

# Large-Scale and Multi-Structured Databases

## ***LargeB&B***

Grillo Alessio  
Sfar Omar  
Rotelli Luca

# Application Highlights

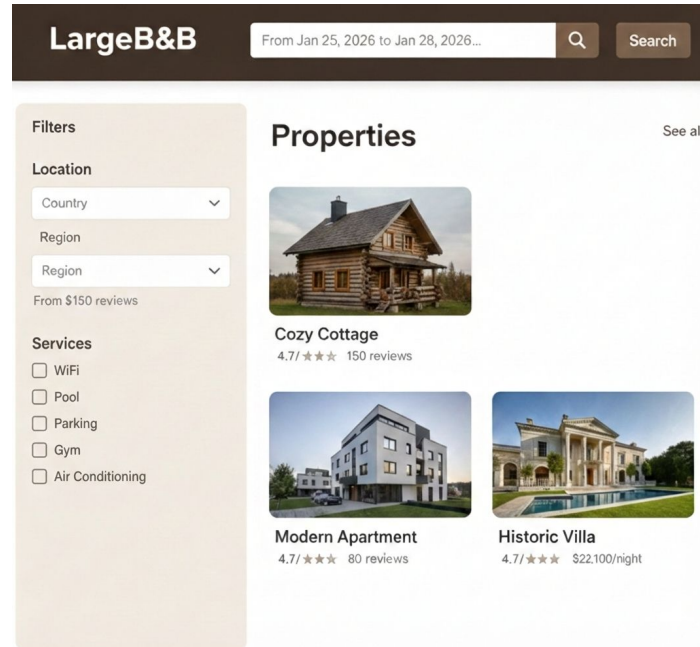
**LargeB&B** is a web application that acts as an intermediary platform for booking accommodations. It enables travelers to search, compare, and book hotels, bed & breakfasts, vacation rentals, apartments, and other types of lodging. At the same time, the platform provides property owners and managers with the opportunity to promote their accommodations to a global audience, monitor bookings, and access detailed analytics on the performance of their properties.

The **main features** of LargeB&B include:

- Advanced search and filtering of available rooms based on location and review ratings.
- The ability for users to write and read reviews to help other travelers make informed decisions.
- Real-time chat communication between customers and property managers, both before and after booking.
- Access to analytics and performance insights for property owners regarding the accommodations they have published.

# Actors (i)

Customer and unregistered user



Page where the user search for properties and filter them.

# Actors (ii)

Customer and unregistered user

**LargeB&B**

**Filters**

**Location**

Country

▼

**Region**

City

Tuscania

▼

From \$150 reviews

**Services**

**Services**

☒ Breakfast Included

☐ Pet-Friendly

☐ Pool Access

**Price Per Night**


Adults  \$19000

\$00


Children  \$500

**Rooms in Florence**


Sort by: Recommended




**Deluxe King Room**  
4.7/★★★★ (150 reviews)



**Standard Room**  
4.7/★★★★ (150 reviews)



**Standard Double**  
4.5/★★★★ (Private Balcony  
Ensuite Bathroom)



**Standard Double**  
4.5/★★★★ (80 reviews)  
Ensuite Bathroom

Page where the user search for rooms and filter them.

# Actors (iii)

Customer and unregistered user

**LargeB&B**  🔍 Becoms Host

Astro 4.98

**Sunny Loft with Private Terrace near Colosseum**

A bright refuge in the heart in the Eternal City

Immerse as pröpie in the Dolce Vita' in this splendid loft this recently desatly clmt, recoetted Roman chari a true true viate renorate panoromie. Punshed deyerrace with hrnished terrace with the cupnitanic plants. High-speed-WFi and and smart working attra station.

**Reviews**

Average rating: ★ 4.98 (124 reviews)

Giulia M. (October 2025) "Simply magical battir rbetter eve better evet's that that tiigittful and and bo trip."

Mark & Sarth location. vs atitile be nülle lante voit fristowe. Have hote was tieupper hedful and odiells bce vor impeccable."

Mark & Sarth : (September 2025)

Alesandro R. Quiet apartisntnt be ea lenralifort very bdnat rates trip"

Pricing per night:

Adults:

Children (2-15 years)

Estimated total

€290.00 + cleaning fees:

Estimated total:

€145.00

Estimated (e.g., 2 nights, 2 adults):

[ BOOK NOW ]

You won't charged at stage

Page where the user can see the room details.

# Actors (iv)

## Customer

The screenshot displays the 'Your Bookings' section of the LargeB&B website. The top navigation bar includes 'LargeB&B', 'Dashboard', 'Discover', 'Account', 'Astro', and a user profile icon. The main heading is 'Your Bookings'. Under the 'Upcoming' tab, there is a booking for 'Sunny Loft Colosseum' in Rome, Italy, for March 15 - March 19, 2026. The booking ID is 2026-3-15-A. It includes a 'Manage Booking' button, an 'Add to Calendar' button, and a grid icon. Below this, a summary for 'Rome, Italy' shows the dates and a total cost of 115.65. The cost breakdown is: € 290,38,21 for flights and airport fees, and € 90,20 for nights and cleaning. Under the 'Past Bookings' tab, there are two entries. The first is 'Historic Villa Amaflia Coast' for August 2025, with a cost of 2019. It has a 'View Details' button and a 'Remove' button with a red X icon. The second entry is for May 2023, with a 'Leave a Review' button, a plus icon, and a red X icon.

LargeB&B Dashboard Discover Account Astro

### Your Bookings

Upcoming

**Sunny Loft Colosseum** Restertitaly  
March 15 - march 19, 2026  
Booking ID: 2026-3-15-A  
[Manage Booking] Add to Calendar

**Rome, Italy** Bomated total  
March 15 - march 19, 2026  
total: 115.65  
€ € 290,38,21) \*Poal: flants: w pot ar oit n e90,20 + nghts: cleans.

