Databases II
Workshop No. 2 — Data System Architecture and Information Retrieval

Team Members:
Santiago Sanchez Moya – 20211020032
Juan Daniel Rodriguez Hurtado – 20211020152
Faider Camilo Trujillo Olaya – 20192020136
Computer Engineering Program
School of Engineering
Universidad Distrital Francisco José de Caldas
Professor: Eng. Carlos Andrés Sierra, M.Sc.

Full-time Adjunct Professor

Semester 2025-III

Contents

1. Introduction

This workshop represents the starting point of the course project for *Databases II.* The project focuses on designing and implementing a real-world, data-intensive application for a property rental platform that connects **hosts** who offer accommodations with **guests** seeking short-term stays.

The main goal of the project is to provide a secure, reliable, and efficient system that supports essential operations such as property listings, bookings, payments, and user management. The platform aims to simplify the accommodation process by enabling guests to easily discover and book properties, while ensuring that hosts can manage their listings, track earnings, and receive payouts automatically after completed bookings.

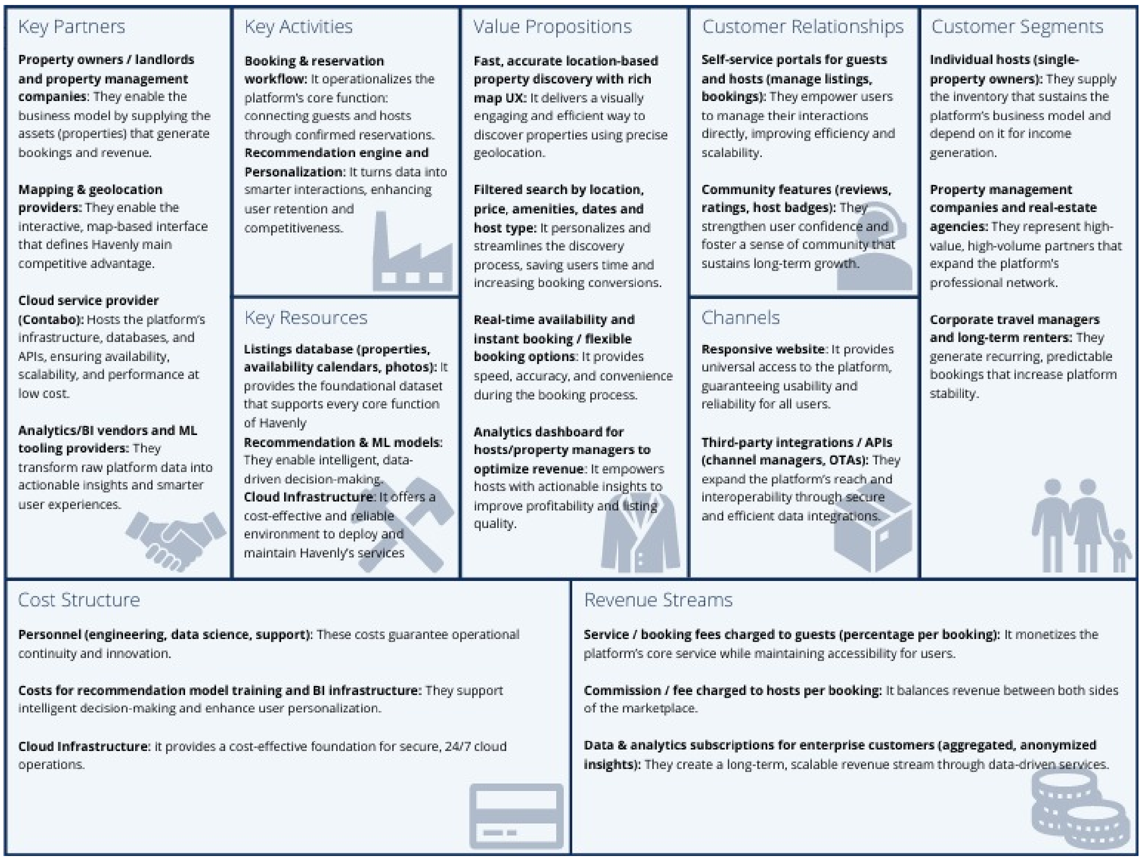
The system addresses the need for transparency, trust, and automation in the property rental market, reducing manual intervention and potential errors in booking and payout management. The core users include:

- **Guests:** Individuals searching for short-term stays.
- **Hosts:** Property owners or managers listing accommodations.
- **Administrators:** System operators ensuring platform compliance, content moderation, and security.

The primary objectives of this project are:

- To model the data requirements and relationships for the property rental domain.
- To define user stories and functional requirements aligned with business needs.
- To design the initial database architecture as the foundation for system implementation.

2. Business Canvas Model



Key Partners

Property owners / landlords and property management companies: They enable the business model by supplying the assets (properties) that generate bookings and revenue.

Mapping & geolocation providers: They enable the interactive, map-based interface that defines Havenly main competitive advantage.

Cloud service provider (Contabo): Hosts the platform's infrastructure, databases, and APIs, ensuring availability, scalability, and performance at low cost.

Analytics/BI vendors and ML tooling providers: They transform raw platform data into actionable insights and smarter user experiences.

Key Activities

Booking & reservation workflow: It operationalizes the platform's core function: connecting guests and hosts through confirmed reservations.
Recommendation engine and Personalization: It turns data into smarter interactions, enhancing user retention and competitiveness.

Key Resources

Listings database (properties, availability calendars, photos): It provides the foundational dataset that supports every core function of Havenly
Recommendation & ML models: They enable intelligent, data-driven decision-making.
Cloud Infrastructure: It offers a cost-effective and reliable environment to deploy and maintain Havenly's services

Value Propositions

Fast, accurate location-based property discovery with rich map UX: It delivers a visually engaging and efficient way to discover properties using precise geolocation.

Filtered search by location, price, amenities, dates and host type: It personalizes and streamlines the discovery process, saving users time and increasing booking conversions.

Real-time availability and instant booking / flexible booking options: It provides speed, accuracy, and convenience during the booking process.

Analytics dashboard for hosts/property managers to optimize revenue: It empowers hosts with actionable insights to improve profitability and listing quality.

Customer Relationships

Self-service portals for guests and hosts (manage listings, bookings): They empower users to manage their interactions directly, improving efficiency and scalability.

