Mastering System Design

Design an Online Rental Platform (aka Airbnb)

Understanding the Problem & Key Actors

- “Design an Online Rental Platform where property owners can list accommodations and users can search, book, and pay for short-term stays.”
- Core Concepts:
  a. Listings (properties, availability, pricing)
  b. Search and filters
  c. Bookings and calendar synchronization
  d. Payments and guest reviews
  e. Media uploads (images, videos)
- Who Are the Key Actors?
  a. Guest: Searches for and books accommodations
  b. Host: Lists properties, manages availability and pricing
  c. Admin: Moderates content, manages flagged users/listings
  d. Payment Gateway: Processes secure transactions
  e. External Calendar: Syncs host calendars (Google Calendar, iCal, etc.)

Functional Requirements

- Account creation & login (Host/Guest)
- Host can:
  - Create & update listings
  - Upload media
  - Set pricing and availability
- Guest can:
  - Search listings with filters
  - View listing details & reviews
  - Book listings and make payments
- Admins can manage users and moderate content
- Calendar sync with external platforms
- Notification system (email, push)

Non-Functional Requirements

- High Availability: 24/7 uptime, especially during peak travel seasons
- Scalability: Handle millions of users & listings
- Security: Payment security, personal data privacy
- Performance: <300ms response for key operations like search
- Reliability: Consistent booking logic, no double-booking
- Localization: Multiple currencies, time zones, and languages

Assumptions and Constraints

- Payments are handled via a 3rd-party gateway
- Reviews are moderated by the platform
- Users must verify email before booking/listing
- Media (images/videos) is uploaded to cloud object storage
- Users use web and mobile apps (we need REST APIs)
- Real-time search, but bookings can have slight delays (eventual consistency for availability)

Estimating Scale – Users, Listings, and Bookings

- Daily Active Users (DAU): ~5 million globally
- Concurrent Users: ~100k at peak
- Listings: 50 million+ (growing daily)
- Searches per day: 40–50 million
- Bookings per day: ~1 million
- Media per listing: Avg. 10 images, some videos
- Payment transactions/day: ~1 million (sync with bookings)

Implications:

  - High volume of reads (search, view listings)
  - Frequent writes (bookings, availability updates)
  - Large media storage and delivery
  - Real-time sync needed (calendar, availability)

Data Size & Storage Needs

- Rough Estimations:
  - Listings DB: ~50M listings × 5 KB = ~250 GB
  - Bookings DB: ~1B records/year × 1 KB = ~1 TB+
  - Media storage: 10 images × 1 MB × 50M = ~500 TB
  - User profiles + history: ~5M DAU × 0.5 KB = ~2.5 GB/day
- Hot Paths:
  - Searching listings (frequent reads)
  - Availability/calendar updates (write-heavy)
  - Booking confirmation (atomic write + payment + calendar update)
- Cold Paths:
  - Reviews history
  - User profile edits
  - Admin moderation

Identifying System Bottlenecks and Challenges

- Search Service
  - Handles very high query volume (QPS)
  - Needs fast, filtered, geo-based search
  - Requires scalable indexing and distributed querying
- Availability Calendar
  - Frequently updated due to bookings and sync with external calendars (iCal, Google Calendar, etc.)
  - Requires consistency to avoid double-bookings
  - Time zone management adds complexity
- Booking System
  - Requires atomic operation: availability lock + payment + confirmation
  - Needs to prevent race conditions during peak traffic
  - May benefit from queues or transactions to ensure reliability
- Media Storage & Delivery
  - Huge storage needs (images/videos for millions of listings)
  - High bandwidth and performance requirements for content delivery
  - Needs CDN caching and object storage (e.g., S3, Azure Blob)
- Payment Integration
  - High dependency on third-party APIs (Stripe, PayPal, etc.)
  - Must be fault-tolerant and secure (PCI compliance)
  - Needs retry logic, logging, and fallbacks

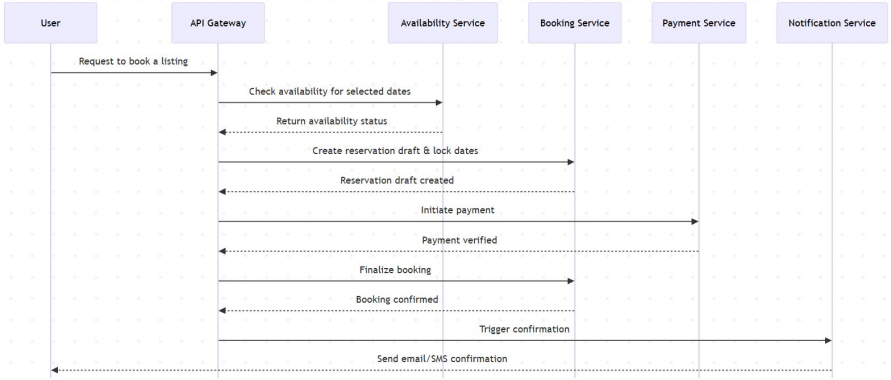
High-Level Architecture Overview and Key components

