewhat new, not widely used. **Zod vs Yup**: We lean Zod for reasons above.

Recomendación Final - Forms: Usar **React Hook Form** para manejo de estado de formularios + **Zod** para validación de esquemas. Rationale: - RHF's performance and flexible control align with our need for snappy UI (no lag in large forms or on mobile). - The integration (via `hookform/resolvers`) is straightforward – example:

```
const { register, handleSubmit, formState: { errors } } = useForm({
resolver: zodResolver(MySchema) });
```

Then RHF will run Zod validation on submission or blur, and populate `errors`. - We can then easily display errors near fields (which we will ensure to do for accessibility). - Zod's messages can be customized to Spanish for user-facing if needed (maybe using Zod's `.describe` or custom error map). - This combo is well documented, widely used (ensuring we can find solutions if stuck), and fully OSS MIT. - Additional: For complex nested forms (like perhaps dynamic fields for routing rules), RHF provides useFieldArray, etc., which is robust.

We also consider using **react-hook-form +** `zod` beyond forms – e.g. in data model validation in some state machines. But primarily forms.

We'll avoid Formik due to performance and older pattern.

F) Notifications UI (toasts & center):

1. **Radix UI Toast + Sonner (Shadcn)**:
2. Radix Primitives incluye `Toast` primitives [89], pero Shadcn (the component library we align with) sugiere usar **Sonner** in place [90].
3. **Sonner** – *MIT*, ~11.9k stars (trending) [91]. Es una librería de toasts minimalista pero con muy buenas animaciones y diseño (opinionated styling, but looks nice – modern Apple-like).
4. Importante: Sonner provee `<Toaster>` container y una simple función `toast(<Content>)` global usage, similar a React-hot-toast. Shadcn incluso ofrece un ready integration [92].
5. Pros: Muy fácil de usar; animaciones fluidas; soporta swipe para cerrar, cuenta regresiva, etc. TS support: yes (written in TS [93]).
6. Contras: "Opinionated" means limited custom theming (but we can likely override CSS variables or at least it's pretty in line with modern design).

7. Sonner cubre toasts. Para un **notification center** (list of persistent notifs) nosotros implementaremos panel manual, no library off-shelf does exactly that integrated (we can simply list from our Notification[] state).

8. **React Hot Toast** – *MIT*, ~10.8k stars [94]. Already widely used. Similar API to Sonner: global <Toaster> and `toast()` calls. **Pros:** Very customizable in content (you can render any JSX in toast), supports promise toasts (loading -> success). Lightweight (~5kb). TS: included types. **Contras:** Look is decent out-of-box but might need theming to match (some default style might conflict with shadcn, but can be overridden).

9. It's battle-tested (600k uses on GH [95]).
10. We used it previously possibly.

11. Sonner basically emerged as a competitor with slightly more refined animation by default, but either is fine.

12. **React-Toastify** – *MIT*, ~21k stars. An older library that attaches toast container and offers a bit less modern look by default. **Pros:** Very configurable positions, etc., robust. **Contras:** Larger (~16kb), styles not as elegant, and not tree-shakable (pulls CSS global).

13. We lean to the newer generation libs (HotToast/Sonner) for better DX.

Also: **Notistack** (for Material UI) – irrelevant since we are not using MUI.

Recomendación Final - Notifications: Adopt **Sonner** for toast notifications. - Reason: It aligns with Shadcn's recommendation (the Shadcn documentation explicitly has replaced their toast component with Sonner [90]). This means if we've been following shadcn patterns, Sonner is likely to fit in seamlessly. - It comes with built-in accessibility (ARIA live regions), queue management, and nice animations (swipe to dismiss, etc.) that Radix Toast would require us to implement from primitives. - We can easily call `toast("message")` anywhere, which is great for our use-cases (feedback on save, etc.). - If needed, it supports different types (success, error with icons) easily, and can render custom JSX (like a <button> in toast). - We'll put `<Toaster richColors />` (richColors gives colored background, presumably) at root app as per docs, then use toast calls.

For the **in-app Notification Center** (bell icon with list): - There's no off-the-shelf library because it's quite app-specific. We'll implement our own small panel, likely using Radix Dialog or Popover for the dropdown and map our notifications state. - We will ensure to style read vs unread differently (maybe bold vs normal). - Possibly use a thin library like **Headless UI's Menu** or since we have Radix, a `Popover` + `Command.List` could list notifs, or just a simple `ul`. - Mark all as read button, etc., we handle manually. It's straightforward enough not to need a specialized library.

G) Micro-interactions / Motion libs:

(This is broad: covers gesture, animation libraries beyond framer-motion, and possibly Lottie or confetti)

1. **Framer Motion** – *MIT*, we already have it. It's a powerful animation library for React. We will **continue usando Framer Motion** as primary for orchestrating UI animations (as per motion spec). For micro-interactions:
2. Framer Motion covers most needs (variants, gestures like drag, layout transitions).
3. Also, **React-use-gesture / @use-gesture** (MIT) if we needed advanced gesture recognition (like swipes not already covered by Radix or Sonner). But likely not needed since Radix Toast covers swipe, and DnD kit covers drag.
4. **Dnd Kit** (MIT) – for drag and drop interactions (especially pipeline Kanban). It's modern, TS, and no heavy dependencies. We will likely use DndKit (or react-beautiful-dnd which is older but if needed for Kanban – but that one is not actively maintained by Atlassian since they archived it). DndKit is well-regarded now.