Community features (reviews, ratings, host badges): They strengthen user confidence and foster a sense of community that sustains long-term growth.

Channels

Responsive website: It provides universal access to the platform, guaranteeing usability and reliability for all users.

Third-party integrations / APIs (channel managers, OTAs): They expand the platform's reach and interoperability through secure and efficient data integrations.

Customer Segments

Individual hosts (single-property owners): They supply the inventory that sustains the platform's business model and depend on it for income generation.

Property management companies and real-estate agencies: They represent high-value, high-volume partners that expand the platform's professional network.

Corporate travel managers and long-term renters: They generate recurring, predictable bookings that increase platform stability.

Cost Structure

Personnel (engineering, data science, support): These costs guarantee operational continuity and innovation.

Costs for recommendation model training and BI infrastructure: They support intelligent decision-making and enhance user personalization.

Cloud Infrastructure: it provides a cost-effective foundation for secure, 24/7 cloud operations.

Revenue Streams

Service / booking fees charged to guests (percentage per booking): It monetizes the platform's core service while maintaining accessibility for users.

Commission / fee charged to hosts per booking: It balances revenue between both sides of the marketplace.

Data & analytics subscriptions for enterprise customers (aggregated, anonymized insights): They create a long-term, scalable revenue stream through data-driven services.

Figure 1: Business Canvas Model.

3. Requirements Documentation

3.1. Functional Requirements (FR)

FR1. **User Registration & Authentication**

FR1.1 Users must be able to register as guest or host.

FR1.2 Users must be able to login/logout with secure password handling.

FR1.3 Hosts must be able to verify their properties (upload info, photos).

FR2. **Property Management (Hosts)**

FR2.1 Hosts must be able to create, update, and delete property listings.

FR2.2 Hosts must manage availability calendars for their properties.

FR2.3 Hosts must be able to view bookings and revenue history.

FR3. **Property Discovery (Guests)**

FR3.1 Guests must filter results by price range, amenities, and availability.

FR3.2 Guests must be able to view property details, reviews, and host profile.

FR4. **Booking & Reservations**

- FR4.1 Guests must be able to book available properties for selected dates.
- FR4.2 Users must not be able to double book for the same property/date.
- FR4.3 Users must receive booking confirmations.

FR5. **Payments**

- FR5.1 Guests must be able to pay for bookings using a payment gateway.
- FR5.2 Hosts must receive payouts after bookings are completed.

FR6. **Reviews & Ratings**

- FR6.1 Guests must be able to leave reviews and ratings after a stay.
- FR6.2 Hosts must be able to respond to reviews.

FR7. **Admin Functions**

- FR7.1 Admins must manage users (ban, verify, reset).
- FR7.2 Admins must manage reported properties and reviews.

3.2. Non-Functional Requirements (NFR)

NFR1. **Performance**

- NFR1.1 The system must return property search results within 2 seconds for at least 95% of user requests.
  Justification: Based on an estimated 50 concurrent users for this regional prototype. With indexed geospatial queries in PostgreSQL (PostGIS) and a Contabo 2 vCPU / 4 GB RAM VPS, this response time is achievable.
- NFR1.2 Booking confirmations must be processed in 3 seconds or less from payment submission to confirmation.
  Justification: Stripe and PayPal APIs average 1–1.5 s latency; additional processing time covers backend validation and message delivery.

NFR2. **Scalability**

- NFR2.1 The system must support up to 100 concurrent users without a response-time increase greater than 10%.
  Justification: Value scaled down from Airbnb's global user base (275M users) to a realistic regional prototype scope.
- NFR2.2 The database must handle 50,000 property records and 250,000 booking entries while maintaining query performance under 3 seconds.
  Justification: This dataset represents a feasible subset of Airbnb's scale, achievable on PostgreSQL with spatial indexing.

NFR3. **Availability**

NFR3.1 The platform must maintain 99.0% uptime, allowing no more than 7 hours of downtime per month.
Justification: Contabo's SLA guarantees 99.9% uptime; the extra margin accounts for local maintenance activities.

NFR3.2 Scheduled maintenance windows must not exceed 2 hours per month and should occur during off-peak hours.
Justification: Minimizes user disruption while enabling necessary updates.

NFR4. **Security**

NFR4.1 All client-server communication must use HTTPS/TLS 1.2 or higher.
Justification: Required for PCI DSS compliance when processing payments.

NFR4.2 The system must perform input validation on all user forms and API endpoints (100%).
Justification: Prevents SQL injection, XSS, and other common web vulnerabilities.

NFR5. **Usability**

NFR5.1 The web interface must fully load within 3 seconds on a 10 Mbps internet connection.
Justification: Matches Google Lighthouse's performance benchmarks for responsive websites.

NFR5.2 A user must be able to complete the full booking process in five clicks or fewer from the search results page.
Justification: Reduces interaction friction and improves conversion, consistent with Nielsen usability principles.

NFR6. **Maintainability**

NFR6.1 The database schema documentation must be updated within 48 hours of any structural modification.
Justification: Keeps technical documentation synchronized with the system's actual design.

3.3. Requirements Definition Approach

For the functional requirements, we initially defined them from a user-centered perspective to capture a holistic view of platform interactions. Once the global vision was established, we refined and reduced their scope to match the project's achievable range. Conversely, the non-functional requirements were redefined to transform generic expectations into measurable, testable, and technically justified performance and quality targets.

4. User Story Prioritization using the MoSCoW Method

The following table summarizes the prioritization of user stories for the application according to the MoSCoW method.