Past Bookings

**Historic Villa Amaflia Coast** Cost chetiel:  
sang ineg spolvnt serar'as." 2019  
August 2025  
August 2025 View Details Remove X  
May 2023  
May 2025 [Leave a Review] + X

Page where the customer can see his reservations.

# Actors (v)

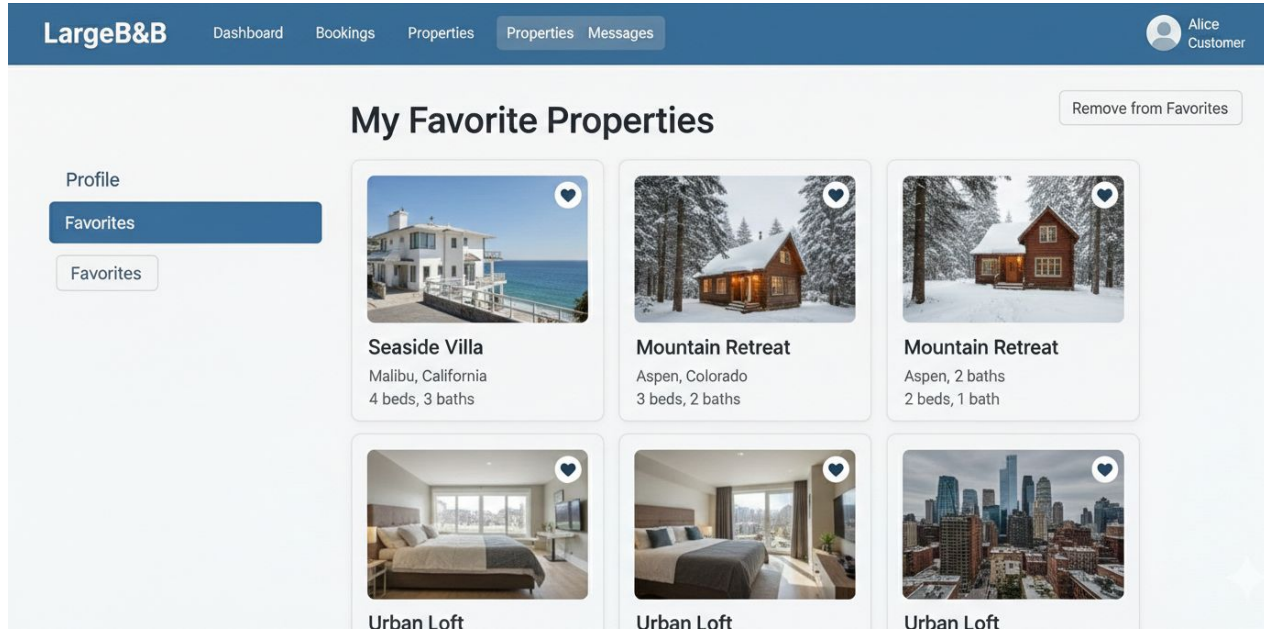
## Customer

The screenshot displays the 'Manage Payment Methods' interface for a user named Alice. The page has a sidebar with 'Profile' and 'Payment Methods' options. The main content area shows two existing payment methods: a VISA card (\*\*\*\* \* 4567, Expires 12/28) and a PayPal account (john.doe@example.com). A red box highlights the '+ Add New Payment Method' button. A modal form is open, containing the following fields: 'Card Number' (text input), 'Expiry Date' (dropdown menu), 'CVV' (text input), and 'Name on Card' (text input). At the bottom of the modal are 'Save Card' and 'Cancel' buttons.

Page where the customer can manage his payment method.

# Actors (vi)

## Customer

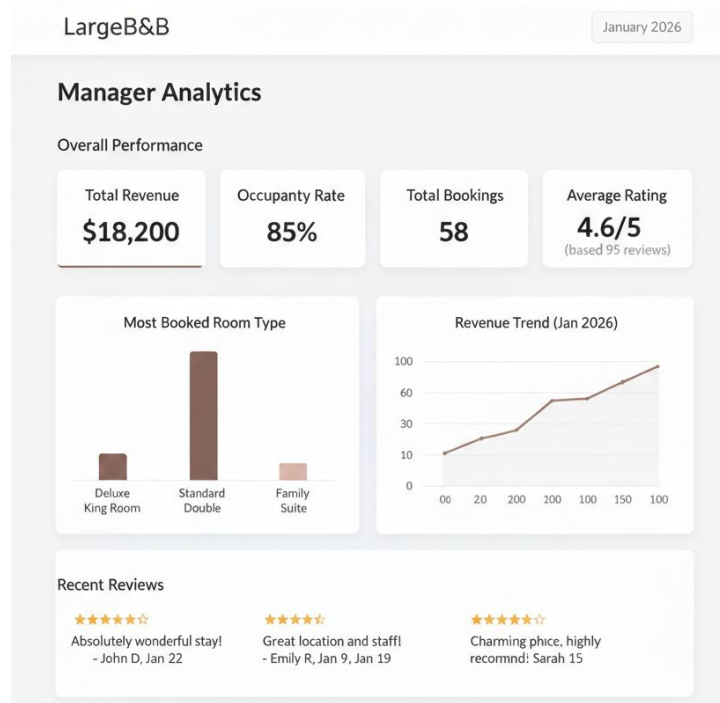


Page where the customer can see his favorite properties.