5. DndKit provides sensors for mouse/touch, and has good docs for draggable + droppable contexts. We'll use it to implement lead cards drag between columns.

6. **Lottie (for animations)** – *MIT*, widely used for vector animations exported from After Effects. If we want to include any fancy illustrative animations (confetti explosion or some mascot moving), Lottie is best.

7. We would use the official Lottie Web or Lottie React player (both MIT [96]).

8. For confetti specifically, perhaps simpler to use a confetti library.

9. **Confetti libraries:**

10. **canvas-confetti** – *MIT*, a tiny lib to do confetti using canvas. No react specifically, but can just call it in a useEffect on trigger. Very popular (like 7k stars).

11. **React Confetti** – MIT wrapper that draws confetti on canvas full-screen. Could use, but canvas-confetti might suffice as it's straightforward.
12. There's also **tsParticles** or others but too heavy.

13. I'd go with canvas-confetti because it's trivial to fire (just call confetti() with config).

14. **GSAP** – not truly open source (GreenSock license is custom, free for non-sellable web apps but might conflict if we distribute software). Also heavy, we probably avoid since Framer covers our needs.

15. For a quick occasional tween or complex timeline, maybe but I'd not bring GSAP to a TS React project with already Framer in place.

16. **React Spring** – (MIT) alternative animation lib using springs. Possibly used by Nivo internally, but we don't need it separate if using framer.

17. Could be used for some gestures maybe but Framer's spring physics suffice.

Recomendación Final - Micro-interactions libs: - Seguimos con **Framer Motion** para animaciones generales y we will leverage it for micro interactions (spring on button press, etc.) as needed, integrated with Tailwind classes for initial states, etc. - Para **drag & drop** (p.ej. Kanban pipeline, reordering tasks): **Dnd Kit** (MIT) es la elección. Modern, modular, and we can style draggables freely. It's better than the old React DnD or Beautiful DnD for latest React. - Para **confetti/micro-animations especiales**: - **canvas-confetti** for confetti bursts (super small and straightforward). - **Lottie React** for any custom AfterEffects animation (like maybe a subtle trophy animation when agent reaches 100% profile). - For gestures like **swipe to action** (if not covered by Radix or Sonner), we can consider **use-gesture** + Framer Motion, but likely not needed since limited use-case. If needed, DndKit can also detect pointer events for swiping if configured.

We note licenses: - Framer Motion MIT, - Dnd Kit MIT, - canvas-confetti MIT, - Lottie MIT. So all good.

H) Command Palette + Hotkeys:

1. **cmdk (React Command Palette)** – *MIT*, ~12.2k stars [97] . Highly regarded, used by Vercel etc. **Pros:** Very fast, accessible (combobox semantics) [58] , and headless/un-styled so we can style it with our Tailwind or use one of their example stylings. It's composable: we define <Command> container, <Command.Input>, <Command.Item> etc., and it handles filtering and keyboard nav automatically [98] [99] . TS: yes with generics. Active project by esteemed dev (paco).
2. We saw in previous research that Shadcn integrated or examples exist, plus reddit praises it.
3. It's minimal overhead (4kb).

4. We will likely use `cmdk` to implement our palette because of flexibility to define groups, items. Also can re-use for things like a Spotlight search for leads maybe.

5. **kbar** – *MIT*, ~5.1k stars. Another popular command palette lib. **Pros:** Also easy to setup (wrap <KBarProvider> and define actions). It has default UI but can be customized. It supports nested actions and even a simple search by default. **Cons:** Slightly more opinionated styling (though you can override, but it's less headless than cmdk).

6. Kbar had hype a year or two ago, but many have switched to cmdk for more control and possibly better performance (some claimed kbar had overhead).

7. Also kbar might not handle huge lists as well as cmdk does with virtualization (not sure).

8. We'll lean cmdk since it appears more flexible and trending.

9. **React Hotkeys (for global shortcuts)** – There is **react-hotkeys-hook** (MIT) or we can use **tinykeys** (3.9k stars, ~1kb, good for global key listeners).

10. Considering command palette will cover many nav actions, we still want some direct shortcuts:
    - Could implement via our own `useEffect` for keydown (like example hooking 'g + i' etc.), but a hook library might make combos easier.
11. **react-hotkeys-hook** by JohannesKlauss (MIT) – 3.4k stars 100 . Provides `useHotkeys` hook.
12. Or **tinykeys** (by Jamis Charles) – ~4k stars, no dependencies, just a function to bind combos to handlers easily. Not React-specific.
13. Maybe we use **tinykeys** combined with React useEffect. Since it's small, might be fine.

But for simplicity, we might not need an external library for a handful of global shortcuts. - Could just do:

```
useEffect(() => {
  const handler = (e) => { if(e.ctrlKey && e.key === 'k')
openPalette(); ... }
  window.addEventListener('keydown', handler);
  return () => window.removeEventListener('keydown', handler);
}, []);
```

- For multiple combos, a lib helps to parse combos like "g+i" elegantly.