- **Frontend (Web & Mobile Apps)**: User-facing clients for searching, booking, and managing listings
- **API Gateway**: Entry point for all client requests, Handles routing, authentication, rate limiting, and request shaping
- Core Microservices
  - **User Service**: Handles sign-up, login, profiles, preferences
  - **Listing Service**: Manages property listings (details, images, amenities)
  - **Search Service**: Indexes listings for geo-based, filtered search
  - **Availability Service**: Manages booking calendar and date availability
  - **Booking Service**: Handles reservation lifecycle and booking flow
  - **Payment Service**: Integrates with payment providers, tracks payments
  - **Notification Service**: Email, SMS, and push notifications
  - **Review & Ratings Service**: User feedback, property ratings
- Support Services
  - **Media Service**: Upload & store images/videos (via S3, CDN)
  - **Calendar Sync Service**: Syncs with external calendar providers
  - **Analytics & Logging Service**: Tracks user events and system health

Service Interactions (Example: Booking Flow)

- Step 1: User requests to book a listing
- Step 2: Availability Service checks if dates are open
- Step 3: Booking Service locks availability, creates a reservation draft
- Step 4: Payment Service initiates and verifies payment
- Step 5: Upon success, Booking Service finalizes booking
- Step 6: Notification Service sends confirmation email/SMS

All interactions are asynchronous where possible (e.g., payment confirmation, notifications) using message queues (e.g., RabbitMQ, Kafka).



Communication Patterns & APIs

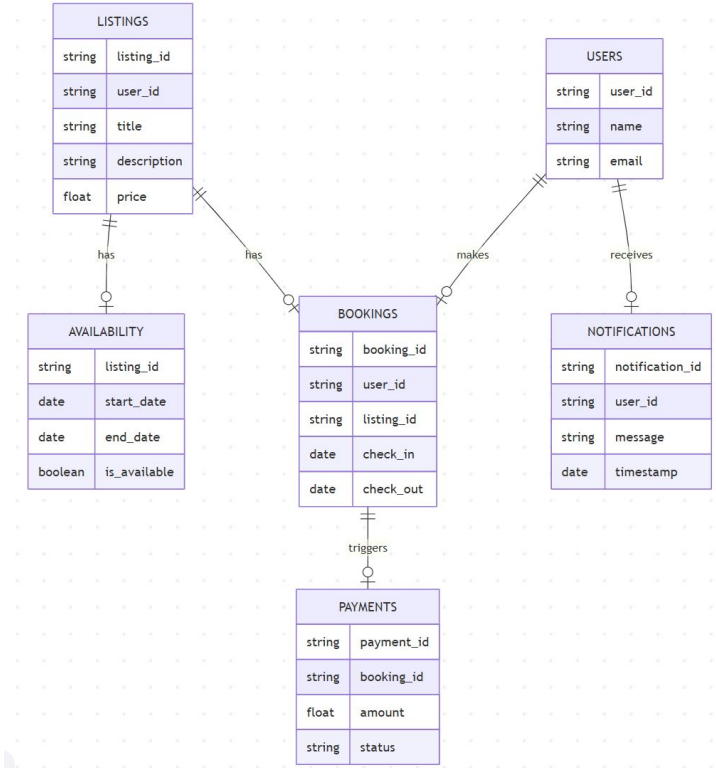
- Sync Communication (REST/gRPC):
  - Used for user interactions, search, listing fetch
  - APIs exposed via API Gateway
- Async Communication (Message Queue/Event Bus):
  - Booking events (e.g., booking_created, booking_failed)
  - Notifications, calendar syncs, email dispatches
  - Payments webhook events
- Authentication & Authorization
  - OAuth2/JWT-based tokens
  - Role-based access: guest, host, admin

Data Storage & Indexing Strategy

- Primary Datastores
  - Relational DB (PostgreSQL/MySQL) for transactions, user & listing data
  - NoSQL (MongoDB/DynamoDB) for availability snapshots, reviews
- Search Index
  - Elasticsearch for full-text + geo search on listings
- Caching Layers
  - Redis for hot data (recent searches, popular listings)
  - CDN (Cloudflare/Akamai) for images and static content

Sample DB Schema for Online Rental Platform

- Users
- Listings
- Availability
- Bookings
- Payments
- Notifications
- Media



Strategic Tech & Infra Decisions

- Scalability
  - Use Azure for flexible scaling and horizontal scaling across multiple servers.
- Redundancy & High Availability
  - Leverage load balancers and multi-region database replication for redundancy and availability.
- Data Storage
  - Store transactional data in SQL and media assets in Azure Blob Storage for durability and cost-effectiveness.
- Asynchronous Communication
  - Use RabbitMQ or Kafka for asynchronous tasks like payments and notifications to decouple services.
- Caching
  - Integrate Redis for caching frequently accessed data to improve response time and reduce load.
- Microservices Architecture
  - Adopt microservices to scale and maintain services (e.g., booking, payment, availability) independently.
- Monitoring & Logging
  - Use Azure Monitor for health tracking and centralized logging (ElasticSearch) for real-time diagnostics.

The Final Design - Online Rental Platform