User Story	**MoSCoW Category**	**Justification**
User Registration	**Must-have**	It is essential for platform functionality, as users must register to access booking and account management features.
User Authentication	**Must-have**	Ensures secure access to personal data and complies with fundamental security standards.
Property Discovery with Filters	**Must-have**	Core functionality for guests to search and locate properties according to their preferences and location.
Booking	**Must-have**	Central to the business model—users must be able to reserve accommodations for the platform to function.
Booking Confirmation	**Should-have**	Adds value by confirming reservations, though it could initially be handled manually or through notifications.
Guest Payments	**Must-have**	Critical for processing transactions and ensuring booking commitments between hosts and guests.
Guest Reviews	**Should-have**	Enhances trust and transparency in the platform but is not required for the MVP.
Host Payouts after Completed Bookings	**Should-have**	it's important for host satisfaction and retention, but the platform can temporarily function with manual payout processing if needed.
Property Creation	**Must-have**	Allows hosts to list their accommodations, an indispensable feature for platform operation.
Property Update	**Should-have**	Facilitates host management and data accuracy, but the system can operate initially without frequent updates.
Property Deletion	**Could-have**	Offers convenience to hosts to manage listings but is not critical for the first release.
Host Availability Calendar	**Should-have**	Prevents double bookings by showing only available dates, yet can be implemented in later iterations.
Host Revenue and Booking History	**Could-have**	Provides useful analytics for hosts but has minimal impact on the core functionality.
Host Respond to Reviews	**Won't-have (this release)**	Enhances communication but is postponed to a future release to prioritize essential features.
Admin User Management	**Must-have**	Ensures platform control, security, and compliance by allowing administrative user management.
Admin Moderation of Properties and Reviews	**Should-have**	Contributes to content quality and trust but can be integrated after the core system is stable.

Table 1: User Story Prioritization using the MoSCoW Technique

5. Effort Estimation using the Planning Poker Technique

To estimate the development effort for each user story, the **Planning Poker** technique was applied. This agile estimation method combines expert judgment, consensus building, and relative sizing to produce reliable effort estimates expressed in *story points*.

The process was conducted collaboratively by the development team, considering factors such as technical complexity, implementation risk, dependencies, and required testing. The Fibonacci sequence (1, 2, 3, 5, 8, 13, 21) was used as the reference scale, as it reflects the natural increase in uncertainty as stories become larger or more complex.

Each team member assigned a point value to the user story; discussions were held to reconcile differences, and a consensus value was established. The results are presented in Table 2.

User Story	**Story Points (Fibonacci)**	**Justification**
User Registration	3	Moderate complexity: requires form validation, input control, and database integration.
User Authentication	5	Involves token management, encryption, and secure session handling.
Property Discovery with Filters	8	Requires data querying, filter logic, and dynamic rendering of multiple attributes.
Booking	8	Includes managing availability, dates, and data integrity between users and hosts.
Booking Confirmation	5	Needs integration with the booking system and email or notification services.
Guest Payments	13	High complexity due to payment gateway integration and transaction validation.
Guest Reviews	5	Moderate logic for CRUD operations and relational linkage between users and properties.
Host Payouts after Completed Bookings	13	It equires simple business logic but integration with multiple existing components (bookings, payouts, status tracking) without being trivial.
Property Creation	8	Involves form management, media uploads, and database persistence.
Property Update	5	Similar logic to creation, but requires conditional updates and validation.
Property Deletion	3	Simple CRUD operation with basic confirmation logic.
Host Availability Calendar	8	Requires calendar UI, date validation, and synchronization with booking logic.
Host Revenue and Booking History	5	Involves data aggregation and basic reporting functions.
Host Respond to Reviews	2	Simple response feature, low complexity, limited business impact.
Admin User Management	8	Involves role-based access control and CRUD operations for multiple user types.
Admin Moderation of Properties and Reviews	5	Requires approval workflows and review state management.

Table 2: Effort Estimation using the Planning Poker Technique

6. User Stories

The following tables present the main user stories for different roles: Guest, Host, and Admin.

Title	User Registration
Priority	Mo
Estimate	3
User Story	As new a User, I want to create a new account either as a user or as a host. with my personal information, So that, I can access the platform's booking features.
Acceptance Criteria	Given I'm on the registration page, when I submit valid email, password, and required personal information, then my account should be created.

Table 3: User Registration

Title	User Authentication
Priority	Mo
Estimate	5
User Story	As a Registered user, I want to log in securely with my credentials So that I can access my account and personal data.
Acceptance Criteria	Given valid credentials, when I log in, then I should access my dashboard

Table 4: User Authentication

Title	Property Discovery with Filters
Priority	Mo
Estimate	8
User Story	As a Guest, I want to search for properties using location and filters so that I can find the most suitable accommodation.
Acceptance Criteria	Given the gest is on the search page. When they apply filters (price, amenities, dates), then the system must display matching properties on the map.

Table 5: Property Discovery with Filters

Title	Booking
Priority	Mo
Estimate	8
User Story	As a Guest, I want to book a property for selected dates so that I can reserve my accommodation.
Acceptance Criteria	Given I select available dates, when I proceed to book, then a booking should be created with "pending" status.

Table 6: Booking confirmation

Title	Booking confirmation
Priority	S
Estimate	5
User Story	As a Guest, I want to receive confirmation when my booking is approved so that I can secure my stay.
Acceptance Criteria	Given my booking is confirmed by the host, then I should receive an email confirmation.

Table 7: Booking confirmation

Title	Guest Payments
Priority	Mo
Estimate	13
User Story	As a Guest, I want to pay for my booking through a secure payment gateway so that I can guarantee my reservation.
Acceptance Criteria	Given I have selected a property and confirmed dates. When I provide payment details and submit, then the application must process the payment securely and confirm the booking.

Table 8: Guest Payments

Title	Guest Reviews
Priority	S
Estimate	5
User Story	As a Guest, I want to leave a review and rating after my stay so that I can help other gests make better decisions.
Acceptance Criteria	Given I have completed a verified booking. When I submit a review with rating, then the application must link the review to the booking and display it on the property page.

Table 9: Guest Reviews

Title	Host Payouts after Completed Bookings
Priority	Must-have
Estimate	13
User Story	As a Host, I want to receive automatic payouts once a guest's booking is successfully completed, so that I can get compensated for my accommodation service without manual intervention.
Acceptance Criteria	Given a booking is marked as `Completed`, the system must automatically trigger the host payout.

Table 10: Host Payouts after Completed Bookings

Title	Property creation
Priority	Mo
Estimate	8
User Story	As a Host, I want to create new property listings, So that I can offer my accommodation on the platform
Acceptance Criteria	Given the Host is logged in. When they add property details with photos, then the system must validate, save and display the listing in search results.

Table 11: Property creation

Title	Property Update
Priority	S
Estimate	5
User Story	As a Host, I want Update my property information, so that I can keep listings accurate and attractive.
Acceptance Criteria	Given I edit property details, when I save changes, then they should be reflected immediately

Table 12: Property update

Title	Property Deletion
Priority	Co
Estimate	3
User Story	As a Host, I want to remove properties from the platform, So that I can manage my active listings.
Acceptance Criteria	Given I delete a property, then it should be removed from search results.