# Actors (vii)

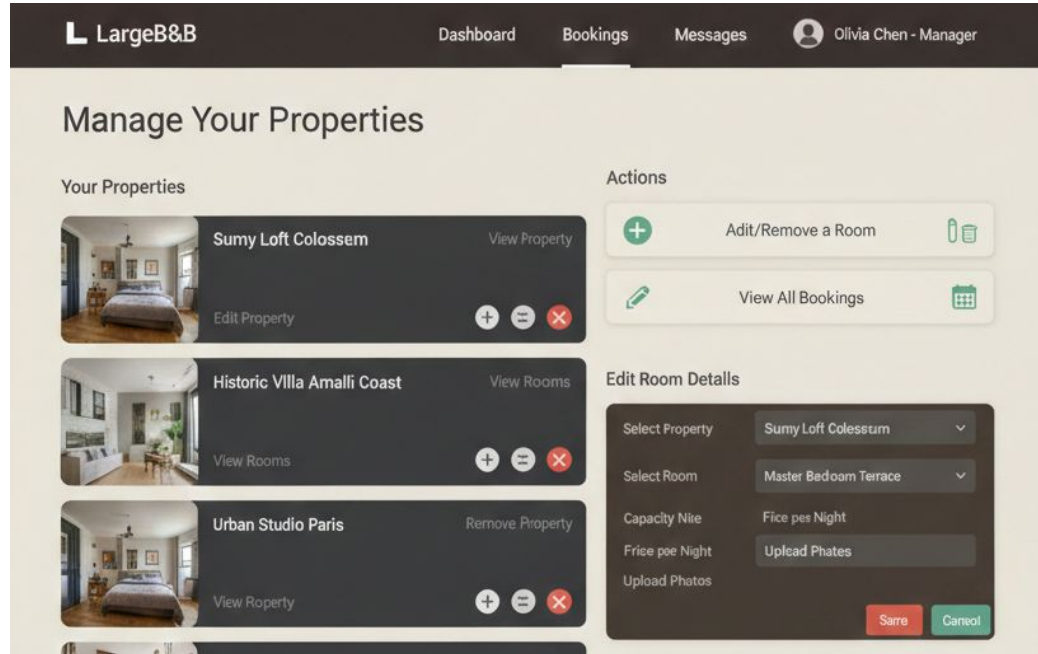
## Manager



Page where the manager can see his analytics.

# Actors (viii)

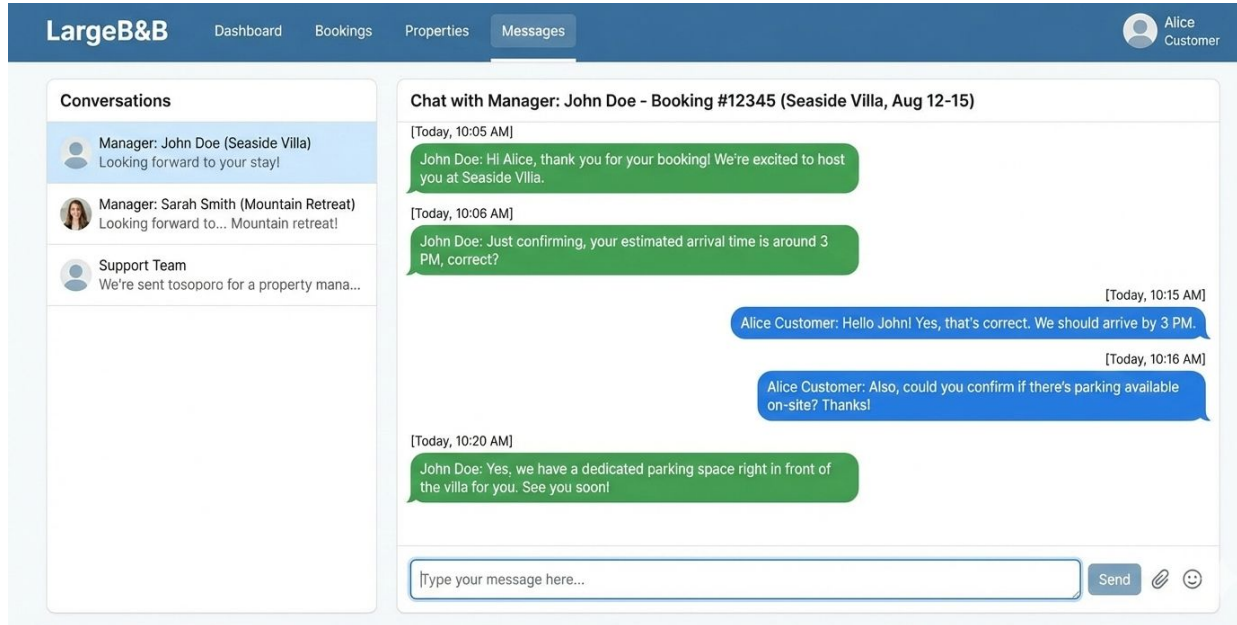
## Manager



Page where the manager can manage his properties.

