Movie Theater Ticketing System

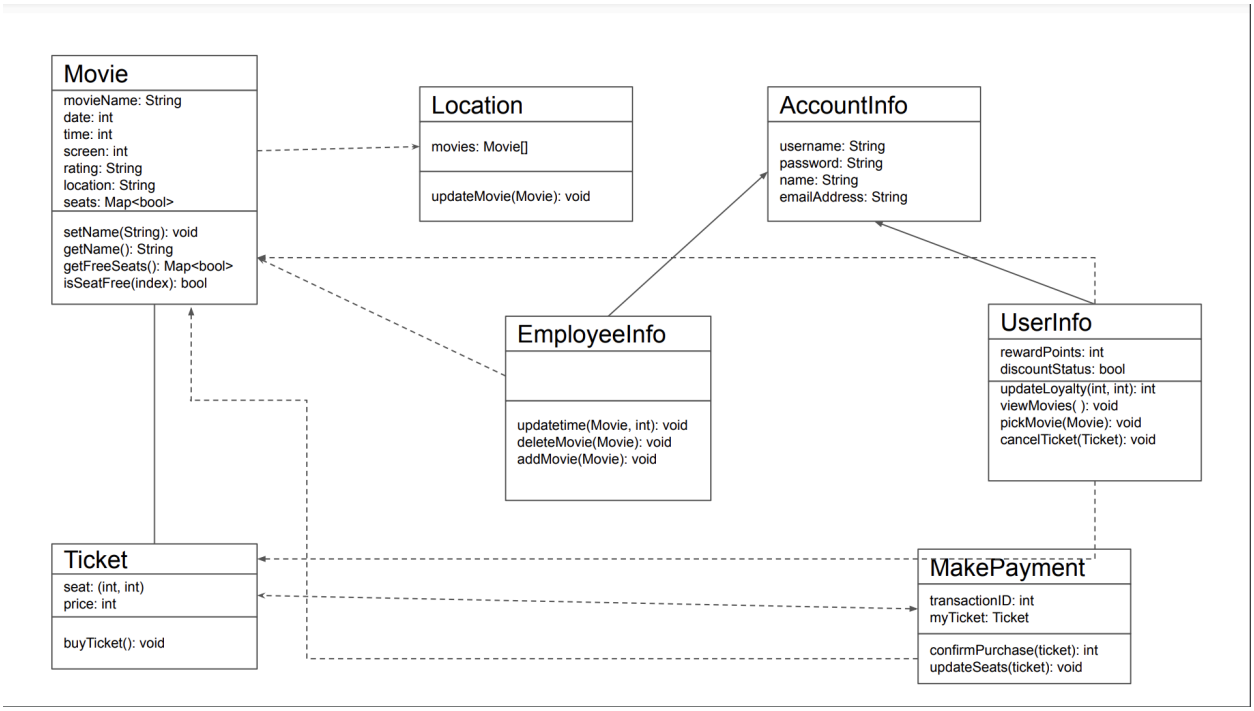
Team Members: Liam Hayes, Youngmin Park, Alex Colmenar

‘

Brief Overview:

The Movie Theater Ticketing System focuses on the stakeholders and applications that allow for online ticket sales, distribution, and marketing of movies. It will provide an accessible and intuitive interface for users to purchase tickets through a browser. The system will ensure it manages showtimes, ticket availability, and user transactions, ensuring consistency and security for the different movies. It will display movie reviews from critics updated constantly for live reviews and will not produce its own reviews.

UML Diagram:



Description:

Class: Movie

- Attributes:
  - `movieName: String` - The name of the movie.
  - `date: int` - The showing date of the movie.
  - `time: int` - The showing time of the movie.
  - `screen: int` - The screen number where the movie is shown.
  - `rating: String` - The rating of the movie (PG, PG-13, R).

        - `location: String` - The location of the movie showing.
        - `seats: Map<bool>` - A map representing the seats and their availability (true for available, false for taken).
- Operations:
    - `setName(String): void` - Sets the name of the movie.
    - `getName(): String` - Retrieves the name of the movie.
    - `getFreeSeats(): Map<bool>` - Returns a map of seats with their availability status.
    - `isSeatFree(index): bool` - Checks if a specific seat is free.