Table 13: Property Deletion

Title	Host Availability Calendar
Priority	S
Estimate	8
User Story	As a Host, I want to manage an availability calendar for my properties so that Guests cannot book unavailable dates.
Acceptance Criteria	Given I have created a property. When I update the availability calendar, then Guests must only see available dates when booking.

Table 14: Host Availability Calendar

Title	Host Revenue and Booking History
Priority	Co
Estimate	5
User Story	As a Host, I want to view my bookings and revenue history so that I can track my earnings and occupancy.
Acceptance Criteria	Given I am logged in. When I access the booking history page, then the application must display all past bookings and payouts.

Table 15: Host Revenue and Booking History

Title	Host Respond to Reviews
Priority	W
Estimate	2
User Story	As a Host, I want to respond to reviews so that I can engage with guests and maintain my reputation.
Acceptance Criteria	Given a Guest has left a review. When I writes a response and submits, then the response must appear below the Guest's review.

Table 16: Host Respond to Reviews

Title	Admin User Management
Priority	Mo
Estimate	8
User Story	As an Admin, I want to ban, verify, and reset user accounts so that I can ensure platform security and compliance.
Acceptance Criteria	Given the I'm logged into the dashboard. When I select a user account and choose an action (ban, verify, reset), then the application must apply the changes immediately.

Table 17: Admin User Management

Title	Admin Moderation of Properties and Reviews
Priority	S
Estimate	5
User Story	As an Admin, I want to manage reported properties and reviews so that I can maintain quality and trust in the platform.
Acceptance Criteria	Given a property or review has been reported. When the I reviews the report, then I can approve, remove, or take further action on the content.

Table 18: Admin Moderation of Properties and Reviews

6.1. Improvements in the Current user stories

The revised user stories demonstrate significant improvements over the initial version by adhering to agile best practices for clarity and manageability. The most critical enhancement is the decomposition of compound stories into singular, focused narratives; for instance, the original "Guest Registration and Login" has been separated into distinct "Guest Registration" and "User Authentication" stories. This eliminates ambiguity and allows for more accurate prioritization and development. Furthermore, each story now follows a strict, three-line structure for the "As a / I want to / So that" format, which sharpens the focus on the user's role, goal, and underlying motivation. The introduction of a standardized framework—including MoSCoW prioritization and story point estimates—provides a consistent mechanism for the team to evaluate the backlog, while the acceptance criteria have been expanded to be more specific, testable, and comprehensive, covering both success and failure scenarios to ensure a higher quality deliverable.

7. Initial Database Architecture

7.1. High-level Architecture

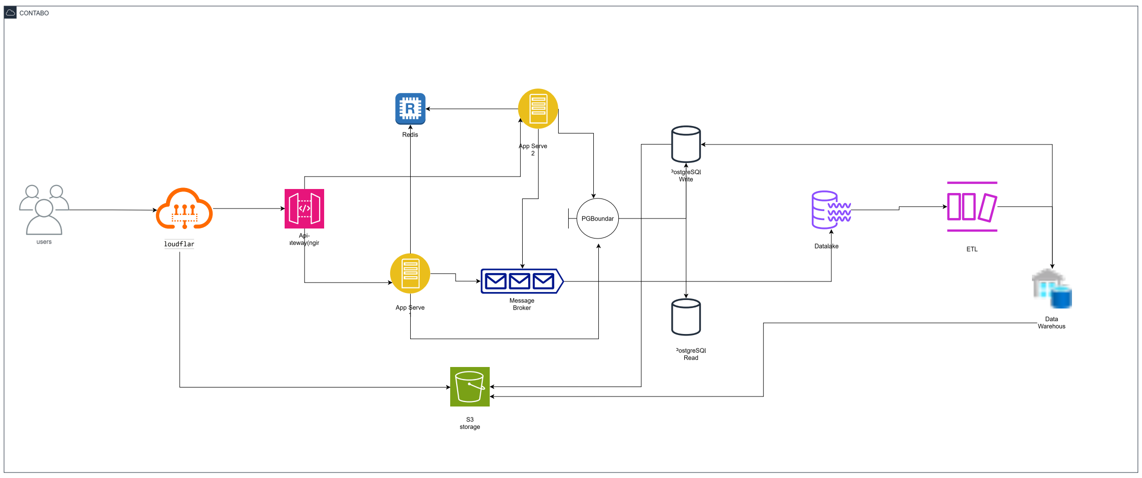


Figure 2: High-level database architecture.

The revised architecture represents a scalable and modular system designed for an Airbnb-style accommodation rental platform. It was restructured based on the previous feedback to improve clarity, rationale, and component justification. The new diagram now presents a clean separation between application services, data storage, and analytical layers, ensuring that each element has a defined role and purpose.

Key Components and Their Roles:

- **Users (Web Browsers / Mobile)**:The end users who interact with the web or mobile application. They generate concurrent traffic (searches, bookings, messages). The system must deliver content quickly and support 1000+ simultaneous users. With Cloudflare (requests pass through the CDN/WAF). Indirectly with App Servers (via the API Gateway) for dynamic actions (bookings, messages). The main entry point; user behavior drives all scalability, caching, and security decisions.

- **Cloudflare (CDN + WAF)**: A content delivery network and web protection layer (edge layer). Reduces latency for static resources, filters malicious traffic

(WAF), mitigates DDoS, and terminates TLS. It improves global performance without increasing VPS usage. Serves static files from S3 (origin cache). Forwards dynamic traffic to the API Gateway (preserving TLS, applying security rules, rate limiting). Essential for good user experience and security while keeping costs low — ideal for deployment on economical VPS providers like Contabo.

- **API Gateway (Nginx) — Public Layer**: The public gateway for all APIs; manages routing, authentication, rate limits, and logging. Centralizes policies (auth, throttling, CORS), acts as the single entry point for the backend, and protects real servers. Receives traffic from Cloudflare. Routes (and balances if needed) to App Server 1/2 or a load balancer. With many concurrent users, it's essential for access control, security, and routing to backend instances.

- **App Server 1 App Server 2 (Backend / API Servers)**: Identical backend instances that implement business logic (search, bookings, messaging, payment processing).