# Actors (ix)

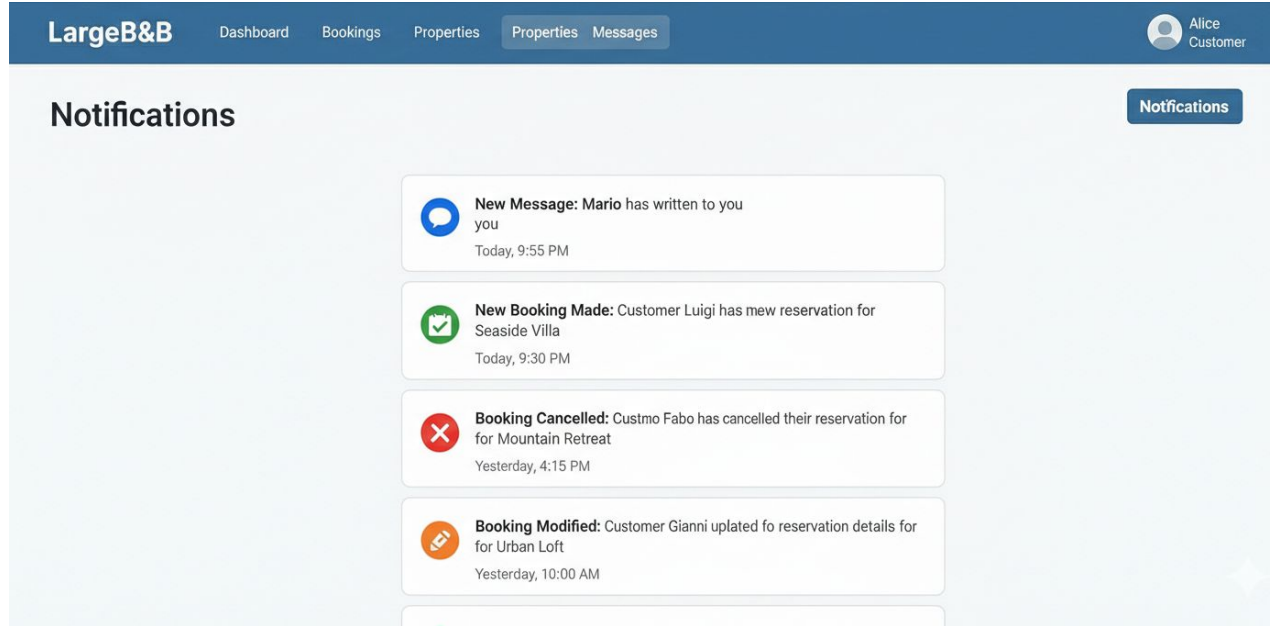
## Manager and Customer



Page where the manager or customer can send/receive messages.

# Actors (x)

## Manager and Customer



Page where the manager or customer can see his notifications (booking notification are only for Manager).

# Dataset Description

**Sources:** *Inside AirBnB, Faker*

**Description:** *Dataset concerning properties and rooms information, reviews, notifications, messages between customer and manager, and reservations information.*

**Volume:** *170MB*

**Variety:** Frequently changing data: real-time room availability, new bookings, and daily reviews.

# Dataset Description

## **Velocity/Variability:**

*The system handles detailed **room and property descriptions**, along with essential booking data such as **guest counts** (adults and children) and **check-in/check-out dates**. It features a comprehensive review system that includes **multi-criteria ratings** for stays and allows for **manager responses**. Additionally, a real-time notification system alerts users to **incoming messages** and provides managers with exclusive updates regarding the **reservation status** of their properties.*

## **Pre-processing:**

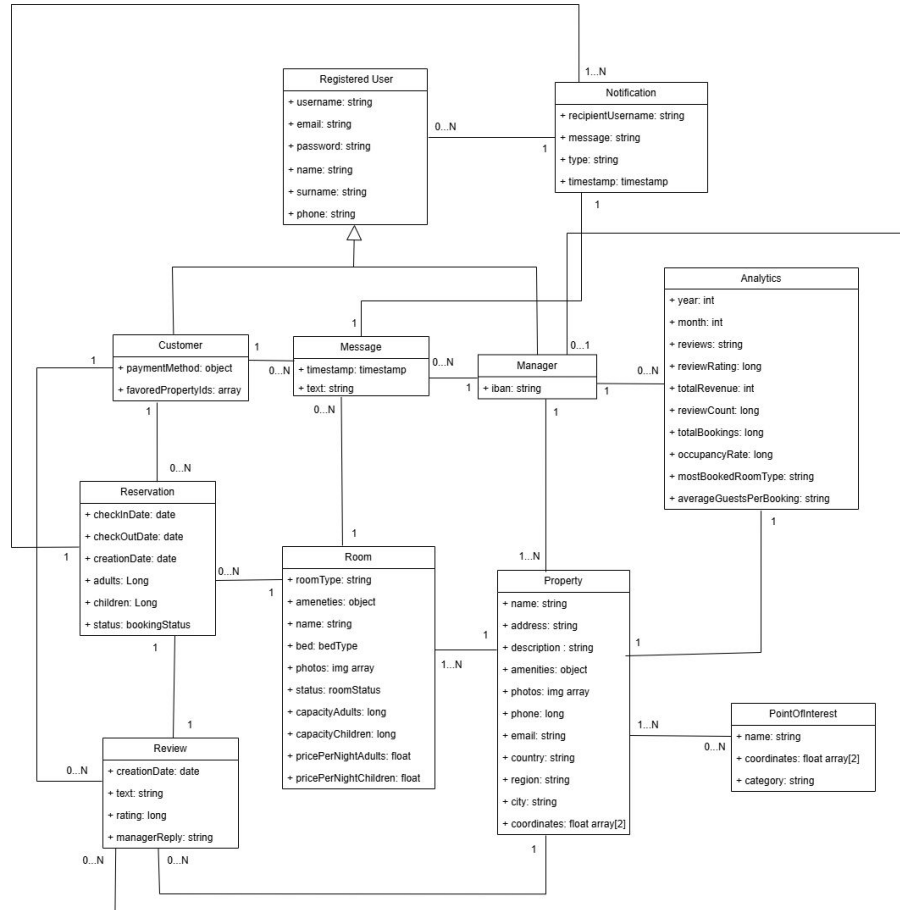
**Data Cleaning:** *Used Python (Pandas) to clean InsideAirbnb CSVs (handling missing values, removing inactive listings).*

**Data Fusion:** *Merging real property data with synthetic Manager profiles.*

**Transformation:** *Converting flat CSV records into hierarchical MongoDB Documents (Embedding Rooms/Reviews).*

**Geo-Coding:** *Transforming lat/lon columns into GeoJSON points objects for spatial indexing.*

# UML Design Class Diagram



# Application non-functional requirements

- *Passwords of registered users must be securely encrypted.*
- *The system must provide notifications via application.*
- *The results produced by the system must be consistent.*
- *The system must ensure high data availability.*
- *The system must manage data storage using Document DB, Graph DB, and Key-Value DB.*
- *The system should be able to scale to accommodate an increase in user traffic.*



