ChatGPT

Livana (Short-Term Rentals & Local Experiences Platform)

Livana is an online marketplace that connects travelers (guests) with hosts and local service providers. Guests can discover and book short-term home rentals as well as unique local experiences and services offered by trusted providers. The platform emphasizes trust, convenience, and community through verified profiles, secure payments, real-time messaging, and a robust review system. AI-driven personalized recommendations and semantic search further enhance the user experience by helping guests find the perfect stay or activity tailored to their preferences.

Core Features

1) Registration / Login

Goal:

Quick and easy account creation with appropriate onboarding for different user roles (guests, hosts, service providers) to ensure users can start using Livana with minimal friction.

User Stories:

- As a new user, I want to sign up via email or social login (Google/Facebook/Apple) so that I can quickly create an account.
- As a host or service provider, I want to provide additional information (such as identity verification and payout details) during registration so that I can list my offerings.
- As a returning user, I want to log in securely to access my bookings, listings, and messages.
- As a user, I want to update my profile (avatar, bio, contact info) so that others can see who I am and build trust in my identity.

Flow/UX:

1. **Sign Up**: User chooses to register (with option to sign up as a guest, host, or both) → fills basic info or uses a social login → verifies email/phone if required.
2. **Onboarding**: If registering as a host/provider, the system guides them to complete additional profile sections (e.g. adding a payout method, ID verification). If a guest, they may be prompted to set travel preferences (interests, home type preferences, etc.) to personalize their experience.
3. **Profile Completion**: The system reminds users to complete missing profile details (e.g. add a profile photo, verify identity) for a more trustworthy profile. A progress indicator (e.g. "Profile 70% complete") encourages filling out all details.
4. **Login**: Users enter email/password (or use OAuth). On success, a session is created (e.g. JWT token stored) and they are redirected to their dashboard or the homepage.

Permissions & Security:

- Secure authentication using hashed passwords and session tokens (e.g. JWT with refresh tokens for continuous login).
- Email verification is required to activate new accounts (click a verification link or input an OTP code).
- Role-based feature access: only hosts/providers see listing management menus, only guests can make bookings, etc. (a single user account can have multiple roles if needed).
- Password recovery flow (user requests reset → platform emails a secure reset link).
- **Must-have**: Basic CAPTCHA or anti-bot measures on registration to prevent fake accounts.
- **Should-have**: Two-factor authentication (2FA) via SMS or authenticator app for added login security, especially for hosts handling payouts.
- **Good-to-have**: Device recognition and notification of new login (e.g. "New login from Chrome on Windows") to alert users of suspicious account access.

Edge Cases:

- **Duplicate Account**: If a user tries to register with an email that's already in use, the system prompts them to log in or reset password instead of creating a new account.
- **Unverified Email**: If a user hasn't verified their email, they cannot create listings or make bookings until they do. Provide a way to resend verification.
- **Account Lockout**: After several failed login attempts, temporarily lock the account and require additional verification (to prevent brute-force).
- **Role Change**: If an existing guest decides to become a host, allow them to add host capabilities to their account (no need to create a separate account).
- **Social Login Conflicts**: If a user signs up with Google but later tries Facebook with the same email, ensure it links to the same account (avoid duplicate accounts via social logins).

Must-have:

- Email/password registration and login with validation and error feedback.
- OAuth social logins (at least one, e.g. Google or Facebook) for convenience.
- Basic user profile management (name, profile photo, bio, phone number, etc.).
- Secure password storage (hashed with salt, e.g. bcrypt) and secure session handling.
- Email verification process on sign-up.
- Password reset functionality via email link.
- Distinct account roles or flags (guest/host/service provider) stored in user profile to gate certain features.

Should-have:

- Support for Sign in with Apple (especially for iOS app compliance).
- Phone number collection and verification (SMS OTP) to increase trust, especially for hosts and for using messaging/notifications.
- Identity verification flow for hosts/providers (upload ID or integrate with a verification service) to earn a "Verified" badge.
- Profile completeness meter and incentives (e.g. cannot publish a listing until certain profile fields and verifications are done).
- Option for users to deactivate or delete their account (with proper data handling).

Good-to-have:

- Login with biometric on mobile (fingerprint/FaceID after initial login).
- Session management: ability for user to see and revoke active sessions/devices logged in.
- Multi-language support for the registration interface, catering to international users.
- Admin tools to manage users (approve hosts, view verification status, ban fraudulent accounts).

2) Listings (Homes and Services)

Goal:

Enable hosts and service providers to create detailed listings for accommodations and experiences, and allow guests to browse these listings with all the information needed to make a booking decision.

User Stories:

- As a **host**, I want to list my property (home/apartment) with details like photos, description, amenities, location, and price so that guests can learn about it and book it.
- As a **service provider**, I want to create a listing for a local experience or service (e.g. a city walking tour, cooking class) with a description, schedule, and pricing so that travelers can book it.
- As a **guest**, I want to view a listing's details (photos, descriptions, reviews, host info) so I can decide if it suits my needs before booking.
- As a **host**, I want to edit or update my listing (change pricing, add new photos, update availability) to keep information accurate and attract more bookings.
- As a **host**, I want to pause or deactivate my listing temporarily (e.g. if my property is under renovation or I'm unavailable) without deleting it, so that I can reactivate it later.

Must-have:

- **Create Listing Flow**: A guided form for hosts/providers to input listing details. Required fields:
- *For homes*: Title, detailed description, property type (e.g. apartment, house), location address (with map integration to pin it), number of guests/rooms/bathrooms, list of amenities, high-quality photos upload, nightly price, and an availability calendar (or default availability if always open).
- *For experiences*: Title, description of the experience, category (e.g. adventure, food, culture), location or meeting point, duration, available dates or schedule, minimum/maximum group size, price per person (or per group), and photos.
- **Listing Page Display**: A public page for each listing showing all the info:
- Photo carousel (with the ability to enlarge images).
- Key details at a glance (location, number of guests, bedrooms, amenities for homes; or date/time availability for experiences).
- Full description and house rules or experience itinerary.
- Host/Provider profile snippet (name, profile photo, verification badge, and a way to view full profile).
- Guest reviews and ratings (see Reviews & Ratings feature).
- "Contact Host" and "Book Now" buttons readily accessible.
- **Calendar Management**: Hosts can mark dates as unavailable or set custom pricing for specific dates (e.g. weekends or holidays) on a calendar interface for each home listing. Experience providers can set up time slots or sessions for their activities.

- **Listing Management Dashboard**: Hosts have a dashboard to see all their listings in one place, each with status (active/inactive), and options to edit, manage calendar, or delete the listing. They can also see upcoming bookings for each listing.
- **Validation & Media**: The system validates inputs (e.g. price can't be negative, title isn't empty). Photo uploads are stored securely (e.g. cloud storage) and optimized for web. There should be a limit on number of photos (e.g. up to 20 photos) and size per photo (with helpful error messages if exceeded).
- **Distinct Categories**: The platform clearly distinguishes accommodations (homes) from experiences in search and UI (different sections or badging, so users know if a result is a place to stay or an activity to do).

Should-have:

- **Listing Approval**: New listings might go into a pending state for admin review to ensure quality and prevent inappropriate content before they appear in search. (Admins can approve/reject or ask for edits.)
- **Rich Descriptions**: Support simple text formatting or even Markdown in descriptions so hosts can structure information (bullet points for house rules, etc.).
- **Seasonal & Dynamic Pricing**: Hosts can set seasonal rates (e.g. high-season vs off-season pricing) or discounts for longer stays (weekly/monthly discount). For experiences, allow setting different prices for private vs group or for children vs adults if applicable.
- **Additional Fees**: Hosts can specify extra fees (cleaning fee, security deposit) and those will be shown in the price breakdown.
- **Amenities & Features**: A comprehensive list of amenity options with icons (for homes: Wi-Fi, Air Conditioning, Parking, Pet-Friendly, etc.). These can be filterable in search.
- **Experience Scheduling**: Providers can offer multiple upcoming dates for the same experience listing (recurring events). A guest can pick a specific date/time slot to book.
- **Multi-language Support for Listings**: Hosts can provide their listing description in multiple languages. Guests see the version based on their language preference (or an automatic translation if none available).
- **Host Calendar Sync**: Allow hosts to sync their Livana calendar with external calendars (iCal/Google) to avoid double-booking if they list on multiple platforms.
- **Analytics for Hosts**: Provide hosts with simple analytics (number of views, booking conversion rate, etc. for their listings) to help them improve attractiveness.

Good-to-have:

- **Video & Virtual Tours**: Hosts can upload a short video of the property or a YouTube link. For experiences, providers might upload a video teaser of the activity.
- **Draft Listings**: Hosts can save a listing as draft and publish later (useful if they need time to gather info or photos).
- **Geo-aware Amenities**: If location has special attributes (e.g. "Beachfront", "City Center"), auto-tag or allow hosts to tag these for search filters.
- **Listing Clone**: Hosts with similar properties can duplicate a listing as a template and then tweak details, to save time.
- **Offer Management**: Hosts can create special offers or discounts (e.g. last-minute deals, early-bird pricing for experiences) that are highlighted on the listing.

- **Trust & Verification Badges**: Besides host identity verification, show if a listing is "verified" by the platform (perhaps after an in-person check or certain quality criteria) or if the host is an experienced superhost (based on ratings and number of trips).

Flow/UX:

1. **Creating a Listing**: Host clicks "Add New Listing" → selects type (Home or Experience) → goes through a multi-step form:
2. Step 1: Basic info (title, location, category).
3. Step 2: Details (amenities for home, itinerary for experience, etc.).
4. Step 3: Photos upload (with drag-and-drop or mobile camera integration).
5. Step 4: Pricing and availability.
6. Step 5: Preview & Publish. The host can see a preview of how the listing will appear to guests and then confirm to publish.
7. **Viewing a Listing**: A guest finds a listing via search or link. On the listing page, they scroll through images (tap to open fullscreen gallery on mobile), read the description and amenities, see the map location, and browse reviews. They can select dates (for a home) or a session date (for an experience) and click "Book Now" to initiate booking.
8. **Editing a Listing**: Host opens their listing management page, chooses a listing and edits any field (description, price, etc.) or updates the calendar (e.g. block dates). After saving changes, the listing page is updated for all users (significant changes might re-trigger admin approval if that's in place).
9. **Pausing/Deleting**: Host toggles a listing to "snooze" (temporarily unlist) via a switch in the dashboard. If a listing is deleted, it's removed from search immediately; any future bookings might need to be handled or cancelled via support.