Class: Ticket

- Attributes:
    - `seat: (int, int)` - The row and column of the seat.
    - `price: int` - The price of the ticket.
- Operations:
    - `buyTicket(): void` - Processes the ticket purchase.

Class: Location

- Attributes:
    - `movies: Movie[]` - An array of `Movie` objects available at the location.
- Operations:
    - `updateMovie(Movie): void` - Updates the movie details.

Class: AccountInfo

- Attributes:
    - `username: String`
    - `password: String`
    - `name: String`
    - `emailAddress: String`

Class: EmployeeInfo

- Operations:
    - `updateTime(Movie, int): void` - Updates the time for a movie showing.
    - `deleteMovie(Movie): void` - Removes a movie from the schedule.
    - `addMovie(Movie): void` - Adds a new movie to the schedule.

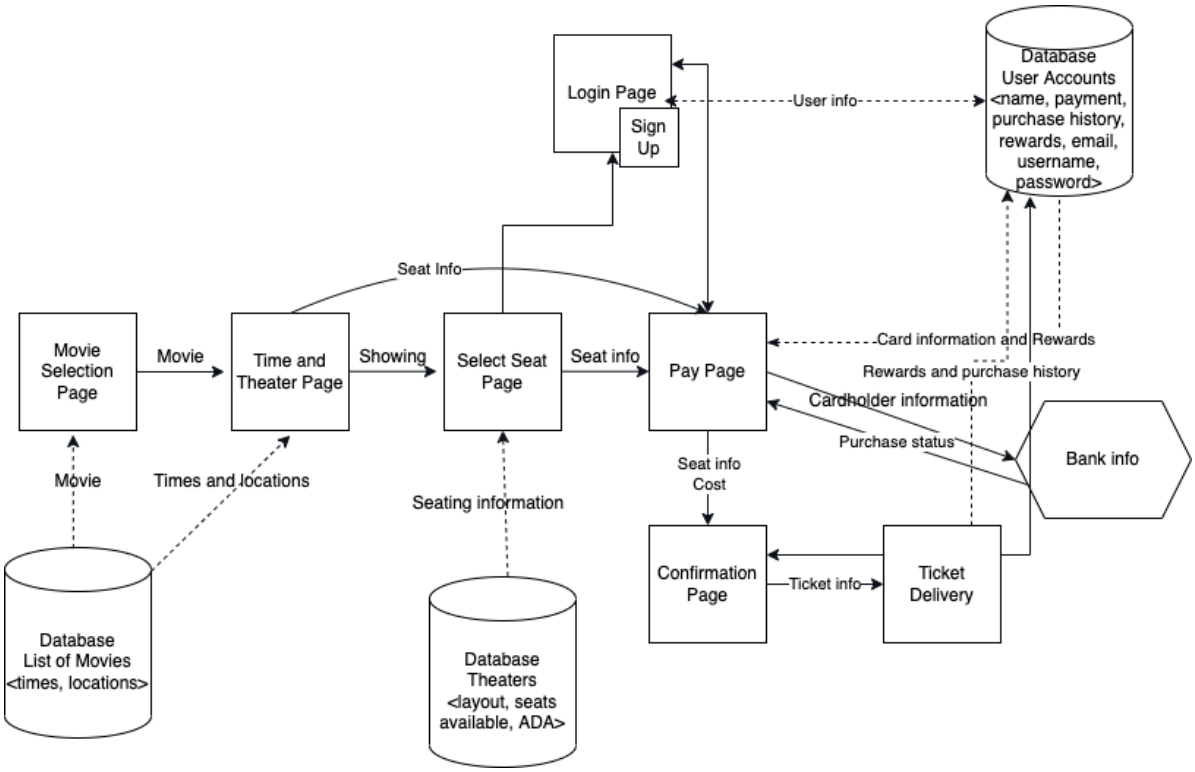
Class: UserInfo

- Attributes:
  - `rewardPoints: int` - The loyalty points of the user.
  - `discountStatus: bool` - Indicates if the user is eligible for a discount.
- Operations:
  - `updateLoyalty(int, int): int` - Updates loyalty points for the user.
  - `viewMovies(): void` - Displays available movies.
  - `pickMovie(Movie): void` - Selects a movie for ticket purchase.
  - `cancelTicket(Ticket): void` - Cancels a ticket purchase.