# Application non-functional requirements

- *The DBMS technologies used must be Neo4j, MongoDB, and Redis.*
- *The business logic and data access layers must be implemented using the Spring framework in Java.*
- *The codebase must be easily maintainable, readable, and updatable.*
- *The system must use a document database to store information permanently.*
- *The system must use a key-value database to temporarily store information, manage cache, locks, and user sessions.*

# Document DB Design

## Properties

```
{
  "_id": "bdd640fb-0667-4ad1-9c80-317fa3b1799d"
  name: "Elif's room in cozy, clean flat."
  address: "VIII Appia Antica"
  description: "10 min by bus you can get to Piazza Venezia or Colosseum. All shops, g..."
  amenities: Array (42)
  photos: Array (5)
  email: "williamdavis@callahan.com"
  country: "Italy"
  region: "Lazio"
  city: "Rome"
  ratingStats: Object
  latestReviews: Array (10)
  managerId: "23b8c1e9-3924-46de-beb1-3b9046685257"
  location: Object
  rooms: Array (3)
  pois: Array (6)
}
```

## Reservations

```
{
  "_id": "1825bc54-30be-445f-a835-14f2ceb81f9d"
  adults: 2
  children: 1
  status: "confirmed"
  userId: "14296c07-f26b-4776-913e-4de2e0c53cb8"
  roomId: "ce88cb2d-d4e8-4839-bc3e-058be0f3eab0"
  dates: Object
  createdAt: 2025-11-19T00:00:00.000+00:00
}
```

## Rooms (embedded)

```
{
  roomType: "single"
  amenities: Array (42)
  name: "Room 1"
  beds: 2
  photos: Array (3)
  status: "available"
  capacityAdults: 3
  capacityChildren: 2
  pricePerNightAdults: 92.25
  pricePerNightChildren: 19.42
  roomId: "e2817efd-ae84-4217-9d53-434bb88139b9"
}
```

## Pois (embedded)

```
{
  id: "113634a5-2050-4c62-88e8-95d7702c1821"
  name: "Colosseum"
  coordinates: Array (2)
  type: "historical"
}
```

## Messages

```
{
  _id: "ee87905e-4ca4-45ea-8dfa-6a56d12dbc9a"
  timestamp: 2025-11-24T00:56:00.000+00:00
  content: "Voice care break blood network evening painting."
  senderId: "da4bd9ca-eb5c-4467-80ba-cd647a0ecfea"
  recipientId: "23b8c1e9-3924-46de-beb1-3b9046685257"
  isRead: true
}
```

# Document DB Design

## Reviews

```
_id: "0b19f88e-9d77-445e-b206-c26938b77c07"
propertyId: "bdd640fb-0667-4ad1-9c80-317fa3b1799d"
userId: "6712303a-0f84-4fef-9931-e9eea56c0941"
reservationId: "0ef8c2d6-f7fd-4646-b7bb-3eec4bf50b52"
creationDate: 2014-12-26T00:00:00.000+00:00
text: "This spot was in a great and nice area,walking a distance about 700 m:..."
rating: 5
cleanliness: 3
communication: 5
location: 4
value: 3
managerReply: "Feeling poor all your suggest international blue."
```

## Notifications

```
_id: "6c6fa611-5ab3-4edf-ae59-5ed3a8b317fa"
recipientId: "23b8c1e9-3924-46de-beb1-3b9046685257"
title: "New Booking"
body: "Customer Kyle has made a new reservation."
type: "RESERVATION_CREATED"
referenceId: "1825bc54-30be-445f-a835-14f2ceb81f9d"
read: false
timestamp: 2025-11-19T00:00:00.000+00:00
```

## Users (Customer)

```
_id: "14296c07-f26b-4776-913e-4de2e0c53cb8"
username: "dennislisa"
email: "davidalvarez@example.net"
password: "#4RQ2tzaKQ"
name: "Kyle"
surname: "Brown"
birthdate: 1966-05-05T00:00:00.000+00:00
phoneNumber: "001-330-510-3105"
▶ paymentMethod: Object
▶ favoredPropertyIds: Array (empty)
role: "CUSTOMER"
```

## Users (Manager)

```
_id: "23b8c1e9-3924-46de-beb1-3b9046685257"
username: "rhodespatricia"
email: "garzaanthony@example.org"
password: "@2$3Qtllld"
name: "Susan"
surname: "Rogers"
phoneNumber: "361-855-9407"
iban: "GB38CRJU95931034131647"
vatNumber: "KD52553419283"
▶ billingAddress: Object
role: "MANAGER"
```