Edge Cases:

- **Incomplete Info**: Host tries to publish without filling required fields → highlight the missing info (e.g. no description or zero photos) and prevent publishing until resolved.
- **Invalid Address**: If a host enters an address that the map can't find, prompt them to adjust it or pin the location manually on a map.
- **Image Upload Failures**: If a photo upload fails (network issue or file too large), show a clear error and allow retry. If some photos uploaded and others didn't, ensure partial uploads are handled (host can save draft and come back).
- **Overlapping Listings**: A service provider listing an experience that is actually a property (or vice versa) might confuse categories. Ensure the UI and flow discourage miscategorization (maybe different form fields for each type).
- **Calendar Overbooking**: If a host manually blocks dates or gets an external booking, and a Livana guest tries to book slightly before that update syncs, handle race conditions by verifying availability again at booking confirmation (to avoid double-booking).
- **Listing Removed**: If a listing is removed or deactivated while a guest is viewing it (or after they booked it), handle gracefully:
- If deactivated: show "Listing is no longer available" notice.
- If booking exists and host deleted listing: booking remains but no new ones can be made; prompt host to honor or cancel existing reservations.
- **Content Moderation**: If a host uses disallowed words or uploads inappropriate images, the admin/moderation system should catch it (see AI features) and either automatically hide the listing or flag for review.

3) Search & Filters

Goal:

Allow guests to easily find the most suitable accommodation or experience by searching with location and date criteria and refining results with filters. The search system should be fast, intuitive, and return relevant results even with diverse query types (including textual searches like "cozy cabin with mountain view").

User Stories:

- As a **guest**, I want to search for available homes in a specific location for my travel dates and party size, so that I only see properties I can actually book for that trip.
- As a **guest**, I want to filter search results by price range, property type, amenities (Wi-Fi, pool, pet-friendly, etc.), and guest ratings, so I can narrow down to listings that meet my exact needs.
- As a **guest**, I want to search for experiences in my destination (e.g. "wine tasting in Paris") and filter by date or category, so I can plan activities during my travel.
- As a **guest**, I want to see results on a map as well as in a list, so that I can choose a location convenient for me (for homes, proximity matters; for experiences, I might want ones near where I stay).
- As a **user**, I want search suggestions or autocomplete when I type a location or keyword, so that I can quickly pick a valid destination or popular search term.

Must-have:

- **Location & Date Search Bar**: A prominent search bar where users enter a location (city, region, or even specific attraction name) and select dates and number of guests. (For experiences, date may be optional to see all upcoming sessions.)
- **Result Listings**: A results page that displays listings matching the query:
- Each result item shows a photo, title, location, price per night or per experience, and rating/review count.
- Indicates if it's a home or experience (could use an icon or label).
- If dates are specified for homes, only those available for the entire date range are shown.
- **Filters UI**: Key filters like price range slider, property type (entire home/private room/shared), for experiences: category (sightseeing, adventure, etc.), and checkboxes for top amenities or features. Also a filter for "Instant Book" vs "Request to Book" if relevant.
- **Sorting Options**: Default sort by a combination of relevance and quality (considering factors like ratings, popularity, possibly personalization). Provide options to sort by Price (low to high, high to low) and by Rating, maybe by Distance if map is used.
- **Map View**: Integration of an interactive map on the search results page (especially for accommodations). Users can toggle map view on/off. Map shows pins for listings; clicking a pin highlights that listing in the list (and vice versa).
- **Pagination/Infinite Scroll**: If many results, either paginate or infinitely scroll the results list to load more as the user scrolls, without overwhelming them initially.
- **Empty State**: If no listings match the search, show a friendly message with suggestions (e.g. "Try adjusting dates or filters" or offer to broaden location search to nearby areas).
- **Mobile Responsive**: On mobile, allow filtering and map use in a user-friendly way (e.g. filters as a slide-up panel, map either full-screen toggle or a smaller view).

Should-have:

- **Advanced Filters**: More granular filters such as:
- Amenities (multiple checkable options like "Wi-Fi", "Kitchen", "Parking", "Washer", etc.).
- Accessibility features (wheelchair access, etc.).
- Neighborhood or district filter if searching a big city (e.g. "Manhattan" vs "Brooklyn").
- Language of host (if the guest prefers hosts who speak certain language).
- For experiences: duration (e.g. 2 hours, full day), physical difficulty level, or appropriate for families/kids.
- **Autocomplete & Suggestions**: As the user types in the search bar, show suggestions like city names, popular destinations, or even specific listings ("Did you mean: Beachside Bungalow in Malibu?" if they type something specific). Also support recognizing major tourist attractions or regions (e.g. user types "Eiffel Tower" and system suggests "Paris - near Eiffel Tower").
- **Recent & Saved Searches**: If logged in, users can see their recent searches for quick re-use. Allow users to save a search criteria (e.g. "2 guests in Tokyo next month under $100") and opt-in to get alerts when new listings or price drops match that search.
- **Map Drawing**: An advanced map feature where users draw a shape on the map to restrict results to a specific area (useful in large cities).
- **Highlighting Deals**: If any listing is a new listing or offering a discount, show a badge ("New", "Discount 10%") in results to attract attention.
- **Internationalization**: If the platform covers multiple countries, allow searching in different languages (e.g. someone searches "Londres" should show London), and handle multi-currency display (show approximate price in user's currency if different from listing currency).
- **Integrated Experience Search**: The search interface could allow toggling between "Homes" and "Experiences" results (or show both in sections). For example, a user search in Rome could have tabs for accommodations vs activities.

Good-to-have:

- **Natural Language Search** (AI Semantic Search integration): Allow users to type queries like "pet friendly villa with pool in southern France under $300" and interpret that to apply filters and semantic matching (e.g. find listings that might not literally have "pet friendly" in title but have that amenity).
- **Voice Search**: On mobile apps, enable voice input for search queries (which can be converted to text and processed).
- **Ranking Personalization**: Re-order results subtly based on user's past behavior (if the user often books cheaper places, show affordable options first, etc. – ensure transparency and relevance).
- **Collaborative Search**: Let users share their search or favorites with travel companions (e.g. a link or in-app collaborative list) when planning together.
- **Search Analytics for Hosts**: not user-facing, but provide hosts insight into how often their listing appears in searches and is clicked (so they can adjust price or content if it's appearing but not getting booked).

Flow/UX:

1. **Entering Search Criteria**: User lands on the homepage (which includes the search bar) or a dedicated search page. They enter "Where" (with autocomplete suggestions), select dates and

number of guests (if searching for homes; for experiences, maybe select a date or leave it open), then click Search.

2. **Viewing Results**: The results page loads matching listings. The user sees a list and optionally a side map. They scroll through results; if too many, they use filters (tap "Filter" to open filter panel, select criteria, then apply).

3. **Filtering**: User sets a price range, selects a few amenities, and hits Apply. The list updates immediately to only show listings that match. The map also updates pins accordingly.

4. **Map Interaction**: The user switches to map view (map full-screen with results as pins). They zoom/pan the map to a specific area; if implemented, the system updates results to that area ("Search this area" button or automatically).

5. **Choosing a Listing**: The user clicks on a promising listing in the list (or taps a pin on the map then the preview) to view details. They may use the browser/app back button to return to the search results and continue browsing.

6. **No Results**: If the filters were too narrow (e.g. "under $50 but 5 bedrooms"), the system shows a "no results" message with tips or an option to reset filters. Possibly, it could automatically broaden some criteria (like extend the radius or remove an uncommon filter) and inform the user.

Edge Cases:

- **Zero Availability**: User enters specific dates but every listing in that location is booked. The result might be empty – system could suggest removing dates to see all listings or show nearby cities that have availability.
- **Partial Availability**: If a listing is available for some but not all of the user's dates, it should not appear at all (to avoid confusion) unless the platform decides to show it with a note (but typically it's excluded).
- **Invalid Location Input**: If a user types something too vague or unknown (e.g. "Middle Earth"), handle gracefully with "Please enter a valid destination". If it's a small place not in our database, perhaps use a geolocation API to interpret it or say "Showing results for [nearest known area]".
- **Timing Out**: If the search query takes too long (maybe due to complex filters), ensure a reasonable timeout and show an error or retry option. The search should be optimized to handle many listings quickly (using indexes, etc.).
- **Stale Listings**: If a listing was just booked or deactivated, it might still appear in results the user is looking at. Final availability is confirmed when they attempt to book, but it's good to refresh availability if a user stays on the search page for a long time or add a refresh button.
- **Language Mismatch**: If a user searches in one language and there are no listings with that language in description, semantic search should bridge the gap (see AI Semantic Search). Otherwise, consider falling back to showing results anyway rather than none.
- **Map Pin Overlap**: In dense areas, many listings near each other might cluster. Use clustering pins or slight offsets so all can be accessed.
- **Filter Dependencies**: Some filters only apply to homes or experiences (e.g. "amenities" for homes vs "duration" for experiences). The UI should hide or disable irrelevant filters depending on category to avoid confusion.

4) Booking & Payment

Goal:

Provide a seamless and secure process for guests to book listings (homes or experiences) and handle payments and transactions between guests and hosts/service providers. The system should manage availability in real-time, collect payment from guests, pay out to hosts, and keep a record of all transactions, while enforcing cancellation and refund policies.