  Why there are two?: High availability and horizontal scalability — to distribute load and avoid a single point of failure. Duplication is necessary to handle concurrency and required latency; it decouples presentation and persistence logic.

  - Receive requests from the API Gateway.
  - Use Redis for caching and session storage.
  - Send asynchronous tasks to the Message Broker.
  - Query/write through PgBouncer to PostgreSQL.
  - Upload/download files to S3 (images, verification documents).
  - Send events to the Data Lake (via broker or jobs) for analytics.

- **Redis**: In-memory data store for caching and session management. Reduces database load, speeds up frequent queries (e.g., filters, cached search results), and stores sessions when sticky sessions are not desired.

  App Servers read/write to Redis (cached queries, sessions, temporary tokens).

  Can also be used by WebSocket/realtime components to publish state updates. Improves p95/p99 latency performance when many users are browsing or searching.

- **Message Broker (Queues — e.g., RabbitMQ**: Messaging system for asynchronous tasks (email delivery, image processing, notification generation). Decouples long-running tasks from HTTP requests to maintain low API latency.

  App Servers enqueue jobs (upload processing, notifications, payment integrations).

  Workers (or App Servers themselves) consume the queue to process background tasks.

  The Broker can also feed the Data Lake (event streams) if configured. Essential for handling workload peaks without blocking HTTP threads, ensuring eventual consistency (e.g., sending booking confirmations).

- **PgBouncer (Connection Pooler)**: A lightweight connection pooler between App Servers and PostgreSQL. PostgreSQL cannot efficiently handle thousands of open connections; PgBouncer maintains a controlled number of connections to the DB, preventing overload.

  Improves performance and stability under high concurrency. A practical requirement when scaling backend on VPS infrastructure without managed, auto-scaling databases.

- **PostgreSQL Write**: La base de datos transaccional principal donde se realizan las escrituras (reservas, usuarios, pagos). Mantiene la consistencia ACID para las reservas, los pagos y los estados de las reservas (evitando el overbooking).

  - Receives connections from PgBouncer.
  - Replicates data to PostgreSQL Read (read replica).
  - Exports data to the Data Lake (via dump/CDC) and backups to S3.
  - Can send logs/metrics to observability systems.
  - The transactional core of the system; essential for booking integrity.

- **PostgreSQL Read (Replica)**: A replica optimized for read-intensive workloads (listings, searches). Separates heavy read traffic from the primary database to avoid impacting critical writes.

  App Servers performing heavy queries can read from here.

  ETL or analytics processes consume from the replica to avoid affecting the primary DB. Improves search performance and reduces contention on the main database.

- **S3 Storage (Contabo Object Storage)**:

  S3-compatible object storage. Stores images (properties), documents (verification), database backups, and frontend static files (build artifacts).

  - Cloudflare caches assets originating from S3 (Cloudflare $\rightarrow$ S3 for public assets).
  - App Servers upload/read objects (host uploads, image downloads).
  - Postgres backups and Data Lake snapshots are stored in S3.
  - ETL/Analytics can read raw files directly from S3.
  - Cost-effective, durable storage for media and backups; separates storage from compute resources.

- **Data Lake (Raw Data, in S3)**:

  Raw data volume (logs, events, table copies, uploads) for analytics. Storing raw data enables building ETL pipelines without impacting production systems; also provides historical data for ML and reporting. App Servers / Broker / Jobs write events/files to the Data Lake.

The ETL orchestrator reads from the Data Lake to transform and load into the Data Warehouse. Backups and snapshots can also be versioned within the Data Lake. Separated from OLTP; essential for analytics, demand forecasting, and SLA metrics.

- **Data Lake (Raw Data, in S3)**: Orchestrator of pipelines that extract, transform, and load data from Data Lake / Databases → Data Warehouse. Centralizes and automates ETL processes (data cleaning, joins, aggregations, snapshots for BI).

  Takes data from Data Lake or Postgres Read. Loads processed data into the Data Warehouse. Executes scheduled jobs (daily/hourly) and can write metrics to monitoring systems. Essential for generating dashboards, cohorts, churn predictions, and business KPIs (bookings, occupancy).

- **Data Warehouse (ClickHouse / OLAP)**:

  A data warehouse optimized for analytical queries and reporting. Aggregated queries and dashboards (e.g., occupancy by city, LTV, conversion rate) must be fast and not impact the OLTP system.

  ETL loads the Data Warehouse from the Data Lake / Postgres Read.

  Separation between transactional and analytical layers; enables business intelligence without affecting production performance.