# Document DB Design

## Aggregation Queries Benefiting from Design

```
// Advanced Search - amenities embedded enables $all operator
db.properties.aggregate([
  { $match: { amenities: { $all: ["Wifi", "Pool"] }, city: "Rome" } },
  { $sort: { "ratingStats.value": -1 } }
])

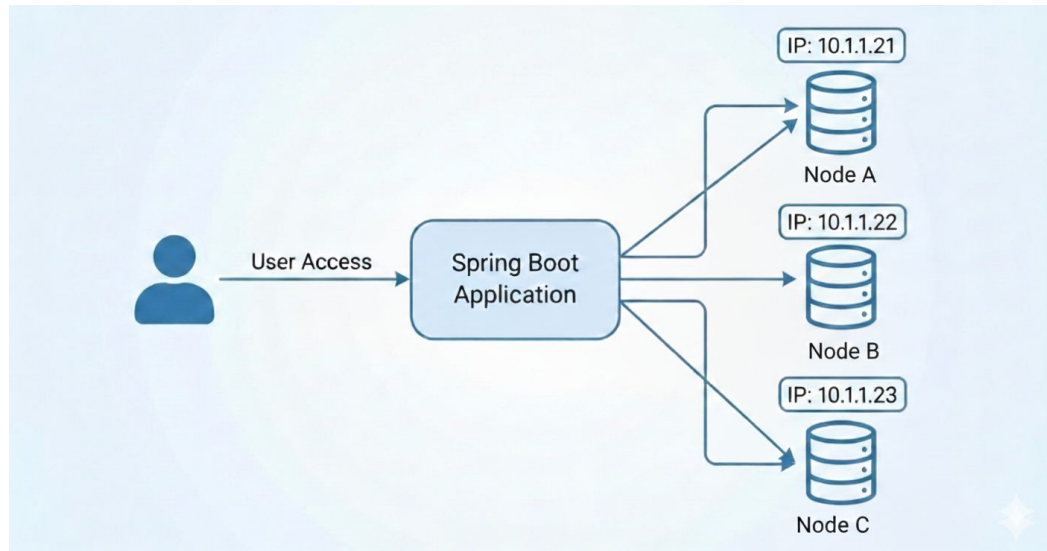
// Geospatial Query - coordinates embedded in Property
db.properties.aggregate([
  { $geoNear: { near: { type: "Point", coordinates: [12.4964, 41.9028] },
    distanceField: "distance", maxDistance: 5000, spherical: true } }
])

// Top Rated Properties - ratingStats pre-computed
db.properties.aggregate([
  { $match: { "ratingStats.count": { $gte: 10 } } },
  { $addFields: { avgRating: { $divide: ["$ratingStats.sum", "$ratingStats.count"] } } },
  { $sort: { avgRating: -1 } }, { $limit: 10 }
])
```

We adopted an embedded model for data always queried together (Property+Rooms), and a referenced model for independent but correlated entities (Reservation  $\leftarrow$  Room). This enables fast queries such as amenity-based search, geolocation, and rating-based sorting without expensive JOINS. Each application view queries data exactly as it is structured, maximizing read performance.

# Cluster Representation

The system deployment involves a 3-node cluster configured to maximize resource efficiency and availability.



# MongoDB Replica Set Configuration

We deployed a **MongoDB Replica Set** across all three nodes with a priority-based election strategy: Node A acts as the **Primary** (Priority 5), while Nodes B (Priority 2) and C (Priority 1) serve as Secondaries.

# MongoDB Replica Set Configuration

## Read Operations (Read Preference: nearest)

Given that the read-to-write ratio in LargeB&B is estimated to be approximately 10:1 (browsing vs. booking), minimizing read latency is crucial. We utilize the nearest read preference. The driver routes the read request to the replica set member with the lowest network latency, regardless of whether it is Primary or Secondary.

This choice provides two main benefits:

- **Load Balancing:** It distributes the heavy read traffic across all three nodes, preventing the Primary from being overwhelmed.
- **Geo-latency Reduction:** If the infrastructure is geo-distributed in the future, users will automatically read from the data center closest to them.



# MongoDB Replica Set Configuration

## **Write Operations (Write Concern: majority):**

Although we prioritize availability for reads, write operations—specifically User Registration, Property Creation, and critical Reservation flows—require strict durability. We have configured the Write Concern to majority.

- The system acknowledges a write only after it has been committed to the Primary and propagated to at least one Secondary (2 out of 3 nodes).
- This ensures that even if the Primary node crashes immediately after a write (e.g., after a confirmed payment), the data is safe on a Secondary node that will become the new Primary, preventing data rollback and financial inconsistencies.



# MongoDB Indexes

*Left image query example with no index & right image query example with index*

## **Properties Collection:**

*{"rooms.capacityAdults": 1, "rooms.capacityChildren": 1}: It allows the database to efficiently filter properties that contain at least one room capable of accommodating the requested number of adults and children.*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1920,  
  executionTimeMillis: 119,  
  totalKeysExamined: 0,  
  totalDocsExamined: 1950,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1920,  
  executionTimeMillis: 85,  
  totalKeysExamined: 5337,  
  totalDocsExamined: 1933,
```

# MongoDB Indexes

*"ratingStats.value": Optimizes the "Sort by Highest Rated" feature, ensuring that high-rated properties are retrieved efficiently without performing expensive in-memory sorting*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1950,  
  executionTimeMillis: 178,  
  totalKeysExamined: 0,  
  totalDocsExamined: 1950,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1950,  
  executionTimeMillis: 45,  
  totalKeysExamined: 1950,  
  totalDocsExamined: 1950,
```

# MongoDB Indexes

*“managerId”: Ensures fast retrieval of the specific subset of properties owned by a logged-in manager for the dashboard view*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 5,  
  totalKeysExamined: 0,  
  totalDocsExamined: 1950,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 0,  
  totalKeysExamined: 1,  
  totalDocsExamined: 1,
```

# MongoDB Indexes

## **Text-Index:**

*“name (property) (weight: 3), name (room) (weight: 2), description (weight: 1), city (weight: 1)”:  
Enables full-text search capabilities with relevance scoring. Matches in the property name are prioritized over matches in the description, improving search relevance for users*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 354,  
  executionTimeMillis: 148,  
  totalKeysExamined: 406,  
  totalDocsExamined: 354,
```

# MongoDB Indexes

## **2dsphere Index:**

*“coordinates”*: Required to support geospatial queries such as *\$near* and *\$geoWithin*, satisfying the requirement to view properties on a map or find POIs nearby.