Class: MakePayment

- Attributes:
  - `transactionID: int` - The ID of the transaction.
  - `myTicket: Ticket` - The ticket associated with the payment.
- Operations:
  - `confirmPurchase(ticket): int` - Confirms the purchase of a ticket and returns a transaction ID.
  - `updateSeats(ticket): void` - Updates the seat availability after a ticket purchase.

SWA Diagram:



Description:

Movie Selection Page: Users start here to browse movies. This component interacts with a database to retrieve a list of movies, including their showtimes and locations.

Time and Theater Page: After selecting a movie, users are directed here to choose a specific time and theater location, pulling data from a database that lists movie times and locations.

Select Seat Page: Users pick their seats based on availability, which is checked against a seating database for the selected theater.

Pay Page: Payment information is collected and processed in this component, with secure transactions being a priority.

Confirmation Page: This component confirms the successful transaction, providing users with their ticket information and seat details.

Ticket Delivery: Manages the distribution of tickets, ensuring users receive them in their chosen format.

The databases—**User Accounts**, **List of Movies**, and **Theaters**—store all relevant data and interact with the system's pages to provide up-to-date information and maintain the integrity of user data and transactional records.

The **Login Page** and **Sign Up** are connected to the **User Accounts** Database, handling user authentication and account creation. Additionally, the system interfaces with external **Bank Info** for payment processing, ensuring secure financial transactions.

Connectors between these components represent data flow and user navigation paths, indicating how users move through the system and how data is exchanged to support functionalities like movie selection, seat reservation, and payment processing.

Development Plan and Timeline

- Partitioning of Tasks:
    - UI Design and Development
    - Backend System Development
    - Payment Integration
    - Testing and Quality Assurance

- Team Member Responsibilities:
    - Alex Colmenar: UI Design and Frontend Development
    - Liam Hayes: Backend System Development, Testing and Quality Assurance
    - Youngmin Park: Database Management and Integration
- Timeline:
    - Week 1-2: Requirement Analysis and Planning
    - Week 3-4: UI Design
    - Week 5-8: Backend Development
    - Week 9-10: Integration and Testing
    - Week 11: Final Review and Deployment

Verification Test Plan

Introduction to Testing

- Purpose of this test is to verify all components of the Ticketing System functions as expected. We want to ensure a seamless user experience, tight security, and data integrity.

Test Strategy

- The test plan will use a multi-tiered testing plan, including functional test, software system tests, and unit tests to cover the entirety of the system.
- It would implement automated and manual testing to ensure efficiency and coverage over every problem that could occur within the system.

Test Levels

- Unit Testing: Focus on testing the smallest pieces of code and make sure each function has the correct and intended output.
- Functional Testing: Focus on testing specific functions within the software system.
- System Testing: Test the system as a whole and make sure it runs as expected on the surface and ready to be used.

Test Cases

- Each case follows a structured approach to make sure it has thoroughly found any errors and provide clarity in how the test is done. In total this covers the unit, functional, and system levels of the Movie Theater Ticketing System.

Failure Management

- Provides a plan for an analysis of the failed areas of the system to find and understand the root cause of the issues at hand. This not only solves the critical issues, but being documented will help refind the SDLC preventing any similar issues from occurring in the future development.