7.2. Entity-Relationship Diagram

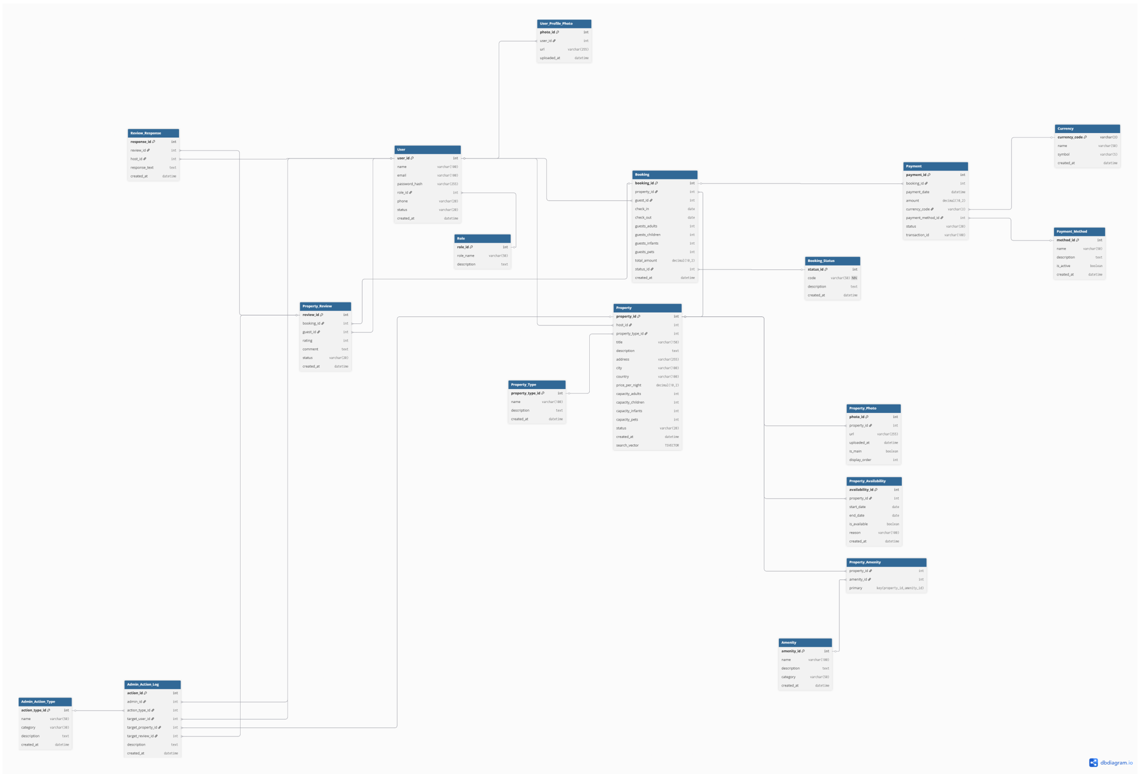


Figure 3: Entity-Relationship (ER) diagram of the database.

The Entity–Relationship (ER) diagram represents the logical structure of the system's database, which supports the main business processes of the platform: user management, property management,listing and discovery, booking and payments, and post-stay reviews. The model is thought to avoid redundancy and ensure referential integrity through foreign keys. Each group of entities is described and justified below.

User Management

- **User**: Central entity representing any system participant (guest, host, or admin). It stores essential identification data (name, email, hashed password, phone, and status). The field `role_id` determines access privileges and enables modular user management.
- **User_Profile_Photo**: Keeps the URL of the user's profile image in a separate table to maintain file management decoupled from the core user data. The one-to-one relationship ensures each user has a single active profile image, which improves clarity and supports potential media service integrations.
- **Role**: Defines user roles such as *guest*, *host*, or *admin*. This table allows role scalability (e.g., adding "superadmin" or "support") without altering user logic.
- **Relationships**:
  - **User–Role:** Many-to-One. Each user is assigned exactly one role, while each role can be shared by many users.
  - **User–User_Profile_Photo:** One-to-One. Each user has one photo entry, ensuring consistent linking of media data.
- **Justification:** This modular separation improves data normalization, reduces duplication, and allows flexibility for authentication and authorization mechanisms.

Property Management

- **Property**: Represents a listing created by a host. It includes details such as title, description, location, price, and capacity attributes (adults, children, infants, pets). The field `host_id` connects it to the user who owns the listing.
- **Property_Type**: Categorizes properties (e.g., Apartment, Cabin, Villa) and supports filtering in search functionalities.
- **Property_Photo**: Manages property images independently to allow multiple photos per listing. The inclusion of `is_main` and `display_order` supports image prioritization and user interface consistency.
- **Property_Availability**: Records periods when a property is available or blocked. This enables hosts to manually restrict dates, complementing availability derived automatically from bookings.

- **Amenity** and **Property_Amenity**: Define a many-to-many relationship between properties and their available features (WiFi, kitchen, parking, etc.). The join table `Property_Amenity` ensures flexibility and avoids data repetition.
- **Relationships**:
  - **Property–User:** Many-to-One (each host can own multiple properties).
  - **Property–Property_Type:** Many-to-One.
  - **Property–Property_Photo:** One-to-Many.
  - **Property–Property_Availability:** One-to-Many.
  - **Property–Amenity:** Many-to-Many (via Property_Amenity).
- **Justification:** This structure minimizes redundancy, allows flexible management of property metadata, and supports scalability for new property categories or features. The inclusion of availability records and full-text search (`search_vector`) enhances booking efficiency and search performance.

Booking and Payments

- **Booking**: Represents a reservation made by a guest for a specific property, including check-in/out dates, number of guests, and total amount. The foreign key `status_id` links to a dynamic list of states (pending, confirmed, cancelled, completed), improving system extensibility.
- **Booking_Status**: Enumerates booking lifecycle states to maintain referential consistency and allow standardized status handling.
- **Payment**: Stores financial transactions associated with bookings. It records the amount, currency, payment method, transaction identifier, and payment status.
- **Payment_Method** and **Currency**: Define controlled vocabularies for available payment methods and currencies, supporting multi-currency operations and compliance with financial APIs.
- **Relationships**:
  - **Booking–Property:** Many-to-One (each booking belongs to one property).
  - **Booking–User:** Many-to-One (guest makes bookings).
  - **Booking–Booking_Status:** Many-to-One.
  - **Booking–Payment:** One-to-One (each booking generates one payment record).
  - **Payment–Currency:** Many-to-One.
  - **Payment–Payment_Method:** Many-to-One.
- **Justification:** This structure ensures atomicity of financial transactions and clear traceability between reservations and payments. It supports both real-time processing and future extensions for refunds or payout automation.

Reviews and Responses

- **Property_Review**: Allows guests to leave a review after a completed booking, including rating and comment fields. The relation to `Booking` guarantees that only verified stays generate reviews, improving credibility.
- **Review_Response**: Enables hosts to respond to reviews, maintaining a transparent communication channel between both parties.
- **Relationships**:
  - **Property_Review–Booking:** One-to-One (each booking can generate one review).
  - **Property_Review–User:** Many-to-One (guest authoring the review).
  - **Review_Response–Property_Review:** One-to-One.
  - **Review_Response–User:** Many-to-One (host posting the response).
- **Justification:** The review structure enforces referential integrity, prevents duplicate evaluations per stay, and aligns with user experience goals of post-stay feedback and trust-building between guests and hosts.

In summary, the ER model ensures a normalized, scalable, and maintainable schema aligned with the application's functional requirements. Each relationship was carefully defined to balance data consistency, system performance, and future extensibility for features such as payouts, notifications, and administrative auditing.