Recomendación Final - Command Palette & Hotkeys: - Emplear **cmdk** para implementar la paleta `Ctrl+K`. Nos da base accesible y rápida, y podemos style it to match (Shadcn likely has an example style to copy). - Para **hotkeys globales adicionales**, usaremos ya sea un hook minimal: - Posible plan: integrate them into the command palette as well (some palettes allow adding hidden actions for shortcuts only, but might complicate). Simpler: we create a small `useGlobalHotkeys` hook using either `react-hotkeys-hook` or manual. - Leaning to just use **react-hotkeys-hook** because it's lightweight and straightforward for a few combos. Ex:

```
useHotkeys('g i', () => navigate('/agents/inbox'));
```

That library handles modifiers and prevents conflicts with input fields etc. - If `react-hotkeys-hook` causes any issues, fallback to manual with tinykeys. - This isn't a user-facing lib so we can decide at implementation. But likely use `react-hotkeys-hook` (MIT).

All these libraries comply with our license criteria (MIT, no AGPL). We'll mention any that are borderline: - GSAP is not included (license not MIT), so skip. - Others chosen are MIT or similar.

10. Plan de implementación en el repositorio

Para empezar la construcción en la branch `Feature-Page-DashboardAgents-28-01-2026`, seguiremos estos pasos:

Estructura de carpetas/proyecto:

Asumimos en nuestro repo actual existe `src/pages` para page-level components, `src/components` para UI reusables, `src/lib` para lógica utilitaria, etc., y alias `@/*` mapeado a `src/*`.

Propuesta: - Crear **src/pages/agents/**: aquí viven las páginas/route components específicas del portal de agentes: - `index.tsx` (para `/agents` overview dashboard), - `inbox.tsx`, - `leads.tsx`, - `calendar.tsx`, - `listings.tsx`, - `credits.tsx`, - `team.tsx`, - `reports.tsx`, - `settings.tsx`. - Podrían ser React components por ruta, export default, usando data fetching hooks internos. - Además sub-rutas si se usa anidación (ej. `src/pages/agents/inbox/[leadId].tsx` si implementamos lead detail page). - Crear **src/pages/agents/layout.tsx**: Un layout compartido para todas las sub-rutas de /agents. Este compondrá la estructura con sidebar y header. (Si usamos Next.js App Router, layout.tsx es apropiado; si CRA/React Router, implementamos <AgentLayout> wrapper). - En React Router, definiremos rutas protegidas: e.g. in router config:

```
<Route path="/agents" element={<AgentLayout />}>
  <Route index element={<OverviewPage />} />
  <Route path="inbox" element={<InboxPage />} />
  ... etc
</Route>
```

Y antes se verifica autenticación/rol (ver más abajo Guards). - **src/components/agents/**: UI components específicos al portal: - Ej: `LeadCard.tsx` (para card en pipeline o list), - `LeadKanbanColumn.tsx`, - `ChatWindow.tsx` (maybe wrapper around chat kit), - `TaskItem.tsx`, - `TeamMemberRow.tsx`, etc. - Además modales/dialogs e.g. `ScheduleAppointmentDialog.tsx`. - **src/components/ui/**: Podríamos tener los generics (but likely we have shadcn's in place). - Already might have `ui/Modal, ui/Dropdown` from shadcn. We'll reuse those for consistency. - **src/lib/agents/**: - `api.ts` for agent-specific API calls (wrappers for fetch to endpoints like getLeads, updateLead etc.). Initially, can use mock data or fetch from dev API. - `store.ts` or context for global state if needed (like useAgentsStore for notifications or leads caching). - Possibly a `useAgentAuth` hook if needed. - **src/types/agents.ts**: Consolidate the interfaces from above (Agent, Lead, etc.) for use across pages and to ensure consistency. Could also separate by domain (e.g. types for lead, listing, etc.). But one file manageable enough for now.

Routing y Layout anidado:

Implementaremos un guard antes de /agents: - Si el usuario no está autenticado o no es agente, redirigir fuera (login or show 403). - Ubicación del guard: Dependiendo de router (Next App Router uses middleware or server checks; React Router can do in element or a wrapper route that checks context). - Simplicity: We can make an `<AgentLayout>` component that on mount verifies `user && user.role in ['Agent','Broker']`. If fails, do `navigate('/login')` or similar. - Or at route definition, wrap in a component: `<Route path="/agents/*" element={<RequireAgent><AgentLayout/></RequireAgent>} />`.

The AgentLayout component: - Renders the sidebar with nav links (Inbox, Leads, … icons). - Possibly obtains some data (like current agent info for profile picture in topbar). - Renders an <Outlet/> for child routes content.

We'll also integrate the **command palette** and **toaster** at this layout level (so available in all subpages). E.g. include `<CommandPalette/>` component and Sonner's `<Toaster/>` in layout.

Strategy for Mock Data & Dev without backend:

Given we likely don't have all backend endpoints ready, propose using **Mock Service Worker (MSW)** in dev: - MSW (MIT) allows intercepting network calls and returning fake data as per our type definitions. - We'll create a `src/mocks/agentsHandlers.ts` defining handlers for e.g. GET /api/agents/leads (return a fixture list of leads), etc., and use MSW in dev environment to intercept fetch calls. This way, we can integrate UI with "fake real" calls. - Alternatively, we can create local static JSON for initial UI work. - If the repository uses a certain dev API or local environment, we adapt accordingly.

But MSW is nice to simulate various states (no leads, some leads, error responses). We should document how to turn it off for prod.

