### 7.2 Data Management Strategy

#### 7.2.1 Overview

The Movie Ticketing System relies heavily on data persistence, consistency, and security to support its key functions such as user authentication, movie listings, ticket reservations, and payments. Our data management strategy focuses on ensuring data integrity, efficient access, and scalability while maintaining user privacy.

#### 7.2.2 Database Choice and Rationale

We chose to implement a **relational SQL database (MySQL)** for this system. SQL was selected due to the following reasons: - **Data Consistency:** Transactions (e.g., booking and payment) require strict ACID compliance to prevent double-booking or inconsistent payment states. - **Complex Relationships:** Entities such as users, movies, and bookings are interrelated and benefit from relational modeling. - **Ease of Maintenance:** MySQL provides mature tooling for schema evolution, backup, and recovery.

A NoSQL alternative (such as MongoDB) was considered but dismissed because it would introduce complexity in managing relationships between documents, such as ensuring referential integrity across bookings and payments.

#### 7.2.3 Database Structure

The database is organized into multiple tables corresponding to the main entities:

| Table | Description |
| --- | --- |
| **Users** | Stores user information such as name, email, password (hashed), and role (admin/customer). |
| **Movies** | Contains movie metadata including title, description, duration, rating, and available showtimes. |
| **Theaters** | Stores theater information such as name, location, and seating layout. |
| **Bookings** | Represents a user’s ticket purchase, referencing both user and movie tables, along with seat and showtime details. |
| **Payments** | Tracks payment details such as transaction ID, method, amount, and booking reference. |

This structure supports efficient joins and ensures normalized data storage, minimizing redundancy while maintaining flexibility for future extensions.

#### 7.2.4 Data Flow and Access

The application follows a layered architecture where the **data access layer** communicates with the database using an ORM (Object-Relational Mapping) framework such as Hibernate. This ensures clean separation between business logic and data persistence.

* **Read Operations:** Queries for movies, available seats, or user bookings are optimized using indexing and caching mechanisms.
* **Write Operations:** Insertions and updates (e.g., booking confirmations) use transactions to ensure consistency.

#### 7.2.5 Data Security and Privacy

Data security is critical for protecting user and payment information. The following measures are implemented: - **Encryption:** Sensitive data such as passwords and payment tokens are encrypted both in transit (TLS/SSL) and at rest. - **Access Control:** Role-based access ensures that administrators and users only access authorized data. - **Backup and Recovery:** Automated daily backups are stored securely in a separate environment to enable disaster recovery. - **Data Minimization:** Only necessary user data is stored, aligning with privacy best practices.

#### 7.2.6 Alternatives and Trade-offs

| Option | Advantages | Disadvantages |
| --- | --- | --- |
| **SQL (MySQL)** | Strong consistency, clear structure, relational integrity | Lower horizontal scalability |
| **NoSQL (MongoDB)** | High scalability, flexible schema | Weaker consistency, complex joins |
| **Single Database** | Easier management, simple deployment | Potential bottleneck at scale |
| **Multiple Databases** | Better fault isolation, scalable by component | Increased complexity |

#### 7.2.7 Summary

The chosen SQL-based data management strategy provides the right balance between consistency, performance, and maintainability. It supports transactional reliability, simplifies data handling through normalization, and aligns well with the architectural goals of a modular, service-oriented movie ticketing platform.