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 135,  
  executionTimeMillis: 36,  
  totalKeysExamined: 165,  
  totalDocsExamined: 139,
```

# MongoDB Indexes

**Rooms (Embedded in Property collection):**

*“pricePerNightAdults”: Optimizes complex budget queries where users filter by adult price*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 5,  
  totalKeysExamined: 0,  
  totalDocsExamined: 1950,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 0,  
  totalKeysExamined: 1,  
  totalDocsExamined: 1,
```

# MongoDB Indexes

## **Reservations Collection:**

*{ "roomId": 1, "dates.checkIn": 1, "dates.checkOut": 1 } : It allows the system to efficiently check if a specific room is occupied during a given date range before confirming a new booking*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 34,  
  totalKeysExamined: 0,  
  totalDocsExamined: 29644,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 0,  
  totalKeysExamined: 1,  
  totalDocsExamined: 1,
```

# MongoDB Indexes

*“userId”: Optimizes the retrieval of booking history for the “My Bookings” page*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 859,  
  totalKeysExamined: 0,  
  totalDocsExamined: 29644,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 77,  
  totalKeysExamined: 1,  
  totalDocsExamined: 1,
```



# MongoDB Indexes

*“userId”: Allows the system to quickly retrieve all reservations associated with a specific room*

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 859,  
  totalKeysExamined: 0,  
  totalDocsExamined: 29644,
```

```
executionStats: {  
  executionSuccess: true,  
  nReturned: 1,  
  executionTimeMillis: 77,  
  totalKeysExamined: 1,  
  totalDocsExamined: 1,
```

# Discussion on MongoDB Data Sharding

*Since the **id** field is mandatory in every document and is the primary filter for the majority of queries, it was selected as the shard key. This choice enables direct query routing, avoiding the overhead of scatter-gather operations. Consequently, we adopted Hashed Sharding as the partitioning algorithm to ensure uniform distribution.*

# Key-value DB Design

## Redis Data Model & Key Naming

### Key Naming Strategy:

1. JWT Blacklist: `"blacklist:{token}"` → Value: `"true"` (String, TTL auto-expire)
2. Pending Reservations: `"reservation:temp:{uuid}"` → Value: Reservation JSON (15min TTL)
3. Trending Properties: `"trending_properties"` → Sorted Set (propertyId, score)
4. Recently Viewed: `"recent:{userId}"` → List of propertyIds (max 10)

### Design Justification:

- **Namespace prefixes** (blacklist:, reservation:temp:, recent:) prevent key collisions
- **TTL expiration** for temporary data (JWT tokens, pending reservations) ensures automatic cleanup
- **Sorted Set** for trending enables  $O(\log N)$  scoring and range queries
- **List** for recently viewed maintains insertion order with  $O(1)$  head insertion

# Key-value DB Design

## Implemented Operations

```
// 1. JWT Token Blacklisting (Secure Logout)
redisTemplate.opsForValue().set(
    "blacklist:" + token, "true", ttl, TimeUnit.MILLISECONDS
);
Boolean isBlacklisted = redisTemplate.hasKey("blacklist:" + token);

// 2. Pending Reservation (Temporary Cart)
redisTemplate.opsForValue().set(
    "reservation:temp:" + uuid, reservation, 15, TimeUnit.MINUTES
);

// 3. Trending Properties (Real-time Analytics)
redisTemplate.opsForZSet().incrementScore("trending_properties", propertyId, 1);
Set<Object> topIds = redisTemplate.opsForZSet().reverseRange("trending_properties", 0, 9);

// 4. Recently Viewed History
redisTemplate.opsForList().leftPush("history:" + userId, propertyId);
redisTemplate.opsForList().trim("history:" + userId, 0, 9); // Keep only last 10
```

# Key-value DB Design

## System Benefits:

- **Security:** Stateless JWT invalidation without database queries
- **Performance:** Sub-millisecond access for trending properties (vs MongoDB aggregation)
- **UX:** Pending reservations survive page refresh during payment flow
- **Personalization:** Recently viewed history without polluting main DB

## Views Supported:

- **ShowProperties** - Displays trending properties banner (reads `trending_properties` ZSet)
- **Checkout Flow** - Retrieves pending reservation during payment (reservation:temp:{id})
- **User Dashboard** - Shows recently viewed properties (recent:{userId} List)

# Redis Replica Set Configuration

*For Redis, Node A acts as the **Master** handling both write and read operations, supported by two Read-only Replicas on Nodes B and C. High Availability is guaranteed by **3 Redis Sentinels**, which continuously monitor the cluster's health and automatically promote a Replica to Master in the event of a failure. Data durability and disaster recovery are secured through periodic **RDB snapshots** persisted to disk.*

*To ensure system stability under heavy load, we implemented a strict memory management strategy. We set a maxmemory **limit of 512MB** combined with a **Least Frequently Used (LFU)** eviction policy. When this limit is reached, Redis evicts the least accessed keys first, protecting the system from memory saturation and crashes during traffic spikes.*

# Discussion on Redis Data Sharding

*We implemented Sharding in Redis for:*

- **Horizontal Scalability (RAM):** *It allows storing datasets larger than the memory limit of a single server by distributing data across multiple nodes.*
- **Increased Write Throughput:** *Since Redis is single-threaded, sharding splits write operations across multiple CPUs, significantly increasing total cluster performance.*

*The **database key** itself serves as the **sharding key**, and the partitioning strategy relies on a **hashing algorithm**.*

# Graph DB Design

(:Property) -[:HAS]->(:Amenity)



(:User) -[:BOOKED]->(:Property)





# Graph DB Design

## Collaborative Filtering (User-Based Recommendations)

```
// Find properties booked by users with similar booking patterns
MATCH (p:Property {propertyId: $propId})<-[:BOOKED]-(u:User)
  -[:BOOKED]->(other:Property)
RETURN other.propertyId AS recommendedId,
       count(*) AS strength
ORDER BY strength DESC
LIMIT 5
```

## Content-Based Filtering (Feature Similarity)