Parallel Work approach:

The specification suggests the output should allow multiple devs to implement in parallel (UI + logic + mocks simultaneously). To facilitate that, we can: - Divide tasks by route. Eg. Dev A works on Inbox & Chat, Dev B on Calendar & tasks, Dev C on Listings & credits, etc., using the same structures. - Ensure design tokens (colors, etc.) and component patterns are decided early (which likely in our case, we rely on existing Tailwind config and shadcn components). - Create reusable components like LeadCard early so they can be used in both inbox list and pipeline.

Checklist for each PR or feature: - *Empty states:* Each page should handle no data scenario gracefully with an empty state UI (with maybe an illustration, per our motion spec a subtle animation). Example: If no leads, Inbox should show "No leads yet" rather than blank table. - *Loading states:* While data fetching (if not SSR), show skeleton or spinner. We'll likely use skeletons as per our motion guidelines. For MSW, we can add artificial delay to test skeleton appearance. - *Error states:* If fetch fails, show error message with retry option. Possibly toast an error. - *Accessibility (A11y):* - All interactive elements must be reachable via keyboard (we rely on Radix/shadcn for base components like modals, they handle focus traps). - Use appropriate ARIA roles (like role="alert" for toasts, aria-live for notifications if needed, headings, labels on form fields, etc.). - Ensure text contrast meets guidelines (Tailwind default on our design likely fine). - Each page should have a primary <h1> (maybe the topbar can have the title or we include in content). - Command Palette from cmdk is already accessible as combobox with ARIA as per their docs. - Manage skip links perhaps or at least allow quick nav (but small scope, skip content maybe not needed if main content is first anyway). - *Keyboard navigation:* - e.g. In leads pipeline, allow using left/right arrow to move focus between columns, and up/down within a column, and maybe Enter to open detail or trigger drag. Not trivial, can be later improvement but at least basic tabbing around content should be possible. - We'll implement global shortcuts (like those `g + ...`) as planned. - *Prefers-reduced-motion:* - Test by enabling that setting in dev tools. Ensure modals etc. still functional (should be, if using Radix, they handle disabling animations). - For our custom animations via Framer, we can use `useReducedMotion()` hook to conditionally turn off variants or set duration 0. We'll incorporate that in key places (like confetti won't run). - *Testing:* - We'll write basic unit tests or integration tests as time permits. Key logic like lead state changes (maybe state machine if implemented as pure functions) could have unit tests. - Components testing: e.g. using React Testing Library to mount <InboxPage/> and verify it shows a lead from mock, that clicking leads open chat, etc. Might stub out network via MSW in tests as well (cool). - Also test important flows manually: - Accept new lead and see it moves in UI. - Drag pipeline card and see toast appear, etc.

Given time, cover at least critical interactions (drag-n-drop events, sending message optimistic UI, etc.) with tests to prevent regressions.

Step-by-step plan (initial days):

1. **Scaffold routes & layout:** Create pages structure and AgentLayout with dummy links. Protect with a placeholder auth check.
2. This yields a skeleton UI to navigate sections (no content yet, but layout visible).
3. **Implement basic Inbox list & Lead detail panel:**
4. Hardcode some sample leads in state (or use MSW to return them).
5. Use components (maybe start implementing LeadCard and a simple list).
6. For chat, integrate Chat kit with static messages to see it renders.
7. Mark up detail panel with fields like property, contact info, follow-up recommendations stub.
8. **Implement Pipeline (Leads page):** Possibly after having some lead card component from inbox, adapt it into a draggable Kanban using DndKit.
9. Setup Dnd contexts and test moving a card updates state.
10. Connect with a toast on drop (like "Lead moved to Contacted").
11. Keep logic simple at first (no real API calls, just state).
12. **Calendar page:**
13. Integrate RBC or chosen calendar.
14. Show some dummy events (from leads state perhaps).
15. Test drag to move events triggers a console log (later to update state).
16. Add modal for scheduling new (just form with no submission logic at first).
17. **Listings page:**
18. Display a table or list of listings (maybe use TanStack Table headless to test it, or simpler, markup static).
19. Include an "Add listing" button to open a form (scaffold multi-step form or one long form using RHF+Zod minimal schema).
20. For now, form can just console.log input.
21. Provide edit inline maybe or skip at first.
22. **Credits page:**
23. Show balance (hardcode a number).
24. Show a table of ledger entries (try using TanStack Table or simple map).
25. Add a "Buy credits" button: open modal with pretend packages (not hooking actual payments yet). On confirm, update balance in state and show toast.
26. **Team page:**
27. If in context of a team (maybe we decide to always show for certain roles), show list of members (simulate with current user plus maybe one more).
28. Provide a form to "Invite agent" (just UI for now).
29. Show toggles for pause (change some state).
30. If time: a subcomponent for routing rules with a simple list (non-functional demonstration).
31. **Reports page:**
32. Use Victory charts. E.g., a line chart for leads per month, a pie for lead types, etc., using dummy data.
33. Show a table for team performance if team (use TanStack Table grouping by agent).
34. Ensure charts appear nice with theme.
35. **Settings page:**
36. Provide sections: Profile (with form), Notifications (with toggles as per our triggers design), maybe Integrations (dotloop placeholder).
37. Use RHF+Zod for the profile form (validate email, etc.).
38. Possibly allow file upload for profile picture (we can use a file input and preview).