Book.xlsx

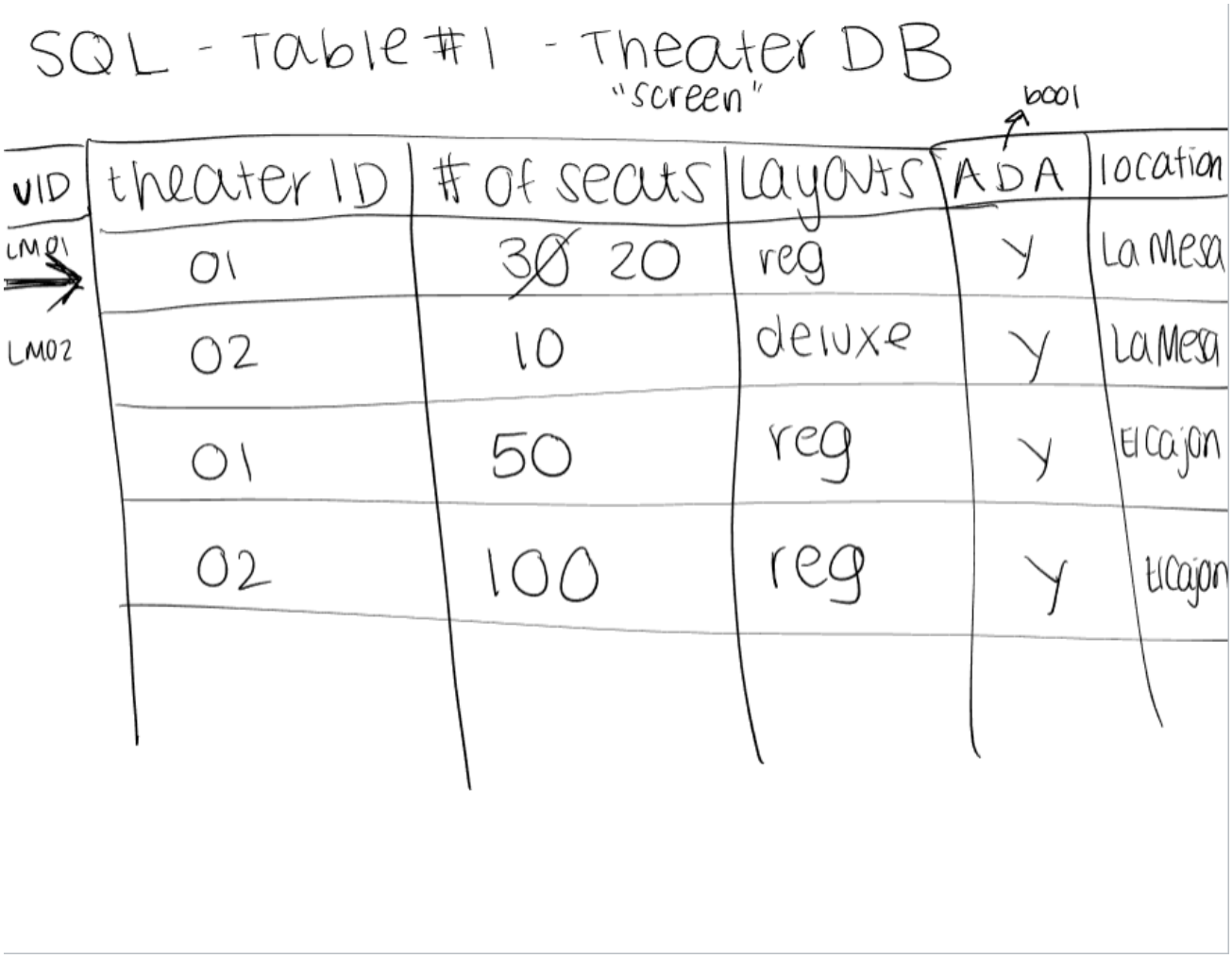
Data Management Strategy

Our data management strategy employs a hybrid database approach to provide the necessary balance between structured data management, scalability, and flexibility. We incorporate both SQL and NoSQL databases to leverage their strengths and ensure optimal performance.