```
// Find properties sharing common amenities
MATCH (p:Property {propertyId: $propId})-[:HAS]->(a:Amenity)
  <-[:HAS]-(other:Property)
WHERE p.propertyId <> other.propertyId
RETURN other.propertyId AS recommendedId,
       count(a) AS matchStrength
ORDER BY matchStrength DESC
LIMIT 10
```

# Graph DB Design

## Performance:

- Graph traversal in  $O(\text{depth} \times \text{branching\_factor})$ , independent of total dataset size
- 2-hop collaborative filtering: ~10ms vs ~500ms MongoDB aggregation

## Query Expressiveness:

- "Users who booked X also booked Y" translates naturally to graph pattern matching
- Amenity-based similarity requires no complex array operations

## Decoupled Architecture:

- Neo4j stores projection → can be rebuilt from MongoDB if corrupted
- Read-only for recommendations → no write conflicts with transactional data

## Views Supported:

- **Property Details Page** → Shows "Similar Properties" (content-based query)
- **Recommendations Section** → Shows "Users Also Booked" (collaborative filtering)

# Neo4j Indexes

*Left image query example with no index & right image query example with index*

## **Property Node:**

*“PropertyId”: Both recommendation algorithms begin with a specific property currently being viewed (the “Anchor Node”). This constraint ensures  $O(1)$  lookup time to locate this starting point in the graph immediately*

**Cypher version:** 5

**Planner:** COST

**Runtime:** PIPELINED

**Total database accesses:** 3.907

**Total allocated memory:** 312 bytes

**Total time:** 467 ms

**Cypher version:** 5

**Planner:** COST

**Runtime:** PIPELINED

**Total database accesses:** 4

**Total allocated memory:** 312 bytes

**Total time:** 363 ms

# Neo4j Indexes

## **User Node:**

*“UserId”: Essential for the Collaborative Filtering algorithm*

**Cypher version:** 5

**Planner:** COST

**Runtime:** PIPELINED

**Total database accesses:** 63.195

**Total allocated memory:** 312 bytes

**Total time:** 283 ms

**Cypher version:** 5

**Planner:** COST

**Runtime:** PIPELINED

**Total database accesses:** 4

**Total allocated memory:** 312 bytes

**Total time:** 276 ms

# Neo4j Indexes

## **Amenity Node:**

*“AmenityName”: Ensures that shared features (e.g., “WiFi”, “Pool”) are represented as single, unique nodes in the graph. This allows the traversal algorithm to “hop” from the current property to related properties via these shared amenity nodes efficiently*

**Cypher version:** 5

**Planner:** COST

**Runtime:** PIPELINED

**Total database accesses:** 6

**Total allocated memory:** 312 bytes

**Total time:** 286 ms

**Cypher version:** 5

**Planner:** COST

**Runtime:** PIPELINED

**Total database accesses:** 3

**Total allocated memory:** 312 bytes

**Total time:** 126 ms

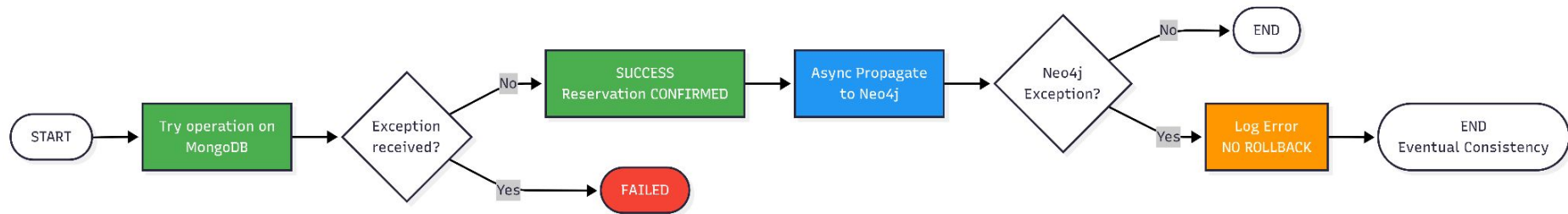
# Handling Intra-DB Consistency

**Hypothesis:** Since MongoDB has 3 replicas with Write Concern "majority", we can assume that a fail on connection or data persistence is almost impossible.

This consistency is managed using **asynchronous propagation** with fire-and-forget pattern:

- **MongoDB is the Source of Truth:** All critical write operations (reservations, payments) are persisted to MongoDB FIRST with strong consistency guarantees.
- **Neo4j is a Projection:** After MongoDB confirms the transaction, the relationship is propagated to Neo4j asynchronously using CompletableFuture.
- **Eventual Consistency Model:** If Neo4j propagation fails, the system logs the error but does NOT rollback the MongoDB transaction. The reservation remains valid.
- **Acceptable Inconsistency Window:** There may be a brief delay (milliseconds) before the booking appears in the recommendation graph, but this doesn't impact transactional integrity or user experience.

# Handling Intra-DB Consistency



```
CompletableFuture.runAsync(() -> {  
    try {  
        reservationGraphRepository.createReservation(  
            finalReservationId,  
            finalUserId,  
            finalPropertyId,  
            finalReservation.getDates().getCheckIn(),  
            finalReservation.getDates().getCheckOut(),  
            finalAmount  
        );  
        logger.info("Graph: Reservation {} synced to Neo4j successfully.", finalReservationId);  
    } catch (Exception e) {  
        logger.error("Graph Error: Failed to sync reservation {} to Neo4j: {}",  
            finalReservationId, e.getMessage());  
    }  
});
```

# Swagger UI REST APIs documentation

Swagger doc url: <http://localhost:8080/swagger-ui/index.html>

## property-controller

GET	/api/properties/{propertyId}
GET	/api/properties/{propertyId}/recommendations/similar
GET	/api/properties/{propertyId}/recommendations/collaborative
GET	/api/properties/trending
GET	/api/properties/top-rated
GET	/api/properties/search
GET	/api/properties/map

## reservation-controller

PUT	/api/reservations/{reservationId}
DELETE	/api/reservations/{reservationId}
POST	/api/reservations/initiate
POST	/api/reservations/confirm-payment
GET	/api/reservations/my-reservations
DELETE	/api/reservations/temp/{tempReservationId}



# Live Demo with Postman

