# Private Dining Reservation System

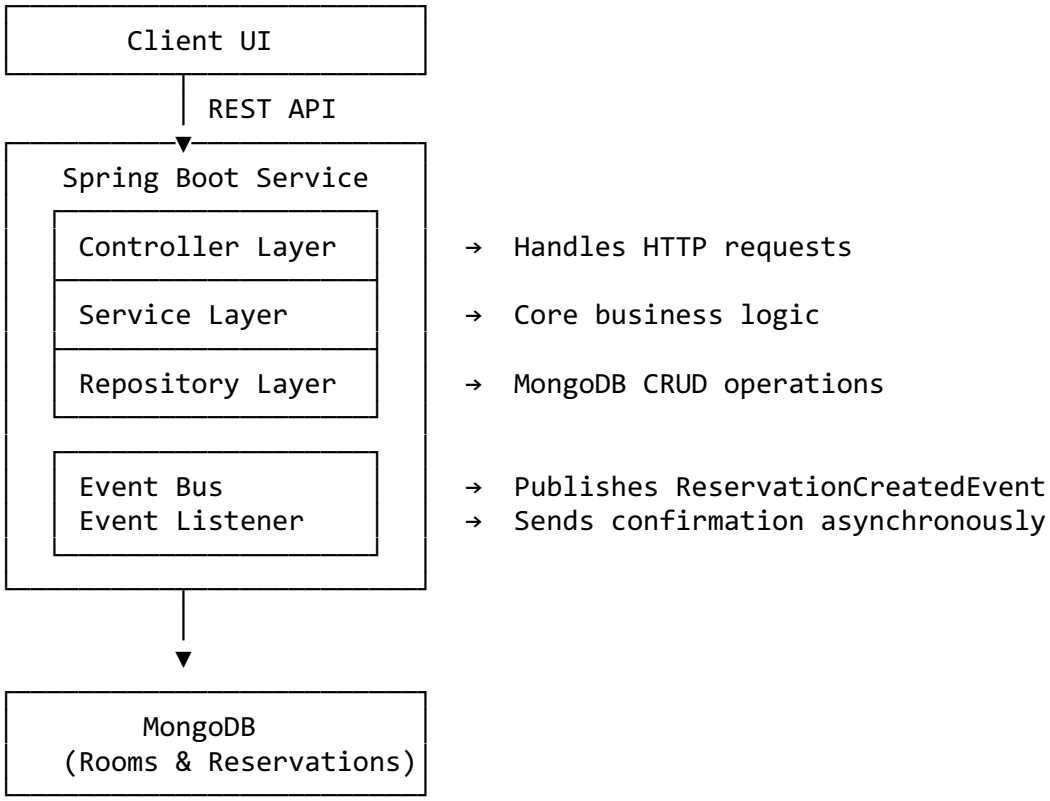
## 1. System Overview

The **Private Dining Reservation System** is a scalable, event-driven backend built using **Spring Boot** and **MongoDB**, designed to handle high volumes of restaurant reservations efficiently while preventing double bookings and ensuring data consistency.

### Key Components:

- **Spring Boot (Java 17)** – Core backend framework providing RESTful APIs.
- **MongoDB** – NoSQL database optimized for flexible, schema-less data storage and horizontal scaling.
- **Spring Data MongoDB** – Simplifies CRUD operations and index management.
- **ApplicationEventPublisher** – Enables asynchronous event-driven communication.
- **Mongo Partial Unique Index** – Enforces uniqueness for active (CONFIRMED) reservations only.

### Architecture Overview




### Data Flow Summary

1. A diner requests a reservation for a specific room, date, and time.
2. The system validates capacity and checks for availability.
3. If available, the reservation is saved and a confirmation event is published.

4. Listeners handle asynchronous notifications or analytics.

This design decouples reservation persistence from notification handling, allowing independent scaling of business logic and background tasks.

## 2. Database Schema and Justification

### Choice of Database: MongoDB

MongoDB was chosen for its flexible schema, native support for JSON-like documents, and ease of handling dynamic reservation structures. In contrast to PostgreSQL, MongoDB provides:

- **Simpler horizontal scaling** for high read/write traffic.
- **Natural document representation** of restaurant–room–reservation relationships.
- **Efficient querying** for time-based searches and partial unique indexes.