39. Notifications toggles hooking into some global context for preferences (store in local storage in absence of backend).

40. **Global enhancements:**

    - Add Sonner Toaster in Layout; use it in places: e.g., after moving lead stage, after sending message, after error occurs, after saving profile.
    - Add Command Palette (cmdk):
        - define list of commands (like "Go to Inbox", "New Task for [lead]" etc.). For now, ensure "Go to [section]" commands and open with Ctrl+K working.
        - Possibly integrate search: "Search Leads: John" to filter leads (maybe not immediate, can have as future).
    - Add global hotkeys:
        - e.g., implement 'g' then 'i' sequence using a state: maybe simpler is single combos like 'Shift+I' for inbox direct. But if we want sequence, might need more logic or skip sequence.
        - At least implement "Ctrl+K" in case the built-in in cmdk might need manual if not automatically binding.
        - Also "?" to open a help modal listing shortcuts.
    - Add MSW for API endpoints:
        - E.g., `GET /api/leads` returns our dummy leads list.
        - Use that in useEffect of Inbox to fill state.
        - Setup MSW to run in dev (maybe already configured in repo if they used it; if not, integrate).
    - Loading states:
        - e.g., in leads list, show skeleton while fetching; in chat conversation, show spinner if sending message, etc.
        - Use react-loading-skeleton or shimmer CSS. Possibly create a <Skeleton> component if shadcn has one (shadcn does have a Skeleton component).
    - Verify all flows manual: simulate new lead: add one to list and see UI; simulate message: push onto conversation.
        - Possibly implement a small dev-only button to add a fake lead or trigger to test triggers.

41. **Agent Score calculation** (if time):

    - On Reports, show an "Agent Score" card derived from some formula on the dummy data (like 90/100).
    - Present it positively (with maybe star icon).
    - But if time short, skip actual calc, just static.

42. **Final checks and refine:**

    - Cross-check all acceptance criteria from spec (like optimistic UI points, micro animations).
    - E.g., ensure drag lead triggers a small motion (we can use framer on dragging if needed).
    - Ensure any AI/next best suggestions we wanted to display somewhere are at least stubbed (maybe on Overview a "tip of the day" which is static text).
    - Clean up any console.log, ensure error handling messages are user-friendly.
    - Write test cases for critical logic: state machine transitions (we can implement a pure function updateLeadStatus(lead, newStatus) that returns updated lead or error if invalid transition; test that).

- Possibly test the command palette filter logic with a couple commands ensure it finds relevant.

Parallelization Potential: - Layout & Navigation (one dev). - Inbox/Leads/Chat (one dev). - Calendar & Tasks (one dev). - Listings & Credits (one dev). - Team & Settings & Reports (one dev, as these are smaller pages each). - This needs good coordination (shared types in agents.ts, and maybe dummy data structure in context to not duplicate). - Setup an **AgentContext** with initial data (like an object containing leads, tasks, etc.) so multiple components can access and update. Alternatively, use something like Zustand or Redux if needed, but might be overkill. Possibly a simple useReducer at AgentLayout level passing down via context is enough for MVP state management.

Focus on shipping MVP quick: we might use simpler static data for initial UI and integrate real API calls gradually.

PR Checklist (embedding what was mentioned): - Did we cover empty state? (Check by clearing dummy list). - Does every interactive element have text label or aria-label (for icons like the bell, we must add aria-label "Notifications"). - Are modals/dialogs properly focus-trapped and closable via Esc? (Radix ensures it, test manually). - `prefers-reduced-motion`: simulate and ensure e.g. our confetti doesn't run: we can check by turning on the setting in OS or devtools, or using the hook's boolean. - Code quality: lint, type-check. Add JSDoc or comments for complex logic (like lead drag handler). - Testing: run unit tests, perhaps cypress e2e if available to simulate a quick scenario (if time). - Documentation for next dev: maybe update README or a Notion with how triggers are implemented or how to toggle features if needed.

11. Reporting, Métricas y "Score del agente"

Definimos métricas clave que el agente podrá ver en su portal (y que nosotros mediremos para evaluar uso):

KPIs visibles al agente (en Reports/Overview): - **Tiempo de respuesta promedio:** (o tasa de respuesta en X minutos). Ej: "Tiempo medio de primera respuesta: 5 min". Zillow midió "answer rate" en % [41], pero podríamos mostrar % de leads responded within 5m. Presentación: un texto o gauge "Respuesta Rápida: 90% (por encima de la media)". - **Conversión a cita:** de todos los leads recibidos, qué porcentaje se convirtieron en al menos una cita. "Conversion a visita: 40%". - **Tasa de no-show:** porcentaje de citas donde el cliente no se presentó. (Queremos minimizarlo; si es alto, indica potencial mala calificación). - **Leads activos vs. inactivos:** cuántos leads están en pipeline todavía (no cerrados ni perdidos), para que sepa su workload. Ej: "Leads activos: 8". - **Desempeño por listing:** para listing agents, puede ser "Leads generados por cada listing" o "Promedio de views por listing vs promedio zona" (eso puede ser en listing detail more). - **Tiempo desde lead a cierre:** (si trackeamos Closed date) – podría mostrarse como "Ciclo de venta medio: 45 días". - **Tasa de cierre:** leads cerrados / leads totales (excluyendo lost?). - **Porcentaje de perfil completado:** visible quizás en Overview as progress bar.