User Stories:

- As a **guest**, I want to book a home or experience for specific dates and know instantly if it's available, then pay securely through the platform so that my reservation is confirmed.
- As a **guest**, I want to see a clear price breakdown (listing price, service fees, taxes) before I confirm payment, to avoid surprises.
- As a **guest**, I want to be able to view all my bookings (upcoming and past) in one place, so I can manage my itinerary and see details like addresses or start times.
- As a **host**, I want to receive booking requests or instant bookings and have the ability to accept or decline requests if I choose, so that I have control over who stays or joins my experience.
- As a **host**, I want to be notified immediately of new bookings and have the booking details accessible (dates, guest info, any special requests) so I can prepare for the guest.
- As a **host**, I want the platform to handle collecting money from guests and then paying me after the booking, so I don't have to deal with payments directly.
- As a **guest or host**, I want clear cancellation policies and a way to cancel if needed (with potential refunds or penalties automatically calculated), so that there is fairness in case plans change.
- As a **host**, I want to see a history of all transactions (what I've earned, any payouts pending) for accounting purposes.

Must-have:

- **Real-Time Availability & Booking Lock**: When a guest selects dates for a home (or a session for an experience) and proceeds to book, the system should lock those dates (tentatively) to prevent others from booking them at the same time. Availability must be checked again at confirmation to avoid double-booking.
- **Instant Book vs Request to Book**: Hosts can choose if their listing is Instant Book (no approval needed) or requires request:
- *Instant Book*: The booking is immediately confirmed and paid once the guest completes checkout.
- *Request to Book*: The guest sends a booking request with optional message. The host must accept or decline within a set time (e.g. 24 hours). The guest's payment is authorized but not fully charged until acceptance.
- **Secure Payment Processing**: Integration with a payment gateway (e.g. Stripe) to handle credit/debit cards, and possibly PayPal or other local payment methods. All payments happen through the platform; no cash transactions. Card info is stored securely via tokenization (PCI compliance).
- **Payment Flow**: The guest is shown the total amount and enters payment details → upon confirmation, if instant booking, charge the amount immediately; if request, place a hold. In either case, funds are held by the platform until after the check-in date or experience date, then automatically released to the host (minus fees).

- **Price Breakdown & Fees**: Show line-by-line breakdown: listing price (e.g. $100 x 3 nights = $300), cleaning fee if any, service fee (platform fee charged to guest, say 10%), taxes (if applicable), and total. The host sees a similar breakdown of their earnings (e.g. $300 minus host fee, etc.).
- **Confirmation & Itinerary**: Once booked, generate a booking confirmation page and unique confirmation code. Details include: listing name, address (for stays, revealed only after booking for privacy), check-in/check-out date and times, for experiences the start time and meeting location, guest and host contact info (masked or full depending on stage), and any instructions. Send confirmation via email and in-app.
- **Booking Management (Guest)**: Guests have a "Trips" section listing upcoming bookings and past bookings. They can click an upcoming trip to see details, cancel (if within allowed period), or message the host. Past trips show a summary and a prompt to review.
- **Booking Management (Host)**: Hosts have a "Reservations" view where they see all requests and confirmed bookings for their listings. For each reservation, they can view guest info, booking dates, payout amount, and any special notes. Hosts can accept/decline requests here, or cancel in an emergency (discouraged).
- **Calendar Sync**: The system automatically updates the listing's availability calendar upon booking confirmation (block those dates). If a booking is canceled, dates become available again.
- **Cancellation & Refunds**: Implement basic cancellation policy logic:
- If guest cancels a confirmed booking, apply the listing's cancellation policy (e.g. full refund if far in advance, partial if last-minute). The system calculates refund amount and returns money to guest's card accordingly.
- If host cancels, the guest should get a full refund and possibly some compensation (and host might face a penalty or automatic review stating "host canceled").
- The platform should track cancellations and enforce any penalties (e.g. host cancellation fee or lowering host ranking if they cancel often).
- **Transaction Records**: Each financial transaction (charges, refunds, payouts) is recorded. Guests can get receipts for their payments. Hosts can get statements for payouts. This could be as simple as emails or downloadable invoices.

Should-have:

- **Multiple Payment Options**: Support additional payment methods popular in various regions (PayPal, bank transfer, Apple Pay/Google Pay, etc.). Possibly an in-app wallet where users can hold credits (from refunds or gift cards) and apply them.
- **Host Payout Preferences**: Hosts set up how they receive money (bank account for ACH transfer, PayPal, etc.) in an earnings section. The system automates payouts to hosts X days after check-in or event date (to account for any early cancellations).
- **Partial Pay or Deposit**: For long lead-time bookings or high-cost bookings, allow an option to pay a deposit (e.g. 50% now, 50% closer to check-in) – this is complex but can be a good-to-have for flexibility.
- **Modify Booking**: Allow guests to request changes to a booking (dates or number of guests). The host can approve the change, and the system will handle any additional charge or partial refund as needed (with guest's payment method).
- **Messaging Integration**: Link the booking with the messaging system – e.g. when viewing a booking, have a "Message Host" button that opens the chat specific to that reservation.
- **Automated Communication**: Send reminder notifications to guests before check-in or experience (e.g. "Your check-in is tomorrow at 3 PM" or "Your tour starts in 2 hours, meet at…"). Also remind hosts of upcoming reservations.

- **Guest Identity Verification**: For security, require guests to also verify ID or payment method authenticity for high-value bookings. Could use a risk engine to flag if needed (ties into fraud detection).
- **Commission & Taxes**: Automatically calculate the platform's commission on each booking and record it. If needed, integrate tax collection for jurisdictions that require the platform to collect occupancy taxes or VAT (this might be region-specific).
- **Waitlist for Experiences**: If an experience is fully booked on a date, allow guests to join a waitlist and get notified if a spot opens or new session added.

Good-to-have:

- **Group Payment Split**: Allow a booking's cost to be split among multiple guests at checkout (each pays their share via invite link) – useful for group trips (complex to implement coordination of payments).
- **Loyalty Programs**: Introduce reward points or credits for each booking completed, which guests can accumulate and use towards future bookings (to encourage repeat usage).
- **Insurance Offerings**: During booking, offer travel insurance or damage protection options (via third-party integration), which the guest can add.
- **Dynamic Pricing for Hosts**: (Related to AI) Provide hosts with a pricing tool that suggests optimal prices (this ties to AI features like dynamic pricing recommendations similar to Airbnb's Smart Pricing).
- **Multi-currency display**: If platform is global, show prices in user's preferred currency with conversion (while charging in host's currency or a fixed currency internally).
- **Carbon Offset/Charity option**: Let guests add a small donation or carbon offset fee at checkout (to support sustainability, etc.).
- **Referral Discounts**: If a guest was referred or has a coupon, allow entering a promo code at checkout for a discount.

Flow/UX:

1. **Initiating Booking**: Guest finds a listing and enters the booking details (dates, number of people). They click "Book Now" (for Instant Book) or "Request to Book" (if host approval needed). A booking summary screen appears.
2. **Review & Pay**: The guest is shown the price breakdown, selects a payment method, and enters payment details if not already on file. They must agree to the house rules and cancellation policy (ticking a checkbox) before confirming.
3. **Confirmation**:
4. If Instant Book, within a few seconds the payment is processed and the booking is confirmed. The guest sees a confirmation screen with reservation details. The listing's calendar is instantly updated to block those dates.
5. If Request to Book, the guest sees a pending notice: "Your request has been sent to the host. They have 24 hours to respond. Your card will be charged once they accept." The request appears in their Trips as pending. The host is alerted.
6. **Host Response** (for requests): The host gets a notification and can view the booking request details (guest profile, message, dates). They click Accept or Decline.
7. If **accepted**, the guest receives a notification of confirmation, and the payment is captured. It transitions to a confirmed booking for both parties. If the host ignores the request, it auto-expires after the set time, and the hold on the payment is released.

8. If **declined**, the guest is notified (with a polite message or reason if provided). The guest's payment hold is released. The dates become available again for someone else.
9. **During the Trip Lifecycle**: The guest and host can communicate via messaging. Prior to check-in, automated reminder emails/messages are sent. After check-out, the system triggers the payout to the host's account (assuming no issues reported) and sends review prompts.
10. **Cancellation**: If the guest clicks "Cancel" on a future booking, the system will show the refund amount per the policy (e.g. "You will be refunded $X of $Y paid" or "This booking is non-refundable for the first night but $Z will be returned"). The guest confirms, the booking is marked canceled, calendars freed, host notified. Refund is processed automatically to their card. If a host needs to cancel, they use a "Cancel booking" button on their side (or contact support if restricted); the guest gets full refund and assistance to find alternatives.

Edge Cases:

- **Payment Failure**: If a guest's card is declined during booking, show an error and allow them to retry with a different method. Do not confirm booking without a successful charge or authorization.
- **Concurrent Requests**: Two guests request the same dates for a home around the same time. If the host accepts one, the system should automatically decline the other as the dates just got booked (and inform that guest it's no longer available). This requires checking availability on acceptance as well.
- **Overbooking**: If an external sync or manual error causes a double booking (e.g. host forgot to update calendar), the host may need to cancel one booking. This is handled as a host-initiated cancellation (penalties may apply). The guest who is canceled on should be assisted by support to find alternate accommodations (possibly with a coupon for inconvenience).
- **Last-Minute Booking**: If a guest books a stay for the same day check-in, ensure the host is immediately notified (perhaps an SMS in addition to app notification due to urgency). Also, consider cutoff times (hosts can set no same-day bookings after a certain hour).
- **Fraudulent Booking**: If the system's fraud detection flags a booking (e.g. stolen credit card usage), it might place it on hold or require additional verification (the booking might be marked tentative until cleared).
- **Currency Fluctuation**: If multi-currency is supported and there's a gap between authorization and capture (in request-to-book), currency rates might change slightly. Ideally charge is done at authorization time or locked in, to avoid differences – communicate clearly the currency handling.
- **Host No-Show (for experiences)**: If a host doesn't show up for an experience or a significant issue occurs, there needs to be a support process. While not directly in booking flow, it's part of post-booking issue resolution (refund guest, possibly compensate).
- **Partial Attendance**: For experiences, if a guest books for 5 people but only 4 show up, policies on refund for the missing person need to be considered (usually no refund after booking for no-shows).
- **Payout Failures**: If a host's payout fails (invalid bank account, etc.), system should notify host to update their info and retry the payout automatically or via support intervention.

5) Messaging System

Goal:

Facilitate clear and immediate communication between guests and hosts/service providers through an in-app messaging system. This helps users ask questions, coordinate details (like check-in time or meetup location), and build trust, while keeping contact info confidential until booking for safety.