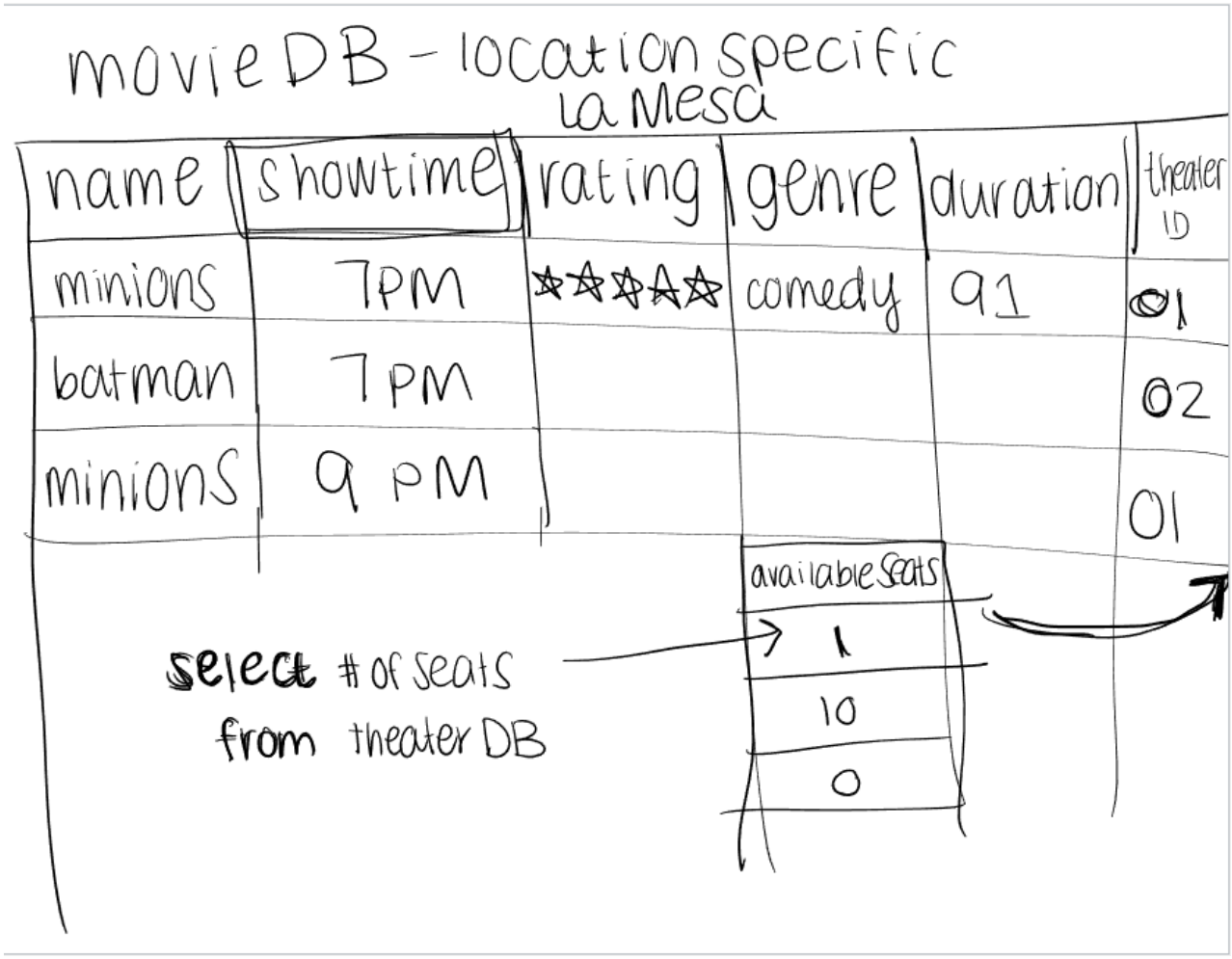
SQL Databases:

For transactional data that requires relational integrity and complex querying, such as user account details, ticket transactions, and showtime schedules, we use SQL databases. They offer ACID properties, ensuring reliable and consistent transactions necessary for the integrity of our system.