Muchos de estos KPIs alimentan un **Agent Health Score**: - **Agent Health Score**: Podríamos definirlo como un índice 0-100 calculado de: - 40% Customer Experience (feedback, if any), - 30% Performance (conversion rates, response time), - 30% Operational (profile completeness, usage of tools).

De momento, sin encuestas reales, nos basaríamos en: - Respuesta rápida: si >90% leads responded quickly, good. - No-show bajo: good. - Conversion lead->cita y cita->cierre: measure ability to close. - Completar perfil y usar tasks etc. Score calculado podría ser simple sum: - e.g. ResponseTimeScore (0-30), ConversionScore (0-30), EngagementScore (0-20, tasks completed etc.), ProfileScore (0-20). - O definimos rubricas fijas: - Responde en <1h -> +20, < 3h -> +10, >3h -> 0 points. - Converted >30% leads

to closing -> +20, 10-30% -> +10, <10% -> 0. - etc. Transparent explanation: En Reports, un panel "Agent Score: 75 (Bueno)" with breakdown showing how it's calculated: e.g. "Respuesta 90% (excelente), Conversión 15% (promedio), Perfil 80% (casi listo)..." so agent knows what to improve.

- We should avoid frustration by:
- Not calling it "score out of 100" in a way that feels like a grade. Perhaps name it "Agent Performance Index" or a badge level (like Bronze/Silver/Gold).
- Emphasize improvement: provide tips if score low, not just number.
- Do not show it on top all the time, maybe hide in Reports behind a click "Ver detalles".
- Possibly if we incorporate *Best of Zillow*-like status, we give badges if score above threshold.

Métricas ocultas (para sistema, no necesariamente mostradas): - We will track in our analytics: - Daily Active Users (how many agents log in daily), - Frequency of returning in a day (multi-session), - Feature usage (e.g. % agents using tasks feature, using pipeline vs list, responding in web vs offline). - But those are internal.

Mostrar métricas sin frustrar: - Provide context and achievable goals: - If an agent's response time is slower, instead of red shame, show maybe an average or suggestion: "Tu tiempo medio es 5h, clientes esperan <1h; intenta responder más rápido para mejorar conversión." (Encouraging tone). - If conversion is low, "Has cerrado 5 de 50 leads. Te recomendamos revisar la calidad de leads o seguir nuestras sugerencias para mejorar seguimiento." - Possibly add gamification: e.g. "¡Completaste 80% de tu perfil, casi listo para el badge de Perfil Perfecto!" - Show benchmarks: "Los agentes top responden en <1h, tu promedio es 2h. ¡Estás cerca!" - Provide positivity: if agent improves, highlight it ("Mejoraste tu tiempo de respuesta esta semana, ¡bien hecho!").

Thus, we give metrics as guidance, not as judgment.

Best of Zillow style feedback: - Without actual surveys, we might integrate something like after a successful closing, we could simulate sending a feedback request and if we get a rating (like via manual entry or something). But for MVP, skip actual external feedback, maybe present a placeholder "Customer Satisfaction: 4.5/5 (en base a reseñas)". That would link to reviews page.

Agent Score Display: - Perhaps on Overview as a small card with either a star rating or a gauge. But to avoid negativity, maybe do a tier system: "Nivel: Avanzado" or "Estás en el top 10% de agentes en respuesta". - If we do numeric, definitely show "75/100 (Bueno)" rather than just "75".

We'll refine with user testing later ideally.

12. Riesgos y Decisiones Abiertas

Listamos potenciales riesgos, su impacto, y notas de cómo manejarlos o pendientes de decidir:

- **PII en chats y datos:** Los agentes manejarán datos personales de clientes (teléfonos, emails dentro de leads). Debemos asegurar que nuestra frontend no expone PII inadvertidamente (ej: en logs). También, en UI, implementar políticas de privacidad: por ejemplo, si agente intenta copiar data? (No heavy enforcement possible en frontend). Riesgo: cumplimiento regulatorio (GDPR). Mitigación: mostrar aviso de privacidad en portal, redactar terms. *Acción pendiente:* Confirmar con legal qué datos podemos almacenar en local (we likely store minimal and rely on backend secure).