User Stories:

- As a **guest**, I want to message a host with questions about a listing (availability of amenities, check-in procedure, etc.) before booking so I can make an informed decision.
- As a **guest**, I want to message my host after booking to discuss arrival time or ask any additional questions, so that our stay/experience is well-coordinated.
- As a **host**, I want to communicate with potential guests who have inquiries and confirmed guests, to answer their questions and provide necessary information (like directions or instructions).
- As a **user**, I want to receive a notification when I get a new message, so that I can respond promptly.
- As a **user**, I want all my conversations stored in one place in the app/website (an inbox), so I can easily refer back to what was said.

Must-have:

- **1-on-1 Chat**: Real-time messaging between a guest and a host (or provider). Each conversation thread is typically tied to a specific listing or booking (contextual messaging).
- **Initiating Conversations**:
- Guests can initiate a message thread with a host either by clicking "Contact Host" on a listing (before booking) or from a booking confirmation page ("Message Host").
- Hosts can send a message to a guest who has inquired or booked. (Hosts should not generally initiate contact with a user who has never messaged or booked them for privacy reasons.)
- **Real-Time Delivery**: Use web socket connections (or long-polling) to deliver messages instantly to online users. Messages appear in the chat UI in real-time, similar to instant messaging apps.
- **Basic Messaging Features**: Support plain text messages. Allow sending newline/paragraphs. Show messages in a chat bubble style with timestamp and sender name (or role, e.g. "Host" or "Guest") clearly. Recent messages should be visible when opening a conversation.
- **Unread Indicators**: The messaging icon in the app shows if there are unread messages (badge count). Within the inbox, highlight conversations with unread messages. Optionally show bold text for unread messages in a conversation.
- **Notifications**: When a new message arrives and the user is not currently viewing that chat, send a push notification (mobile) or show an in-app notification popup (and/or email if user is offline) saying "New message from [Name]".
- **Safety/Privacy Guard**: Automatically mask or prevent sharing of direct contact info in pre-booking messages (to keep communication on-platform). For example, if someone sends a phone number or email, the system could redact it or warn the users (to reduce off-platform transactions or scams).
- **Message History**: All messages are stored so users can scroll up to see previous communications. Messages should be persistent (stored in database) so that if a user logs out or uses a new device, the conversation is still available.

Should-have:

- **File/Photo Sharing**: Allow users to send images or PDF documents in chat. For example, a host might send a photo of a landmark to help find the house, or a guest might send a photo of a receipt if needed. These should preview in the chat and be downloadable. (Include file size/type restrictions for security.)
- **Typing Indicator**: Show a subtle indicator when the other party is typing a message ("... is typing").
- **Read Receipts**: Indicate when messages have been read by the other side (e.g. a small checkmark or "Seen" timestamp).

- **Quick Reply Templates**: Hosts can save canned responses (e.g. "Yes, those dates are available. I look forward to hosting you!") and insert them quickly to speed up common replies.
- **Group Messaging for Experiences**: If an experience is booked by multiple separate guests (group activity), consider a group chat that includes the provider and all guests for that session so everyone can coordinate (e.g. for carpooling to a tour). (This is more complex, could be a good-to-have.)
- **Contextual Prompts**: The system might provide smart replies or reminders in the chat. For example, if a booking is upcoming, an automated message or template could prompt the host to share check-in instructions through the chat.
- **Offline Notice**: Show if a user is likely offline. For example, "Last online X hours ago" for the host, so the guest knows why there might be a delay (but this might conflict with privacy, use carefully).
- **Archiving**: Allow users to archive conversations that are no longer active (to declutter their inbox), while still keeping them in history.
- **Multi-language translation**: Provide a "Translate" button on messages if the app detects the message is in a language different from the user's preferred language (could integrate with a translation API).

Good-to-have:

- **Voice Messaging or Calls**: Allow sending short voice notes in chat, or even VoIP calls through the app once a booking is confirmed (keeping phone numbers private). This can help in scenarios where typing is inconvenient (though recording and storing voice adds complexity).
- **AI Assistant**: An AI chatbot that can assist in answering common questions within the chat. For instance, if a guest asks "Is there parking available?", the AI could auto-suggest an answer from the listing info or FAQ for the host to approve/send (or directly answer if the info is straightforward).
- **Message Search**: A search bar within messaging to find a specific piece of info (e.g. search "wifi" to find the message where the host provided the Wi-Fi password).
- **Delivery Status**: Show whether a message is successfully delivered (one tick) vs not (if the recipient is offline, still queueing).
- **Expiration**: Option for messages to expire or be retracted. E.g., if someone accidentally sent something, maybe allow deletion within a short window (though best to avoid misuse).
- **Integration with Email**: Copy of messages could optionally be sent to email (some users might prefer replying via email which then posts to chat – a complex but useful bridge).

Flow/UX:

1. **Contacting Host (Pre-booking)**: On a listing page, guest clicks "Contact Host". If not logged in, they are prompted to log in or sign up. Then they see a chat interface pop up or navigate to the messaging section with a new thread to that host (listing reference included, e.g. "Regarding: Cozy Cabin in Tahoe"). They compose a message ("Hello, I love your cabin, is it available for New Year's?") and send.
2. **Host Receives Message**: The host gets an alert (if in app, a push; if not, maybe an email saying "You have a new inquiry from [Guest Name]"). The host opens the app, goes to messaging, sees the new conversation, reads the message, and replies.
3. **Real-time Exchange**: If both are online, messages appear in real-time. The guest can see the host typing indicator and then the message appears: "Yes, it's available. We'd be happy to host you!" The guest can then proceed to book or ask more questions.

4. **Post-booking Chat**: Once a booking is confirmed, the existing message thread could be tagged as "Booked - [Listing Name]" and continue to be used for that reservation's communications. The guest might ask "What's the Wi-Fi password?" a day before arrival, and the host responds.
5. **Inbox Management**: The user's inbox shows all conversations, e.g. "Cozy Cabin – Host: Alice" (with a preview of last message) and "Paris Food Tour – Host: Bob". Unread ones might be at top with a badge. The user can tap any to continue chatting. They can also start a new message to another host from a listing.
6. **Notifications & Replies**: If a user taps a notification of a new message, it deep-links into that conversation. If they receive an email (for offline message), clicking a special link could also bring them to the chat after login.

Edge Cases:

- **Spam or Harassment**: If one user is spamming or sending inappropriate content, the recipient should be able to report or block them. Blocking a user means they cannot send further messages (and possibly cannot book your listing if it's a host blocking a guest). The platform should review reported messages (tie-in to moderation AI for automatic detection of abusive language).
- **Messaging after booking is over**: By default, allow continued communication even after the trip (to say thanks or handle left-behind items). But some hosts/guests may not want further contact; blocking or archiving can help. The system might auto-archive threads some time after the trip.
- **Contact Info Exchange**: Despite platform discouragement, some may try to share phone/email. The system might automatically mask detected phone numbers or URLs in early messages ("###" or "[redacted]") and warn users about off-platform risks. Once a booking is confirmed, sharing info might be allowed for coordination (since they will meet in person).
- **Offline Delivery**: If a user is completely offline (no app, no email access temporarily), messages will be queued. They'll get them when they return. There could be scenarios where timely communication is critical (last-minute changes). If a message isn't read, perhaps escalate via SMS or phone call by support if urgent (e.g. host trying to tell guest "Don't come, emergency!").
- **Multiple Guests**: If group chat is implemented, ensure that if one guest cancels their spot, they are either removed from the group chat gracefully, and others remain.
- **API Integration**: If messages are sent via email reply-to (some systems allow replying to the notification email which then routes to chat), handle formatting and stripping email signatures, etc. (advanced).
- **Data Privacy**: Comply with privacy laws – allow users to request their message data or deletion (though deleting might be problematic for dispute evidence; perhaps anonymize after certain time per GDPR if requested).
- **Notification flood**: If many messages arrive (rare in 1-1 chat), ensure not to spam too many notifications. Use standard throttle (i.e., if a user sends 5 rapid messages, maybe group the notification).

6) Reviews & Ratings

Goal:

Establish a trusted feedback system where guests can leave reviews and ratings for listings or experiences after their trip, and hosts/service providers can (optionally) review guests. This two-way review system builds transparency and accountability, helping future users make informed decisions and encouraging everyone to maintain high standards.

User Stories:

- As a **guest**, I want to rate and review the place I stayed or the experience I attended after it's over, so I can share my honest feedback with others.
- As a **future guest**, I want to read reviews from previous guests on a listing's page, so I know the pros and cons and can book with confidence.
- As a **host**, I want to receive feedback from guests so I can improve my offering, and also to have positive reviews that attract more guests.
- As a **host**, I want to (in a private or public way) rate or flag a guest's behavior (if they were very clean, or if they caused issues) to inform other hosts or at least keep a record.
- As a **host**, I want the ability to respond to a review left on my listing to clarify or thank the guest, so that my perspective is also recorded.

Must-have:

- **Review Prompt & Timing**: After a booking is completed (e.g. the day after check-out or after the experience date), the system automatically sends a prompt to the guest to leave a review. Typically, the window for review is limited (e.g. within 14 days of checkout).
- **Star Rating**: Guests give an overall rating, usually 1 to 5 stars. This rating will contribute to the listing's average rating. (If doing category ratings, those are should-have, see below.)
- **Written Review**: Guests can write a text review describing their stay/experience. There should be a reasonable min/max length (e.g. 20 characters min to avoid blank, and maybe 1000 chars max).
- **Review Display**: On each listing page, display the average star rating (e.g. "4.8 stars out of 5, based on 12 reviews") prominently near the title. Below, list individual reviews chronologically (newest first). Each review shows the star rating, the guest's first name and last initial, their photo (if available), the date of their stay, and their comments. Possibly also show if the review was for a specific month/year ("Stayed in Jan 2025").
- **Host Response**: Allow the host (or experience provider) to write a single public response to each review. This response is shown indented under the guest's review (with host label).
- **Two-way Reviews**: The system can also prompt the host to leave a review for the guest. If implemented, host's review of guest might be visible only to other hosts (not publicly on listing, since listing page is guest-focused). Alternatively, hosts rating guests could be private notes or just star ratings that contribute to an internal "guest score". This is often done to allow hosts to see if a guest had issues before (if we implement such a feature).
- **Review Integrity**: Only guests who actually completed a booking can review that listing (no random users or friends posting fake reviews). If a booking was canceled before completion, typically no review is allowed (or if allowed, it should indicate it was a canceled trip).
- **Moderation of Reviews**: Basic profanity filter or check (the content goes through an AI or at least regex filter for disallowed content) before posting live. Also, users should be able to report a review if they believe it violates guidelines (e.g. contains personal attacks or sensitive info). Admins can remove reviews that violate terms (e.g. discriminatory language, etc.).