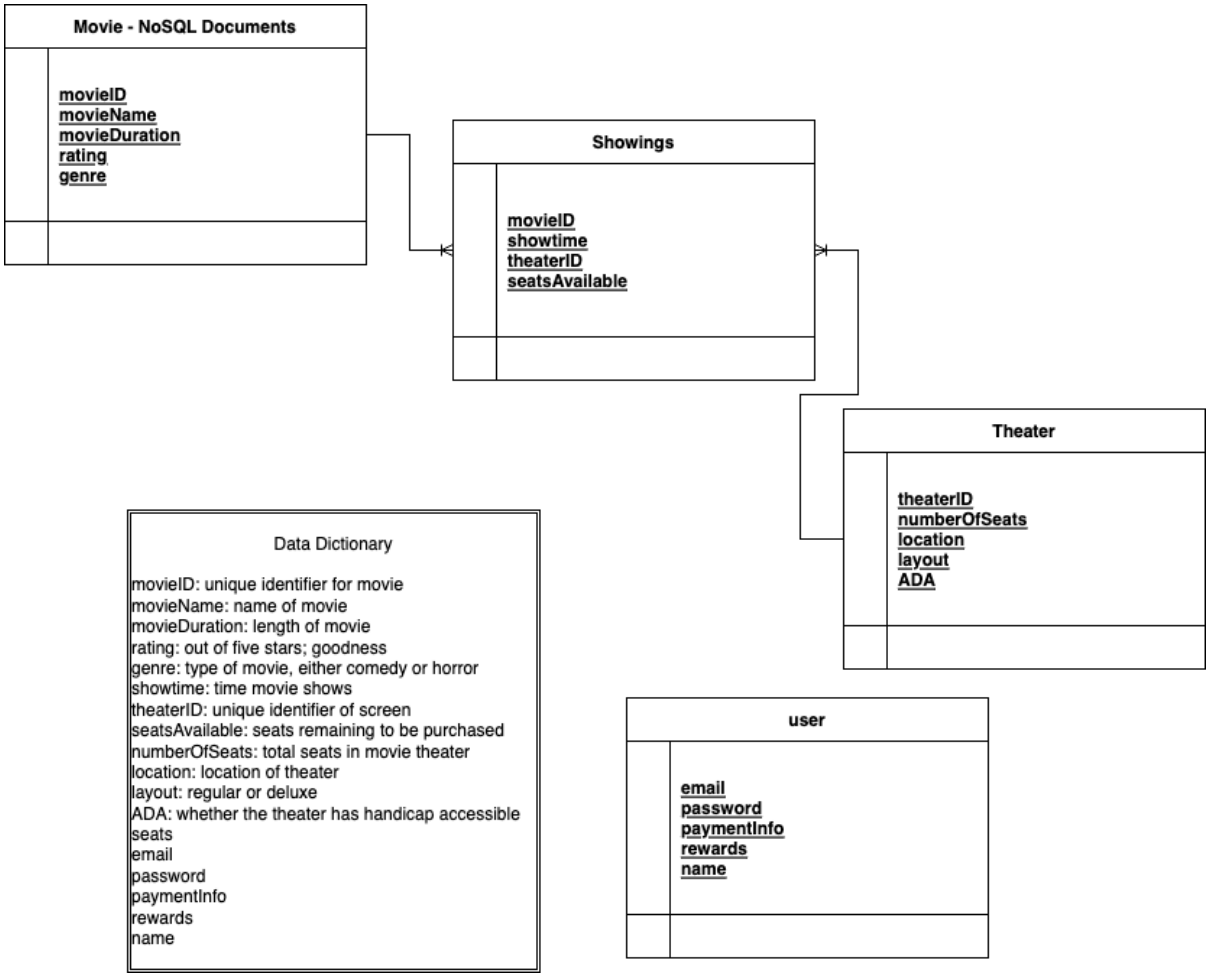
One key component of our SQL database is the Theater Information Table. This allows us to efficiently manage theater-specific information, which is crucial for seat selection and maintaining an accurate representation of each theater's capabilities and layout.



Furthermore, our system features a Movie Database. This design facilitates the relationship between movies and their showtimes at various theaters, while also providing users with valuable information for making informed decisions on movie selections.



Entity-Relationship Diagram for SQL Documents:



This Entity-Relationship Diagram represents the structure of the SQL documents within our system. It highlights the data dictionary for the movie entities, showings, and theater information. The attributes within these documents include unique identifiers, movie details, showtime specifics, and theater attributes. This design supports rapid retrieval and updating of movie and theater data, which is essential for our application's performance and user experience.

NonSQL Databases:

To manage semi-structured or unstructured data like movie reviews, user comments, and potential high-velocity data, we integrate NoSQL databases. These databases provide scalability and the flexibility to store data without a predefined schema, accommodating a variety of data types and structures.

Security Measures:

We enforce strict security protocols, including data encryption at rest and in transit, to protect sensitive user information and payment details. Our APIs, secured with OAuth, ensure that data access is tightly controlled and only authorized operations are permitted.

Backup and Recovery:

A robust backup and recovery strategy is in place to protect against data loss and to ensure high availability. Regular, encrypted backups are taken, and our infrastructure is designed to allow for quick recovery in the event of a failure.

Trade-offs and Alternatives:

While the hybrid database approach provides both flexibility and performance, it also introduces complexity in managing and integrating multiple systems. We have considered this in our design decisions and believe that the benefits, such as improved scalability and flexibility, outweigh the costs. Alternative approaches, such as using a single versatile database system, were considered but found to be less optimal for our use case.

In conclusion, our data management strategy is designed to be dynamic, secure, and efficient, supporting the needs of our Movie Theater Ticketing System and ensuring a seamless and responsive user experience.