- **Auditoría y Seguridad:** Necesidad de un audit log (interface provided). Podríamos necesitar registrar acciones sensibles: ex. export lead data, or if an agent deletes info. For now, we'll trust only internal staff uses an admin interface for audit. Risk: malicious agent does something and denies it; we need logs. *Pendiente:* Possibly backend scope. At least, front should not allow actions with severe consequences without confirm (e.g., deleting listing or removing team member).
- **Spam/Fraude via leads:** Agents podrían recibir leads spam (por apertura de portal a público). Debemos preparar:
- Opción de marcar lead como spam/bloquear. Not covered above, but we should plan to add "Mark as spam" in lead actions. If agent marks, we could filter them out.
- Or user impersonation: if a competitor tries to flood leads to annoy. We rely on backend filters normally.
- For chat: injection or malicious links. We should at least auto-turn any link text into clickable but maybe with a rel=noopener. Possibly warn "Cuidado con links de desconocidos".
- **Suplantación (Impersonation):**
- Agents might impersonate other agents if credentials compromised. Not likely but we use standard auth with good practices. Not a front-end concern primarily.
- If agent invites someone, they could invite a personal email to leak data. That is somewhat trust in business processes.
- **Seguridad UX (report/block):**
- Provide a way for agent to block a harassing lead. Not initially planned, but good practice. Could just mark lead lost if harassing.
- Provide a "Report issue" link for agents to contact support if something in portal is wrong.
- Data security: ensure we use https always, tokens not exposed etc.
- **Escalabilidad front:**
- Using virtualization in chat and lazy loading data in tables addresses immediate performance.
- The structure is modular so adding more leads or tasks should not degrade linearly too much.
- If agent has 1000+ leads, pipeline might become heavy if all loaded. We might implement pagination or virtualization on leads list if needed (TanStack Table can virtualize rows).
- Keep an eye on memory usage if data arrays large (but likely fine).
- We'll monitor if any component gets slow (profile early using dev tools).
- **i18n (Internacionalización):**
- For MVP, all UI text is in Spanish (target users presumably Spanish-speaking as prompt was Spanish).
- If multi-lingual needed in future, we'd externalize strings. Now, we code in Spanish copy directly as it’s presumably the requirement. But we ensure not to mix languages inadvertently (some library defaults are English, e.g. RBC might show "Today" or days in English by default; we must localize RBC to Spanish locale).
- Check tooltips or error messages from libraries (Zod default error in English, but we can use Zod's errorMap to Spanish).
- Risk: if later need English, will need to translate all, but as long as we keep text centralized or clearly marked, it’s manageable.
- **Performance en móviles:**
- The portal is heavy on data and charts. On mobile older devices, some parts (Victory charts with many points) could lag. Ensure any large chart dataset is simplified for mobile or can be toggled off.
- Virtualization helps timeline of messages, but ensure not to overload memory by storing too much at once (maybe limit chat history length loaded at once).
- Also test on a mid-range Android via dev tools if possible.
- Keep bundle size in check: tree-shake unused parts of libs. Use code splitting on big sections (maybe reports charts load on demand).

- Evaluate building a PWA or if we are in an app shell context, maybe not needed now.
- **Mobile constraints:**
- Drag & drop Kanban might not be very user-friendly on touch (perhaps add an alternate way: in mobile, show a dropdown to change status as fallback).
- Chat text input should not be hidden by keyboard (take care with viewport resizing).
- Calendar: on small screen, maybe default to agenda list view or allow horizontal scroll on week view.
- We'll test and adjust some features specifically for mobile UX (the tab bar design we mentioned, etc.).
- **Monetization flows vendor:**
- Payment handling (credits purchase) is critical to get right if real money. For MVP, probably not integrated with actual gateway (maybe just track credits). Eventually will use external service like Stripe, so front must integrate stripe checkout. That integration (Stripe JS) is straightforward but ensure not to block main thread. Not doing now, so fine.
- If future boosting or premium features behind paywall, need to implement gating UI (like disable feature with tooltip "Requires premium").
- Also risk of credits miscalculation bug could upset users if charged wrong. We'll double-check ledger logic carefully.
- **Team vs solo differences:**
- If an agent does not belong to a team, the Team page should be hidden or just show "Upgrade to team account" maybe.
- If a broker invites, the roles interplay must be clearly handled (some features only for admin, e.g. routing rules).
- Risk: edge case where agent is in multiple teams? We'll disallow that in backend likely.
- Privacy: Team lead can see team leads info, we should ensure to not show sensitive personal leads outside team (like if an agent has personal leads not through team, does leader see them? Probably yes if all leads come via system but if agent adds a manual lead? Decide policy. Likely, leads belong to whoever generated them. In team context, leads from system distribution presumably belong to team).
- **Integration readiness:**
- Dotloop integration in UI is just a link now. Real integration (OAuth, fetching transactions) is future. We flagged it but not implementing yet, just a placeholder UI maybe "Link dotloop" as in Zillow help [101] .
- Similarly for other future features (like connecting Calendar with Google?), mention but not active.
- **Data syncing**:
- We rely on realtime for chat and new leads. If not present, agent might need to refresh. We should incorporate maybe a simple polling fallback or refresh button in inbox if no websockets.
- Risk: out-of-sync data (lead status updated by co-agent won't reflect on another agent's screen unless we implement real-time updates via WS or refetch on intervals).
- For MVP single-agent usage, not big, but for team we should consider maybe use SWR or context subscribe so changes propagate. Possibly out-of-scope now; clarify with backend how multi-user updates happen (maybe webhooks to client or just refresh often).
- **Offline usage:**
- Likely minimal support. If agent offline, we won't have data except what was cached. We won't build a full offline mode. Risk: if they lose net mid-chat, message might not send (we already cover with error and reattempt).
- Provide decent error feedback if network down (could be a banner "offline" if we detect).
- **Tech Debt vs timeline:**
- Danger: Over-engineering (like trying to fully implement Best of Zillow surveys now).

- We'll stick to MVP scope and note advanced ideas (like AI recommendations) as future enhancements flagged in code or comments.
- We'll create GitHub issues for "Future: implement survey feedback once backend ready" etc., to not forget.

Finalmente, **Siguiente paso recomendado (mañana):** Elaboramos un plan de arranque inmediato en la conclusión.