Should-have:

- **Category Ratings**: Allow guests to rate specific aspects separately (especially for stays): e.g. Cleanliness, Communication, Accuracy (of listing description), Location, Check-in, Value. These can be 1–5 stars each. The listing can show sub-rating averages for each category (this provides more nuanced feedback to hosts and info to guests).

- **Search/Filter by Reviews**: Allow potential guests to filter search results by rating (e.g. "4+ stars only") or at least sort by rating. Also possibly highlight keywords in reviews (e.g. if a user searches within reviews for "wifi", highlight that).
- **Guest Profile Reviews**: If hosts review guests, then if I as a host get a booking request from John, I should be able to see on John's profile if other hosts have rated or left comments (like "John was clean and followed house rules" or an internal thumbs up/down).
- **Anonymous or Private Feedback**: Provide an option for guests to leave some private feedback to the host or to the platform (like "The host was nice but the place had a weird smell – I don't want to publicly say this, but just for the host's knowledge"). This can help hosts improve without affecting their public rating.
- **Review Notifications**: Notify the host when a new review is posted for their listing. Notify the guest if the host responds to their review. If hosts review guests, notify the next host who gets a booking from that guest (like "Past host feedback: 5 stars").
- **Incentivize Reviews**: If needed, encourage reviews by sending reminders and perhaps showing progress (e.g. "You have 2 pending reviews to write"). Possibly, allow writing reviews via email link for convenience.
- **Display Logic**: If both sides have to review (double-blind), hold posting until both have submitted or the time window ends. If only one side submits, publish it when the window closes. Also, display reviews in an unbiased way (avoid only showing good ones – but maybe allow sorting by most helpful or critical).
- **Most Helpful Review**: Allow users to upvote or mark a review as helpful. Then optionally highlight the most helpful positive review and most helpful critical review at top (this helps balance perspective).

Good-to-have:

- **Summarized Ratings**: Show interesting aggregate info if available, e.g. "Cleanliness: 4.9 (based on 10 ratings), Communication: 5.0" or "95% of guests gave 5 stars".
- **Review Replies from Other Guests**: Possibly allow threaded comments on a review if someone has a question or if another guest found it useful (this can get messy, so often not done).
- **Integration with Social Proof**: Option for guests to share their review to social media (marketing for the platform).
- **Badges based on Reviews**: If a host consistently gets 5-star reviews and zero cancellations, the platform could award them a badge like "Superhost" (Airbnb style) to display on their profile/listings. Similarly, an experience provider with excellent reviews might get "Top Rated Guide" badge. (This uses review data to encourage excellence.)
- **AI Summary**: Use AI to summarize reviews on a listing – e.g. "Most guests mention that the place is sparkling clean and the location is convenient, though a few noted the Wi-Fi could be faster." (This is advanced but helps users get a quick gist).
- **Prevent Review Extortion**: Ensure hosts and guests know they can't promise incentives for good reviews or threaten for bad – include in guidelines and detect if possible.

Flow/UX:

1. **Review Submission (Guest)**: After checkout, the guest receives an email: "How was your stay at [Listing]?" and in the app a notification to review. They click it, going to a form. They select star rating, write a comment, and hit submit. A thank you message appears. If double-blind, they won't see host's review yet (if host wrote one).

2. **Review Submission (Host)**: Similarly, the host is prompted to review the guest. They might just select a star rating and a short note like "Left the place tidy, welcome back anytime." Host submits via their dashboard or a link.
3. **Publishing**: If both sides submitted, both reviews go live. If only guest did (common, many hosts won't write guest reviews each time), the guest's review goes live after the window (or immediately if not using double-blind for simplicity).
4. **Viewing Reviews**: A new user browsing the listing sees the reviews section. They might click "Read all reviews" if there are many. They can scroll. If category ratings are present, they see an overview (e.g. categories with star bars).
5. **Host Response**: If a host sees a review (especially a negative one) and wants to reply, they go to their host dashboard's reviews management, find that review and post a response. E.g. "We're sorry the Wi-Fi was slow; we have upgraded our router now. Thank you for the feedback!"
6. **Impact**: The listing's search ranking or conversion may be affected by reviews (great reviews might boost it in recommendations; very poor average might trigger quality checks or lower ranking).

Edge Cases:

- **No Reviews Yet**: New listing with no reviews – display "New listing - no reviews yet" rather than nothing. Possibly encourage early adopters by highlighting it's new (some guests like new listings for the novelty or maybe a discount).
- **Biased Reviews**: A guest or host might leave an unfair review (maybe out of anger for something unrelated). If it doesn't violate policy, it stays, but if a pattern of unfair reviews from a user emerges, platform might investigate.
- **Review Disputes**: If a host claims a review is false or retaliatory (e.g. they had to charge a guest for damage, guest leaves a lie in revenge), there should be a process via Customer Support to handle this. Possibly they can remove or edit a review in extreme cases, or at least append a note "This review is disputed by the host".
- **Inappropriate Content**: If a review contains hate speech, personal info, or other disallowed content, it should be removed or edited by admin. The system's moderation should catch many before posting (see AI moderation).
- **Guest Didn't Stay**: If a guest leaves early or has a horrible experience and leaves, they can still review because the booking was technically completed. If a guest never showed up (no actual experience), it's a grey area – they might still be allowed to review ("The host was unreachable, I never got in"), which is valuable info. But ensure clarity (maybe ask in review flow "Did you actually check in?").
- **Fraudulent Positive Reviews**: Collusion where hosts and guests exchange fake bookings just to review and boost ratings. Platform should monitor (e.g. a host getting multiple reviews from accounts that only ever booked that host could be suspicious).
- **Legacy Changes**: If a host significantly changes a listing (renovation or new management), old reviews might not reflect current state. Usually all reviews remain, but host can mention in description "newly renovated as of 2025" etc. Some platforms allow starting fresh, but that removes history and trust, so usually not.

7) Customer Support

Goal:

Provide users (guests and hosts) with timely support and effective solutions when issues or questions arise. This includes self-service resources (help center FAQs) and direct support channels (chat, email, phone) for resolving problems such as technical issues, booking disputes, or safety concerns. A strong support system enhances trust in the platform.

User Stories:

- As a **guest**, I want to easily find help if I have a problem with my booking or host (for example, I arrive and can't get into the apartment), so that I can quickly resolve it or get a refund.
- As a **host**, I want to contact the platform's support if I encounter an issue with a guest (like property damage) or need clarification on policies, so that I feel backed by the platform.
- As a **user**, I want to browse common questions and find answers (e.g. "How do I change my reservation dates?") on my own, so I might not even need to contact support if the answer is readily available.
- As a **user**, if I report something (like a message scam or an inappropriate listing), I want the platform to acknowledge it and take action, so the community stays safe.
- As a **guest**, if my host cancels last-minute or something goes terribly wrong, I want the platform to proactively assist me in finding alternative accommodation or provide compensation, so I'm not stranded.

Must-have:

- **Help Center / FAQ**: A well-organized online help center with articles covering common topics: Account & Profile, Booking & Payments, Cancellations & Refunds, Hosting & Listing management, Trust & Safety, etc. Users can search keywords or navigate categories to find answers.
- **Contact Form / Email Support**: If the FAQ doesn't solve it, users should be able to submit a support request. This could be a contact form within the app where they choose a topic, enter a description of the issue, attach images (optional), and submit. This creates a support ticket that goes to the support team's system (with an email confirmation to the user).
- **Response Time Communication**: The support interface should tell users roughly when they can expect a response (e.g. "We typically respond within 12 hours" or show live queue status if possible). This manages user expectations.
- **In-App Support Inbox**: Alternatively or additionally, users can view their open support requests in-app with status updates. For example, a "Support" section where they see "Issue #12345 – In Progress – assigned to agent Alice".
- **Issue Escalation**: Certain urgent categories (like "Safety issue" or "Host didn't show up") should trigger an urgent response workflow (maybe highlight to support staff, show a phone number to call for immediate help).
- **Reporting Mechanisms**:
- Guests/Hosts can report a message in chat (spam or abuse).
- Guests can report a listing (scam, inappropriate).
- Hosts can report a guest (if something bad happened). These reports go to the trust & safety team for review and are logged.

- **Dispute Resolution**: A defined process for handling disputes, e.g. a guest claims the property was not as described and wants a refund after check-in. The platform support mediates: collects info from both sides, checks evidence (photos, messages), and makes a decision per policy. There should be guidelines and tools for support to do things like partially refund, cancel without penalty, etc.
- **Cancellation Assistance**: If a host cancels on a guest last-minute, support should automatically reach out to help the guest re-book elsewhere (perhaps showing a list of similar available listings and offering to transfer the booking or offering a coupon). This requires human or automated help to ensure guest is taken care of.
- **24/7 Emergency Line**: At least for critical issues (like someone's personal safety at risk, or locked out late at night), provide a phone number staffed 24/7.