7.3. Improvements in the Current Relational Model

The current relational model introduces several structural and conceptual improvements that enhance normalization, readability, and alignment with database best practices. First, non-structural entities such as *Capacity* and *Admin_Action* were removed or redefined to ensure that only persistent and meaningful data is represented. The *Admin_Action* entity, for example, was deleted because it considered that it was unnecessary. The *User_Photo* entity was renamed to *User_Profile_Photo* and linked through a one-to-one relationship to improve clarity and enforce unique profile pictures per user. Capacity-related attributes were integrated directly into the *Property* table, reducing unnecessary joins and simplifying queries. The introduction of *Booking_Status* as a separate table improves data integrity by standardizing booking states instead of relying on free-text fields. Likewise, the *Currency* entity now uses the ISO 4217 code as a primary key to ensure consistency with financial systems. Additional improvements include the addition of attributes like *search_vector* in *Property* to support full-text search, and explicit many-to-many relationships through junction tables such as *Property_Amenity.* Overall, these refinements make the model more coherent, modular, and scalable for future application features.

8. Information Requirements

8.1. Guest Information

User Profiles:
Includes personal data such as name, email, and authentication credentials. Supports personalized access and secure login.
Linked to: User Registration, User Authentication.

Property Listings:
Contains property title, description, price, location, amenities, and availability. Enables guests to search and filter listings efficiently.
Linked to: Property Discovery with Filters.

Booking Records:
Stores booking details such as selected property, dates, price, and status. Ensures tracking of reservations and confirmations.
Linked to: Booking, Booking Confirmation.

Payment Data:
Includes transaction ID, amount, method, and timestamp. Used to validate payments and maintain booking integrity.
Linked to: Guest Payments.

Guest Reviews:
Consists of ratings and textual feedback linked to verified bookings, improving trust and transparency.
Linked to: Guest Reviews.

8.2. Host Information

Property Data:
Comprises property details, images, and pricing defined by hosts. Allows property creation, updates, and visibility on the platform.
Linked to: Property Creation, Property Update.

Availability Calendar:
Shows available and booked dates to prevent double reservations.
Linked to: Host Availability Calendar.

Revenue and Booking History:
Summarizes total earnings, completed bookings, and payout information for hosts.
Linked to: Host Revenue and Booking History.

Content Management Data:
Includes all host actions such as listing edits and updates, ensuring data consistency and accuracy.
Linked to: Property Management and related operations.

8.3. Administrator Information

User Management Data:
Covers user roles, verification status, and administrative actions (ban, reset, verify).
Linked to: Admin User Management.

Moderation Records:
Contains data about reported properties or reviews, reasons for reporting, and moderation outcomes.
Linked to: Admin Moderation of Properties and Reviews.

Analytics and Reports:
Includes platform metrics such as number of users, bookings, revenue, and activity rates. Used for evaluating performance and strategic decisions.
Linked to: Business Model – Key Activities (Analytics and BI).

8.4. System-Generated Information

Recommendations:
Automatically generated property suggestions based on user preferences, previous searches, and location data.
Linked to: Value Proposition – Personalized Discovery.

Notifications:
Covers booking confirmations, payment receipts, and review updates automatically sent to users.
Linked to: Booking Confirmation, Guest Payments.

Audit Logs:
Records system and user actions for transparency, debugging, and compliance.
Linked to: Admin Operations and Security Management.

9. Query Proposals

9.1. 1. Guest Information

9.1.1 User Profiles

PostgreSQL Query:

```
SELECT
    u.user_id,
    u.name,
    u.email,
    u.status,
    u.created_at
FROM
    User u
JOIN
    Role r ON u.role_id = r.role_id
WHERE
    r.role_name = 'guest'
    AND u.status = 'active'
ORDER BY
    u.created_at DESC;
```

Purpose: Retrieves recently registered and active guest users. **Insight:** Supports user monitoring and personalized onboarding.

Redis Query:

```
GET user:123:profile
```

Purpose: Fetches cached user profile data for quick login and session restoration.

—

9.1.2 Property Listings

PostgreSQL Query:

```
SELECT
    p.property_id,
    p.title,
    p.price_per_night,
    p.country
FROM
    Property p
WHERE
    p.city = 'Bogot '
    AND p.price_per_night BETWEEN 100000 AND 250000
    AND p.status = 'active';
```

Purpose: Retrieves available properties within a given price range and location. **Insight:** Powers search and filter functionality in guest discovery.

Redis Query:

```
ZRANGEBYSCORE listings:bogota 100000 250000
```

Purpose: Quickly returns property IDs cached for real-time search operations.

—

9.1.3 Booking Records

PostgreSQL Query:

```
SELECT
    b.booking_id,
    u.name AS guest_name,
    p.title AS property_title,
    b.check_in,
    b.check_out,
    bs.code AS booking_status
FROM
    Booking b
JOIN
    User u ON b.guest_id = u.user_id
JOIN
    Property p ON b.property_id = p.property_id
JOIN
    Booking_Status bs ON b.status_id = bs.status_id
WHERE
    bs.code = 'CONFIRMED';
```

Purpose: Tracks confirmed bookings with associated guest and property details.
Insight: Enables occupancy analysis and performance metrics.

Redis Query:

```
HGETALL booking:current:1234
```

Purpose: Retrieves cached booking information for immediate user dashboard access.

—

9.1.4 Payment Data

PostgreSQL Query:

```
SELECT
    p.payment_id,
    p.booking_id,
    p.amount,
    pm.name AS method,
    p.status,
    p.payment_date
FROM
    Payment p
JOIN
    Payment_Method pm ON p.payment_method_id = pm.method_id
WHERE
    p.payment_date >= NOW() - INTERVAL '30 days';
```

Purpose: Fetches recent payments for revenue tracking and reconciliation. **Insight:** Supports financial audits and refund management.

Redis Query:

```
INCRBYFLOAT total_revenue 150.50
```

Purpose: Updates cached total revenue in real-time analytics dashboards.

—

9.1.5 Guest Reviews

PostgreSQL Query:

```
SELECT
    r.review_id,
    u.name AS guest,
    p.title AS property,
    r.rating,
    r.comment
FROM
    Property_Review r
JOIN
    Booking b ON r.booking_id = b.booking_id
JOIN
    User u ON r.guest_id = u.user_id
JOIN
    Property p ON b.property_id = p.property_id
WHERE
    r.rating >= 4
    AND r.status = 'approved';
```

Purpose: Identifies top-rated stays and guest experiences. **Insight:** Enhances trust and supports property ranking algorithms.