Siguiente paso recomendado (mañana):

En la siguiente sección presentaremos qué hacer en los próximos 1–3 días para iniciar la implementación conforme a esta especificación.

---

1 5 6 7 14 19 20 22 25 26 28 29 30 47 50 51 52 53 55 63 Navigating the Zillow Premier Agent App and Your Leads - Zillow for Agents
https://www.zillow.com/agents/app-overview/

2 3 8 9 10 13 21 23 24 31 32 33 35 36 48 49 62 101 Close Efficiently - Zillow for Agents
https://www.zillow.com/agents/close-efficiently/

4 17 18 43 44 Your Best of Zillow Report: Collecting Feedback
https://www.zillow.com/agents/collecting-feedback/

11 34 Converting Your Leads and Connections - Zillow for Agents
https://www.zillow.com/agents/converting-your-leads-and-connections/

12 83 Best React scheduler component libraries - LogRocket Blog
https://blog.logrocket.com/best-react-scheduler-component-libraries/

15 16 39 40 41 42 45 46 Track Your Lead Performance - Zillow for Agents
https://www.zillow.com/agents/track-your-real-estate-lead-performance/

27 37 38 Keep Your Team Accountable - Zillow for Agents
https://www.zillow.com/agents/team-reminders-accountability/

54 68 JavaScript Grid: Community vs. Enterprise
https://www.ag-grid.com/javascript-data-grid/community-vs-enterprise/

56 57 58 97 98 99 GitHub - dip/cmdk: Fast, unstyled command menu React component.
https://github.com/dip/cmdk

59 Toast – Radix Primitives
https://www.radix-ui.com/primitives/docs/components/toast

60 Zillow Premier Agent Program Review 2025
https://realestatebees.com/company/zillow/premier-agent/

61 Free Real Estate CRM Software | Zillow Premier Agent
https://www.zillow.com/premier-agent/crm/

64 65 66 GitHub - TanStack/table: Headless UI for building powerful tables & datagrids for TS/JS - React-Table, Vue-Table, Solid-Table, Svelte-Table
https://github.com/TanStack/table

67 70 71 GitHub - Comcast/react-data-grid: Feature-rich and customizable data grid React component
https://github.com/Comcast/react-data-grid

69 react-data-grid/LICENSE at main · Comcast/react-data-grid · GitHub
https://github.com/Comcast/react-data-grid/blob/main/LICENSE

72 r/react on Reddit: TanStack Table vs AG Grid or other Approach for ...
https://www.reddit.com/r/react/comments/1nu2x84/tanstack_table_vs_ag_grid_or_other_approach_for/

73 74 GitHub - FormidableLabs/victory: A collection of composable React components for building interactive data visualizations
https://github.com/FormidableLabs/victory

75 @types/victory - npm
https://www.npmjs.com/package/@types/victory

76 r/reactjs on Reddit: Best looking Charts/graphs for data vizualization ...
https://www.reddit.com/r/reactjs/comments/1pioy31/best_looking_chartsgraphs_for_data_vizualization/

77 GitHub - plouc/nivo: nivo provides a rich set of dataviz components, built on top of the awesome d3 and React libraries
https://github.com/plouc/nivo

78 TypeScript definitions · Issue #197 · plouc/nivo - GitHub
https://github.com/plouc/nivo/issues/197

79 GitHub - chatscope/chat-ui-kit-react: Build your own chat UI with React components in few minutes. Chat UI Kit from chatscope is an open source UI toolkit for developing web chat applications.
https://github.com/chatscope/chat-ui-kit-react

80 What's the best react component / library for realtime chat UI? : r/reactjs
https://www.reddit.com/r/reactjs/comments/w0v4xa/whats_the_best_react_component_library_for/

81 Virtuoso Message List
https://virtuoso.dev/message-list/

82 React Components for Conversational AI - shadcn.io
https://www.shadcn.io/ai

84 GitHub - jquense/react-big-calendar: gcal/outlook like calendar component
https://github.com/jquense/react-big-calendar

85 [blocks]: Full calendar · shadcn-ui ui · Discussion #3214 · GitHub
https://github.com/shadcn-ui/ui/discussions/3214

86 #opensource #react #typescript #frontend #buildinpublic | Sujeet Kc
https://www.linkedin.com/posts/kc-sujeet_opensource-react-typescript-activity-7420222122351845376-b1mX

87 react-hook-form - Snyk Vulnerability Database
https://security.snyk.io/package/npm/react-hook-form

88 The Zod TypeScript Library
https://devm.io/typescript/typescript-zod-library

89 radix-ui/primitives - GitHub
https://github.com/radix-ui/primitives

90 Toast - shadcn/ui
https://ui.shadcn.com/docs/components/radix/toast

91 93 GitHub - emilkowalski/sonner: An opinionated toast component for React.
https://github.com/emilkowalski/sonner

92 Shadcn Sonner
https://www.shadcn.io/ui/sonner

94 95 GitHub - timolins/react-hot-toast: Smoking Hot React Notifications
https://github.com/timolins/react-hot-toast

96 lottie-player Web Component - GitHub
https://github.com/TeamMaestro/lottie-player

100 JohannesKlauss/react-hotkeys-hook - GitHub
https://github.com/JohannesKlauss/react-hotkeys-hook