Should-have:

- **Live Chat Support**: An in-app live chat with a support agent (or chatbot triage). This provides immediate feedback. For example, a user opens live chat, initially a bot asks "Tell us your issue" then either provides an answer from FAQ or queues them to a human agent if needed. Agents can send links to help articles or perform actions (like initiating a refund).
- **Phone Support**: A customer support phone number for the platform that users can call during working hours (24/7 for emergencies). Phone support is resource-intensive but can resolve complex issues faster (and reassure users by hearing a human).
- **AI Chatbot / Virtual Assistant**: Use an AI to handle simple queries in chat or through the help interface. E.g. user types "I want to change my reservation dates" – the bot can respond with steps or even deep link to the date change function (if allowed) or the cancellation policy. If the question is complex, the bot hands off to a human.
- **Support for Multiple Languages**: Offer support in key languages depending on user base. This could mean having multilingual support agents or at least translating the help center articles. If a user submits a ticket in Spanish, ideally a Spanish-speaking agent responds.
- **Internal Support Tools**: (Back-end feature) Provide support agents with an admin panel to view all relevant info: user profiles, listings, booking details, payment transactions, message history (for disputes), etc., so they can assist efficiently. They should be able to take actions like issuing refunds, canceling bookings on behalf of users, applying credits, banning users, editing listings, etc. (Not user-facing, but crucial for support to function.)
- **Feedback on Support**: After a case is resolved, optionally ask the user to rate their support experience (thumbs up/down or star rating for the agent). Use this feedback to improve service.
- **Community Safety Programs**: e.g. a 24/7 safety line specifically for crisis (if something really bad like harassment or emergency at a property – could be same as emergency line mentioned).
- **Proactive Outreach**: If certain triggers happen (like host cancels last minute, or system detects a possible scam listing that a user booked), a support agent proactively contacts the user involved to check and assist, rather than waiting for them to complain.

Good-to-have:

- **Community Forum**: A place where users can ask general questions and get answers from other experienced users or moderators (this can deflect some support queries and build community, but needs moderation).
- **Guided Issue Flows**: Instead of just a freeform contact form, have guided wizards for common issues. E.g. "Problem with a booking" → choose sub-problem "Host not responding / Listing not as

described / Payment issue" → based on choice, show some immediate advice ("try contacting host via phone") and then option to escalate to support with pre-filled info.
- **Knowledge Base Search in App**: When typing in the support form "refund", the app could surface related FAQ articles in case that answers the question, before they submit.
- **Automated Refund Processing**: For simple cases (like guest cancels within full refund period), process it automatically without requiring human agent approval. The system already knows policy – support should mainly handle exceptions.
- **SLAs and Routing**: Use priority levels for tickets. E.g. "Trip in progress" issues tagged as high priority (agent responds in <15 minutes), whereas a general question might have a 12-24h SLA. The system can route to different teams (trust & safety team vs general support vs technical support for app bugs).
- **Integrated User Feedback Loop**: If an issue was due to a platform bug or policy gap, feed that back to product team to fix underlying cause (internal process, but ensures support isn't just firefighting).
- **Outage Handling**: In case of platform downtime or known issues (payment system outage, etc.), display a banner or status page link so users know it's a known problem. This can pre-empt a flood of support tickets.

Flow/UX:

1. **Self-Help Attempt**: User clicks "Help" in the app. They see a search bar and categories. They type "cancel booking" – an article "How do I cancel a booking?" appears. They read it; if satisfied, they might not need further help.
2. **Contacting Support**: If the user still needs help, they click "Contact us". They select a category and either start a live chat or fill a form. Suppose a guest's host isn't responding at check-in. The guest selects "Trip Issues -> Host No-Show". The app might immediately show the emergency phone number prominently and also allow sending a request.
3. **Live Chat**: If available, a chat agent joins and helps: maybe they call the host on behalf, while chatting with the guest. If resolved (host picks up, problem solved), great. If not, the agent might help the guest find alternate accommodation and process a refund.
4. **Ticket**: If no live chat, the user submits form. They get an email confirmation: "Your request (#4567) has been received." Internally, support staff sees it in their queue. Within a couple of hours, they respond (the user gets an email or app notification). The user and support agent can communicate back and forth via that ticket (either through email or an in-app support messaging interface).
5. **Resolution**: Support resolves the issue (could involve refunds, rebooking, technical advice, etc.). They mark the case closed in the system. The user gets a "We believe your issue is resolved. If not, you can reopen within X days" message.
6. **Follow-up**: Optionally, user might get "Please rate our support" or a short survey. Also, if it was a serious incident (like safety issue), support or trust team might follow up later to ensure user is okay or take further action against offending party.

Edge Cases:

- **High Volume**: During a crisis (say a widespread flight cancellation causing many rebookings), support might be overloaded. The system should show a notice if response times are longer and possibly direct users to alternate help (community forum or expanded FAQ).
- **Users not Tech-Savvy**: Some users might struggle to find support options. Ensure "Contact us" is visible in the app menu or help page, not hidden.

- **Language Barriers**: A user writes to support in a language the support agent doesn't speak. The agent might use translation tools internally or route to an agent who speaks that language.
- **Abusive Users**: Occasionally, users might abuse support (yelling at agents, or a fraudster contacting support to game refunds). Support agents should have tools to flag such users and policies to end abusive chats/calls politely and escalate to higher management or security team.
- **GDPR/Privacy Requests**: Users might contact support to delete their data or ask what data is held. There should be procedures to handle these legally.
- **Refund Complexities**: Multi-payment, partial refunds, currency conversions – support needs clear rules and possibly custom tool support to execute these correctly so that the user gets the right amount back.
- **Host-Guest Conflict**: In a "your word vs mine" scenario (e.g. guest says place was dirty, host says guest is lying), support has to judge based on evidence. The flow might include asking for photos from guest, checking host's history, etc. The outcome may not please one party. Having clear policy (like cleanliness guarantee or so) helps guide decision and communicate it.
- **Legal/Safety Emergencies**: If a serious incident occurs (theft, assault), support should advise the user to contact local authorities first (if not already), and the trust & safety team would take over to assist law enforcement and take platform actions (like banning user, etc.). Not an everyday support scenario but procedures must exist for worst cases.

AI Features

1) Personalized Recommendation Engine

Goal:

Leverage AI to provide personalized suggestions of homes and experiences to users, increasing engagement and booking conversion. By analyzing user preferences and behavior, Livana can surface listings that each user is most likely to be interested in (e.g. on the home page or as "Recommended for you" sections), thereby simplifying discovery and tailoring the platform to individual tastes.

How it works:

1. **Data Collection**: The system gathers data on user behavior and preferences:
2. Explicit data: Wishlists, favorites, past bookings, review ratings given, and profile info (e.g. a user indicates interests like "surfing" or "historical tours").
3. Implicit data: Browsing history (which listings a user viewed or clicked on), search queries, and how the user interacts with recommendations (clicks or ignores).
4. **Profile & Embeddings**: AI models create a "user profile vector" that encodes the user's tastes (for example, prefers entire homes, in beach locations, around $100/night, pet-friendly). Similarly, each listing (home or experience) is represented as a vector capturing its characteristics (location type, amenities, style, past popularity).
5. **Candidate Generation**: Using the user's profile and collaborative filtering:
6. The system finds listings that are popular among similar users (collaborative filtering identifies patterns like "users who booked X also liked Y").
7. It also pulls in content-based matches (e.g. if the user loved a cabin in mountains, find other mountain cabins; if they showed interest in food tours, get other food-related experiences).

8. Include trend-based items: new or highly rated listings in general, ensuring fresh content.
9. **Ranking**: A ranking algorithm or model scores the candidate listings for the user:
10. It considers the match to user's profile (how similar the listing vector is to the user's preference vector).
11. Also factors in diversity (don't show 10 extremely similar apartments in a row, mix it up a bit) and business logic (maybe boost new hosts a bit, or those with special promotions).
12. The output is a sorted list of recommended listings.
13. **Feedback Loop**: As the user interacts (clicks a recommended listing, or ignores/skips them, or explicitly thumbs-up/down), feed that back into the model to continuously improve recommendations. For example, if user consistently ignores luxury villas and clicks budget apartments, the system will adjust to favor cheaper options for that user.

Technology:

- **Machine Learning Models**:
- Use collaborative filtering algorithms (like matrix factorization or item-based similarity) for analyzing user-item interactions (bookings, views, ratings).
- Use content-based models, possibly deep learning or gradient boosted trees, that take listing features and user profile as input to predict affinity. Could use embeddings (e.g. via neural networks) to represent listings and users in the same vector space.
- Potential use of deep learning recommenders (like autoencoders or factorization machines) or even knowledge graphs to capture relationships.
- **Infrastructure**:
- A **Recommendation Service** that runs offline jobs to generate/update recommendations or on-demand scoring. It might use a combination of offline batch processing (e.g. updating collaborative filtering matrices nightly) and real-time components (re-rank on the fly based on current context).
- Data storage: user-item interaction data likely in a database or big data platform, and precomputed candidate lists in a fast data store (like Redis or ElasticSearch for quick retrieval).
- **Embedding computation**: Possibly share the embedding from the Semantic Search feature for listings, and create user embeddings by averaging embeddings of items they liked/booked. Alternatively, maintain separate learned embeddings via a recommendation model.
- **Integration**: The front-end will call an endpoint (e.g. "GET /recommendations") with the user ID and context, and the back-end returns a list of listing IDs with scores which the front-end then displays (perhaps as a carousel like "Recommended for you").
- **Privacy**: The model should ensure not to overly segment by sensitive attributes (like avoiding bias in recommending certain neighborhoods or prices based on profile in a discriminatory way). Focus on behavior-driven patterns.

KPI:

- **Click-Through Rate (CTR)** on recommended items: What percentage of recommended listings presented are clicked by the user.
- **Conversion Rate** from recommendations: How often do recommendations lead to a booking. This is the ultimate measure of success (e.g. X% of bookings come from recommendation section).
- **Engagement Time**: Users spend more time browsing because they find the recommendations appealing (could measure average session duration or number of listings viewed).

- **Diversity & Novelty**: The recommendations shouldn't always be the same or too narrow. Metrics like coverage (what fraction of listings get recommended across user base) or diversity score (variety in a single user's recommendations list) could be tracked.
- **Retention**: Users who receive good recommendations might come back more often. Compare retention or repeat usage of users engaged with recommendations vs those who don't.
- **Feedback**: If there's an explicit thumbs-up/down, track those rates (ideally more thumbs-ups). High negative feedback might indicate the model needs improvement.

2) Semantic Search

Goal:

Enable the search system to understand user queries in a natural, meaning-based way rather than relying solely on exact keyword matches. This means a user can type a descriptive phrase ("spacious loft near nightlife" or "family-friendly activities") and the search will return relevant results even if the exact words aren't in the listing. It also handles typos and synonyms, making search more robust and user-friendly.