—

9.2. 2. Host Information

9.2.1 Property Data

PostgreSQL Query:

```
SELECT
    property_id,
    title,
    price_per_night,
    status
FROM
    Property
WHERE
    host_id = 42;
```

Purpose: Displays all properties managed by a specific host. **Insight:** Enables property management and performance tracking.

—

9.2.2 Availability Calendar

PostgreSQL Query:

```
SELECT
    property_id,
    start_date,
    end_date,
    is_available
FROM
    Property_Availability
WHERE
    property_id = 42;
```

Purpose: Shows availability and blocked dates for host's property. **Insight:** Prevents double-booking and aids scheduling.
Redis Query:

```
SADD calendar:42 "2025-10-20" "2025-10-21" "2025-10-22"
```

Purpose: Quickly updates cached availability dates when bookings are confirmed.
—

9.2.3 Revenue and Booking History

PostgreSQL Query:

```
SELECT
    u.user_id AS host_id,
    SUM(pay.amount) AS total_revenue,
    COUNT(b.booking_id) AS total_bookings
FROM
    Booking b
JOIN
    Property p ON b.property_id = p.property_id
JOIN
    User u ON p.host_id = u.user_id
JOIN
    Payment pay ON b.booking_id = pay.booking_id
WHERE
    u.user_id = 42
GROUP BY
    u.user_id;
```

Purpose: Calculates host's total earnings and booking count. **Insight:** Supports payout reports and dashboard summaries.
—

9.2.4 Content Management Data

PostgreSQL Query:

```
SELECT
    a.action_id,
    u.name AS admin,
    a.action_type,
    a.target_type,
    a.target_id,
    a.created_at
FROM
    Admin_Action_Log a
JOIN
    User u ON a.admin_id = u.user_id
WHERE
    a.target_type = 'Property'
ORDER BY
    a.created_at DESC;
```

Purpose: Tracks administrative edits and actions related to properties. **Insight:** Ensures accountability and auditability.

—

9.3. 3. Administrator Information

9.3.1 User Management Data

PostgreSQL Query:

```
SELECT
    u.user_id,
    u.name,
    r.role_name,
    u.status
FROM
    User u
JOIN
    Role r ON u.role_id = r.role_id
WHERE
    r.role_name IN ('host', 'guest');
```

Purpose: Retrieves user roles and their active statuses. **Insight:** Supports administrative user management.

—

9.3.2 Analytics and Reports

PostgreSQL Query:

```
SELECT
    COUNT(DISTINCT u.user_id) AS total_users,
    COUNT(DISTINCT b.booking_id) AS total_bookings,
    SUM(p.amount) AS total_revenue
FROM
    Payment p
JOIN
    Booking b ON p.booking_id = b.booking_id
JOIN
```

```
    User u ON b.guest_id = u.user_id;
```

Purpose: Generates key performance indicators for the platform. **Insight:** Enables data-driven decision-making and business analysis.

Redis Query:

```
GET dashboard:active_users
```

Purpose: Retrieves real-time count of currently active users.

—

9.4. 4. System-Generated Information

9.4.1 Recommendations

PostgreSQL Query:

```
SELECT
    p.property_id,
    p.title
FROM
    Property p
WHERE
    p.city = 'Medell n'
ORDER BY
    p.price_per_night ASC
LIMIT 5;
```

Purpose: Returns five recommended properties based on location and pricing. **Insight:** Supports personalized property discovery.

Redis Query:

```
LRANGE recommendations:user:123 0 -1
```

Purpose: Retrieves cached recommendation list for instant display.

—

9.4.2 Notifications

Redis Query:

```
PUBLISH notifications:user:123 "Your booking #456 is confirmed
   !"
```

Purpose: Sends instant notifications using Redis Pub/Sub channels. **Insight:** Provides real-time system feedback to users.

—

9.4.3 Audit Logs

PostgreSQL Query:

```

SELECT
    aal.action_id,
    a.name AS admin_name,
    u.name AS reviewer_name,
    r.comment AS review_text,
    at.name AS action_type,
    aal.description AS reason,
    aal.created_at
FROM Admin_Action_Log aal
JOIN User a ON aal.admin_id = a.user_id
JOIN Admin_Action_Type at ON aal.action_type_id = at.
   action_type_id
LEFT JOIN Property_Review r ON aal.target_review_id = r.
   review_id
LEFT JOIN User u ON r.guest_id = u.user_id
WHERE at.category = 'REVIEW_MODERATION'
ORDER BY aal.created_at DESC;
```

Purpose: Retrieves all property-related administrative actions performed in the last 14 days. **Insight:** Helps security and compliance teams trace modifications or deletions made to property data by administrators.

10. References

References

[1] IBM. (2025, July 25). *What is a data architecture?*. IBM Think. Disponible en: https://www.ibm.com/think/topics/data-architecture

[2] Corporate Finance Institute. (s. f.). *Business Model Canvas Examples*. Disponible en: https://corporatefinanceinstitute.com/resources/management/business-model-canvas-examples/

[3] Autor(es). (Año). *Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications*. Editorial/Editorial académica. *[Completar con autor y año cuando estén disponibles]*.

[4] Autor desconocido. (s. f.). *[Figura sobre arquitectura de datos]*. Substack. Disponible en: https://substackcdn.com/image/fetch/f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Feb15f23f-a99f-406b-add4-eaffe0d1633d_2332x2696.png

[5] YouTube. (2025, September 27). *Hotel Reservation (AirBnb, Booking.com) - System Design Interview Question* [Video]. YouTube. Disponible en: https://www.youtube.com/watch?v=m67Mjbx6DMY

[6] Bocconi University. (s. f.). *The rise of the sharing economy: Estimating the impact of Airbnb on the hotel industry*. IGIER Bocconi Working Paper No. 684. Disponible en: https://igier.unibocconi.eu/sites/default/files/media/publication/684.pdf

[7] Radical Storage. (s. f.). *Airbnb Statistics: Key Figures and Insights on the Global Market*. Disponible en: `https://radicalstorage.com/travel/airbnb-statistics/`

[8] Business of Apps. (s. f.). *Airbnb Statistics (2025)*. Disponible en: `https://www.businessofapps.com/data/airbnb-statistics`

[9] Uber Engineering. (s. f.). *Inside Uber's Big Data Platform*. Disponible en: `https://www.uber.com/en-CO/blog/uber-big-data-platform/`