### Collections

***restaurants***

| Field | Type | Description |
|---|---|---|
| _id | String | Unique restaurant ID |
| name | String | Restaurant name |
| address | String | Restaurant address |
| contactEmail | String | Email for customer communication |
| roomIds | Array[String] | Linked room identifiers |

***rooms***

| Field | Type | Description |
|---|---|---|
| _id | String | Room ID |
| restaurantId | String | Parent restaurant reference |
| name | String | Room name |
| type | String | Room type (e.g., rooftop, hall) |
| minCapacity | Integer | Minimum guests allowed |
| maxCapacity | Integer | Maximum guests allowed |
| minSpend | Decimal | Minimum spend amount |
| currency | String | Currency code (AUD) |

***reservations***

| Field | Type | Description |
|---|---|---|
| _id | String | Reservation ID |
| roomId | String | Associated room |
| restaurantId | String | Associated restaurant |
| dinerName | String | Customer name |

| Field | Type | Description |
|---|---|---|
| email | String | Customer email |
| partySize | Integer | Group size |
| reservationDate | LocalDate | Date of reservation |
| startTime | LocalTime | Start time |
| endTime | LocalTime | End time |
| status | String | CONFIRMED or CANCELLED |

### Partial Unique Index (MongoDB)

```
{
  key: { roomId: 1, reservationDate: 1, startTime: 1, endTime: 1 },
  name: "unique_confirmed_room_slot",
  unique: true,
  partialFilterExpression: { status: "CONFIRMED" }
}
```

✅ Ensures that **only one CONFIRMED reservation** exists for a specific time range, while allowing rebooking after cancellation.

## 3. API Design (OpenAPI-style)

### Base URL: /api

***POST /reservations** – Create Reservation*

**Request Body:**

```
{
  "roomId": "room101",
  "restaurantId": "rest1",
  "dinerName": "Gary Dudeja",
  "email": "gary@example.com",
  "partySize": 6,
  "reservationDate": "2025-11-05",
  "startTime": "18:00",
  "endTime": "20:00"
}
```

**Response:** 201 CREATED

```
{
  "id": "resv12345",
   "roomId": "room101",
  "restaurantId": "rest1",
  "dinerName": "Gary Dudeja",
  "email": "gary@example.com",
  "partySize": 6,
  "reservationDate": "2025-11-05",
  "startTime": "18:00",
```

```
 "endTime": "20:00"
 "status": "CONFIRMED"
}
```


**Errors:** - 400 Bad Request → Invalid time or party size. - 409 Conflict → Time slot already booked.

***GET /reservations?email={email}*** *– Retrieve reservations by diner email*

Returns all reservations for the provided email.

***DELETE /reservations/{id}*** *– Cancel a reservation*

Marks reservation as CANCELLED (frees up the slot).

***GET /restaurants*** *– List all restaurants*

***GET /restaurants/{id}/rooms*** *– Get rooms for a specific restaurant*

---

## 4. Event-Driven Architecture

### **Event:** ReservationCreatedEvent

- **Publisher:** ReservationService
- **Consumer:** EmailNotificationListener
- **Mechanism:** @EventListener and @Async

**Workflow:**

1. Reservation is created and saved.

2. A ReservationCreatedEvent is published.

3. EmailNotificationListener handles the event asynchronously, simulating email notification.

**Example Log Output:**

Email sent to gary@example.com | Reservation confirmed for 2025-11-05 (18:00–20:00)

This asynchronous decoupling ensures that notification delays never block the reservation API.

Future Upgrade: Replace event bus with Kafka, RabbitMQ, or AWS SNS/SQS for distributed messaging.

## 5. Scalability Plan

- **Horizontal Scaling**

- The application is stateless → deploy multiple Spring Boot instances behind an AWS ALB or Kubernetes service.
- MongoDB cluster configured with replica sets ensures data redundancy and read scalability.

- ### Load Balancing

- **AWS ALB / NGINX / Kubernetes Ingress** evenly distributes requests.
- Health checks ensure traffic routing to healthy instances only.

- ### Caching Layer

- Introduce **Redis** for caching room availability and frequent queries.
- Reduces database reads during peak load.

- ### Event Queue (Future Enhancement)

- Use Kafka or RabbitMQ for email and audit events.
- Supports retry mechanisms and asynchronous scaling.

- ### Concurrency Safety

- Partial unique index guarantees no double booking.
- Application-level overlap validation ensures fine-grained control.

## 6. Summary

The Private Dining Reservation System balances flexibility and reliability through: - Spring Boot + MongoDB for high scalability and ease of development. - Partial unique indexes for data integrity. - Event-driven architecture for asynchronous task handling. - Stateless design allowing easy scaling under load.

This architecture is production-ready for a high-traffic restaurant reservation platform supporting multi-tenant scalability, fault tolerance, and future integration with event brokers like Kafka.