How it works:

1. **Query Processing**: When a user enters a search query, the system first normalizes and interprets it:
2. Convert to lowercase, correct obvious typos ("appartmant" -> "apartment").
3. Remove or standardize special characters (e.g. "5-star" to "five star" maybe).
4. Identify the parts of the query: Some parts might be location names, some might be descriptive. (E.g. "paris romantic view" – detect "Paris" as location, "romantic view" as description).
5. (If the user filled separate location and filters fields, that structured data is used too. Semantic search mainly improves the free-text part.)
6. **Embedding Generation**: The processed query (or the descriptive portion of it) is passed through an embedding model, such as a neural network that outputs a vector representation (an array of numbers) capturing the semantic meaning of the query. For example, "cozy cottage with garden" might produce a vector close to that of listings which mention "charming country house with backyard" even if wording differs.
7. **Listing Embeddings**: Offline, every listing's content (title, description, amenities, etc.) is also processed through the same embedding model to generate a vector for each listing. These are stored in a vector database or index.
8. **Similarity Search**: The query vector is compared to all listing vectors to find those that are most similar (using cosine similarity or Euclidean distance). This yields a set of semantically relevant listings.
9. **Hybrid Ranking**: Combine semantic relevance with traditional filtering:
10. First, apply hard filters (e.g. location, availability dates, price range) to narrow down the candidate set.
11. Among those, use the semantic similarity scores to rank the results. Also incorporate traditional text match scores (BM25 or similar for exact keyword matches) for cases where the query does have specific words.
12. Example: If user searches "villa with swimming pool in Bali under $200", we ensure results are in Bali and <=$200 (structured filters), then among those, rank higher the listings whose descriptions or titles conceptually match "villa" and "swimming pool" (even if a listing says "has a private pool" it will match "swimming pool" conceptually).

13. **Typos and Synonyms**: Because of the semantic approach, a query like "beautifull view" (typo) or "beautiful scenery" should still find listings described as "stunning view" because the model learns that "beautiful" and "stunning" are related, etc.
14. **Continuous Improvement**: If using a machine learning approach, gather data on search success (which listings users click/book for certain queries) to fine-tune the model or at least validate that semantic results align with user intent.

Technology:

- **Natural Language Processing Models**: Use pre-trained language models or embedding models. For example:
- Transformer-based models like BERT or its variants (or OpenAI's embedding API) to encode text to vectors. Multilingual models could be used if the platform is in multiple languages.
- These models might be fine-tuned on platform-specific data (like known relevant listing-query pairs) to improve accuracy.
- **Vector Database**: Use a vector similarity search index (like **Faiss**, **Milvus**, **ElasticSearch with vector capability**, or Postgres with a **pgvector** extension) to store listing embeddings and query them quickly for nearest neighbors.
- **Search Engine**: Combine with a traditional search engine (Elasticsearch, Solr, or MeiliSearch). For example, Elastic could store listings with a text index and a dense vector field. A hybrid search could first filter by location/date using traditional queries and then use a script or plugin to rerank by vector similarity.
- **Infrastructure**: Possibly a dedicated **Search Service** that orchestrates all this – takes user query and filters, fetches candidates, ranks them, and returns results. Caching layer for common searches might be used.
- **Typo Correction**: Could use existing libraries or simple edit-distance algorithms for minor typos, or rely on the model's robustness. Additionally, maintain a synonym dictionary for important terms (e.g. "hot tub" ~ "jacuzzi", "downtown" ~ "city center") to assist if needed.
- **Evaluation**: Use human-labelled relevance data or online A/B tests to adjust weighting between semantic vs keyword match.

KPI:

- **Search Result Engagement**: Measure how often users click a listing from the search results (Click-through rate on search results). If semantic search is effective, CTR should improve for longer or more complex queries.
- **Conversion Rate from Search**: Ultimately, are users finding what they need? If semantic search works, we'd expect a higher percentage of searches result in a booking (or at least a favorited listing).
- **Refinement Rate**: Track if users immediately refine or rephrase their searches. A drop in "did not find what I want, search again" actions would indicate the first results were good. For instance, if previously many people who searched "cabin lake" had to then try "cottage lakefront", but now they find results in one go, that's a win.
- **Coverage of Queries**: Some queries that used to return zero results (due to no exact match) might now return relevant results with semantic search. Count how many queries get non-zero results now vs before.
- **Performance**: Maintain fast query response times (<300ms ideally even with embedding matching). Track the search latency and ensure the semantic step isn't too slow.

- **Quality Metrics**: If we have labeled data or can infer satisfaction, use metrics like Precision/Recall or Mean Reciprocal Rank (MRR) for search results on test queries to gauge relevance.

3) Fraud & Spam Detection (Moderation)

Goal:

Protect the platform from fraudulent activities and spam, ensuring safety and trust. AI is used to detect suspicious behavior or content proactively: - **Fraud**: Identify fake listings, scammers, or fraudulent transactions (e.g. stolen credit cards, hosts/guests with malicious intent). - **Spam/Scam Messages**: Detect and filter messages that try to take users off-platform (such as someone sending their phone and asking to pay off-platform) or other scam content, as well as general spam or abusive language. - **Content Moderation**: Ensure listings, profiles, and reviews do not contain inappropriate content (hate speech, adult content, etc.) and that they meet guidelines.

How it works:

1. **Listing Screening**: When a new listing is created or updated, an AI moderation service scans its content and metadata:
2. Scans description and title for red flags (e.g. "Western Union only" or too-good-to-be-true claims, or explicit banned terms).
3. Analyzes photos using computer vision to detect if images might be stolen (e.g. the same image appears on internet elsewhere), contain watermarks/contact info, or inappropriate imagery.
4. Gives a risk score. High-risk listings might be automatically suspended for manual review; medium-risk might be allowed but flagged to admins.
5. **User Behavior & Profiles**: AI looks at patterns:
6. User signup patterns (many accounts from one IP or device -> likely fraud farm).
7. Hosts with an unusually high cancellation rate or always asking guests to pay cash might be flagged.
8. Guests doing suspicious search patterns or trying many stolen cards.
9. The system can assign a "trust score" to each user that updates over time based on verified info and any suspicious signals.
10. **Messaging Monitoring**: As messages are exchanged, run them through NLP models:
11. Identify attempts to share contact info or external payment ("email me at …", "WhatsApp me on …", IBAN, etc.) especially before booking – flag or mask these.
12. Detect scam patterns (someone offering a secret discount if the user leaves the platform, etc.).
13. Detect harassment or hate speech (similar to toxic comment moderation) to intervene or warn users.
14. If a message is flagged as likely spam/scam, the system could warn the recipient ("This message looks suspicious") or temporarily block it and notify Trust & Safety team.
15. **Payment Fraud Detection**: Use machine learning (and third-party fraud detection if integrated, like Stripe Radar) to score each transaction:
16. Based on device fingerprint, geolocation of user vs card billing, velocity of transactions, etc., flag potentially fraudulent payments. Decline or hold such transactions for review.
17. If a particular user triggers multiple failed payment attempts or other users reported them, require additional verification or block them.
18. **Reviews Moderation**: When a review is submitted, run it through AI to filter out prohibited content (personal info, slurs, etc.). If the model flags it, either reject automatically or send to manual moderation queue.

19. **Ongoing Monitoring**: AI models continuously look at operational data:
20. Graph analysis to see if a cluster of guest accounts all reviewed each other's listings suspiciously (possible fake review ring).
21. Monitor if a listing suddenly gets many bookings and cancellations (could be a sign of someone trying to cash out stolen cards via refunds).
22. If something triggers (like many users reporting a listing or user), escalate for human investigation.

Technology:

- **NLP Classification**: Models like transformers or simpler logistic regression on text to classify messages, descriptions, reviews into categories: {spam, scam attempt, contains contact info, clean} and {toxic, off-topic, clean} etc. e.g. use OpenAI's moderation model or Google's Perspective API for toxicity; custom rules for contact info.
- **Computer Vision**: Use image recognition to detect text in images (OCR) so scammers can't bypass text filters by putting phone numbers in an image. Also use reverse image search APIs or perceptual hashing to see if listing photos are stolen from web or duplicates of another listing.
- **Anomaly Detection**: Unsupervised models on usage data (like an autoencoder for user activity patterns, or clustering to find outliers). Also rule-based checks (like "if one device creates 5 host accounts in a day -> flag").
- **Fraud Detection Systems**: Possibly integrate with third-party services (e.g. Sift Science, Stripe Radar, etc.) that specialize in detecting fraudulent patterns and give a score for each login or payment.
- **Moderation Pipeline**: All these checks feed into a central **Trust & Safety system**:
- It can auto-block content or users if certain thresholds are exceeded (e.g. a message 99% likely a scam -> don't deliver it, and notify support).
- It provides a dashboard for safety agents to review flagged items and take actions (ban user, remove listing, etc.).
- **Machine Learning**: Continuously train the models with feedback. For example, if some scam messages got through and were later reported by users, those transcripts can be used to improve the classifier.

KPI:

- **Detection Rate**: How effective are we at catching bad stuff? e.g. Percentage of known fraud cases we caught proactively vs ones reported by users. (We want a high proactive catch rate.)
- **False Positives**: We also monitor that we are not wrongly blocking legitimate content/users. E.g. measure how many listings are flagged but turn out fine (should be low after tuning). Similarly monitor user complaints about "my listing was unfairly suspended."
- **Time to Action**: For flagged issues that require human review, how fast does the team act? (Especially critical for things like scam listings – ideally we remove them before anyone books).
- **Reduction in Incidents**: As AI systems improve, things like chargeback rates (fraudulent transactions leading to card chargebacks) should decrease, and reports of spam/scam from users should drop.
- **User Trust Metrics**: Indirectly, maintaining a low rate of incidents contributes to positive user surveys about trust/safety on the platform.
- **Host/Gast Verification**: Perhaps track the percentage of users verified and correlation with incidents – more verified, fewer issues.

- **Scam Funnel Interruption**: For messaging, measure how often attempted sharing of contact info is prevented. If before AI 5% of pre-booking chats had an email/number exchange, and now it's 1% with masking, it's working (assuming that correlates to preventing off-platform deals).
- **Community Health Index**: Could aggregate various safety metrics into a composite.

Overall Data Flow and Technology Stack

Architecture & Tech Stack:

- **Front-End**: Livana provides a responsive web application (SPA) built with a modern framework like **React** (possibly Next.js for SSR for performance/SEO) along with HTML/CSS (perhaps Bootstrap or Tailwind for styling). This delivers the main user interface for desktop and mobile web. For mobile users, dedicated **iOS and Android apps** could be built (either natively Swift/Kotlin or using cross-platform tech like React Native or Flutter) to offer push notifications, offline access, etc. The front-end communicates with the back-end via RESTful API calls or GraphQL.
- **Back-End**: A server-side application handles business logic and serves the APIs. This could be built in **Node.js** (with Express or NestJS for structured MVC) using **TypeScript** for type safety. Alternatively, any scalable web framework (Python Django/Flask, Java Spring Boot, etc.) can be used – but Node.js fits well for real-time features. The back-end is structured possibly as microservices:
- **Auth Service** (manages user accounts, JWT tokens, sessions, OAuth).
- **Listing Service** (CRUD for listings, search indexing).
- **Booking Service** (availability, reservations, payments).
- **Messaging Service** (real-time chat via WebSocket, message storage).
- **Review/Rating Service**.
- **Recommendation & Search Service** (could be separate or integrated).
- These services communicate over a network (could use an API gateway or direct internal APIs). For an MVP, it might be a single monolithic application that modularly handles these domains.
- **Database**: Use a **relational database** like **PostgreSQL** for core data (users, listings, bookings, reviews) because of the need for complex queries and transactions (especially for booking atomicity and consistency). Postgres is reliable and supports spatial data (for location coordinates) and extensions like **PostGIS** if needed for geo queries.
- Additionally, use **Redis** for caching frequently read data (like cached search results, session store if not using JWT, or rate-limiting counters) to improve performance and handle high read loads.
- For the messaging system, a combination of Redis (for Pub/Sub or in-memory chat queue) and a persistent store (could be the SQL DB or a NoSQL like MongoDB for flexibility) might be used to store chat history.
- **Search Engine**: For robust search and filtering, integrate **Elasticsearch** (or **MeiliSearch** as a lightweight alternative). This handles text queries, geospatial queries (e.g. find listings within X km of a location), and aggregations for filters. Each listing is indexed in Elastic with fields for location, price, availability (maybe using a next-available-date field for search on open dates), and text (title, description).
- The **semantic search** feature will either use Elastic's vector capabilities (it now supports approximate nearest neighbor on vectors) or a separate vector database. Each listing would also store an embedding vector, and query vectors will be matched to them.
- **AI/ML Infrastructure**:
- Some AI tasks like recommendations can be done offline. Possibly use Python with libraries (Pandas, Scikit-learn, PyTorch) for model training. For serving, you might have a **Recommendation Service**

that loads the trained model or precomputed data and provides an API for real-time recommendations.

- Similarly, a **Moderation Service** might call external APIs (like OpenAI or Perspective) or run its own models to score text and images. This could be a microservice that other parts of the system call whenever content is created.
- **Model Hosting**: If using custom ML models, they could be served via a service (using something like TensorFlow Serving or an AWS SageMaker endpoint).
- **Real-Time Communication**: Implement WebSockets for features like messaging and live notifications. For example, use **Socket.io** (Node.js) or a pub/sub system. The application might have a **Notification Service** that pushes events (new message, booking status update) to the relevant users' sockets and also to push notification services.
- **Payments**: Integrate with a payment gateway. **Stripe** is a common choice for handling payments and payouts (Stripe Connect for marketplace payouts to hosts). The back-end will integrate with Stripe's API to create charges, handle webhooks for payment events (successful charge, payout sent, etc.). All sensitive payment info is tokenized (no card details stored on our servers).
- **Cloud and DevOps**: The app is deployed on a cloud platform (e.g. **AWS**, **Azure**, or **GCP**). Likely using containerization (**Docker**) and orchestration (**Kubernetes**) for scalability and management, especially if microservices are used. For an MVP, could start on simpler PaaS (like AWS Elastic Beanstalk or Heroku) and then migrate to full Kubernetes when needed.
- Services like AWS can provide **S3** for storing images (listing photos, user avatars) and a CDN to serve them quickly worldwide.
- Use a managed database service (like Amazon RDS for Postgres) for reliability and backups.
- Set up load balancers to distribute traffic across multiple server instances for high availability.
- **Logging & Monitoring**: Implement centralized logging (with tools like Elastic Stack/ELK, or cloud logging services) and monitoring (Prometheus/Grafana for metrics, or DataDog/NewRelic etc.) to keep track of system health. Monitor key metrics like request rates, error rates, response times, etc., so issues can be identified early.
- **Security**: Follow best practices – e.g. all API calls over HTTPS, proper input validation and use of prepared statements/ORM to prevent SQL injection, rate limiting on APIs to prevent abuse, storing secrets (like API keys for Stripe or map APIs) securely (in env vars or secret manager, not in code).
- Use JWT for auth tokens (store minimal info in token, verify signature on each request). Possibly use refresh tokens and short-lived access tokens for security.
- Implement content security policies on front-end to prevent XSS, and use libraries to sanitize any user-generated content that might be displayed.
- **3rd Party Integrations**:
- **Map services**: Use Google Maps or Mapbox API for geocoding addresses (turning an address into lat/long) and for displaying maps on the listing page and search page.
- **Email & SMS**: Integrate an email service (like SendGrid) for sending verification emails, booking confirmations, etc. For critical or realtime messages (e.g. host last-minute cancel), possibly send SMS via Twilio.
- **Push Notifications**: Use Firebase Cloud Messaging (FCM) for mobile push notifications to the apps (for chat messages, booking updates).
- **Social auth**: Google/Facebook sign-in SDKs.
- Possibly **third-party analytics** (Google Analytics or Mixpanel) to track user behavior flows and conversion funnels (e.g. how many users search -> view listing -> initiate booking -> complete booking).

Data Flow:

The platform operates through multiple layers – front-end, back-end services, databases – all interacting cohesively. Below is an overview of how data flows for key operations: 1. **User Actions (Client):** When a user interacts (searches, books, messages) on the web or app, the client sends requests via API to the server. For example, clicking "Search" sends a request to the Search API with query and filters; clicking "Book" sends a request to the Booking API with listing ID and dates. 2. **Processing (Server):** The API endpoints in the back-end receive these requests, authenticate them (check JWT token), and then execute the business logic: - For a search query: The Search service queries the Elasticsearch index (and possibly the vector index) with the parameters, fetches results, ranks them, and returns listing summaries to the client. - For a booking: The Booking service will check availability (query the database or cache), create a tentative booking record, call the Payment gateway to charge the user's card, wait for confirmation, then finalize the booking record and update the listing's availability in the database. It may also enqueue tasks (e.g. send confirmation email, notify host). - For messaging: The client either opens a WebSocket connection or polls an API. When a message is sent, the Message service writes it to the DB and also publishes it via WebSocket to the recipient in real-time. It might also call the Moderation service to scan the content. 3. **Database Operations:** Almost every action involves reading or writing to the database: - Registration writes a new user row in the Users table. - Listing creation writes to Listings table and triggers an index update in Elasticsearch and storing photos in S3. - Bookings write to Bookings table (with foreign keys to users and listings, etc.), and maybe to a Transactions table for payments. - Reviews write to Reviews table and may update an aggregate rating in the Listings table. The back-end uses transactions where needed (especially around booking + payment to ensure consistency). 4. **Asynchronous Tasks:** Some work is done asynchronously to keep user-facing requests fast. A task queue (like **RabbitMQ** or a Redis-based queue or cloud service like AWS SQS) is used: - After a booking is confirmed, a task is queued to send out emails or push notifications (so the API can respond to user immediately and do email in background). - Nightly jobs: e.g. generate fresh recommendation data, re-index search data, purge old data, send review reminders. - Moderation tasks: If AI needs more time, could process content slightly delayed and take action (e.g. listing goes live immediately but gets taken down 10 minutes later if flagged). 5. **External Services Interaction:** The back-end integrates with external APIs: - It calls the Payment gateway (over HTTPS) to execute payments. - It uses geocoding API when a host enters an address (get lat/long and maybe place details). - It calls AI services (maybe our own microservices or external ones) for things like generating recommendation or scanning text. - All these calls are usually made in specific services and the results are either returned immediately or saved for later use (e.g. store geocode results to reuse). 6. **Real-Time & Notifications:** For events like new message or booking status update: - The server side (e.g. Booking service) will create a notification entry in a Notifications table or directly push via WebSocket to the user's session ("booking_confirmed" event with details). - If the user is offline, a push notification via FCM or an email is sent. - Similarly, the messaging system uses WebSocket events to instantly relay messages, and falls back to push notification if receiver is offline. 7. **Frontend Update:** The client, upon getting responses or realtime events, updates the UI: - Search results are displayed to the user. - After booking API responds success, the UI shows a confirmation page. - If a WebSocket message arrives (new chat or notification), the app will show a pop-up or update the relevant part of the interface. - State management ensures consistency (for instance, once a listing is booked, if the user tries to book it again or if they have that listing open in another tab, the system should inform them it's no longer available on those dates). 8. **Admin & Support Workflows:** There's also an admin interface where support staff can intervene: - They can lookup a booking, and possibly trigger a refund or cancellation through the system (which goes through the same Payment integration to refund). - If they ban a user or remove a listing, those changes propagate: e.g. a banned user's token is invalidated, their listings maybe marked inactive in DB and removed from search index.

Overall, data flows in a cycle: user input triggers server logic, which reads/writes data and possibly calls external services, then results flow back to users or other parts of the system. The design ensures consistency (using transactions and careful ordering of operations), scalability (using caching, indexing, and asynchronous tasks to handle load), and maintainability (with a modular architecture and clear data contracts between services). By leveraging a robust tech stack and clear data pipelines, Livana aims to provide a smooth, reliable experience similar to Airbnb, under the hood of which all these components work in concert.