Building a simple movie booking system as explained below.

**Core features:**

* This system focuses only on a use case where multiple users trying to book tickets to the same movie show. However, make sure the data model supports multiple movies, shows, and cinema halls. ===> (Done)
* A user can choose up to 6 seats from a cinema hall. (Done)  🡺 SpringJPA
* A user has to pay for the seats within 2 minutes. If not, the seats will be released for other users to book.  ===> (Done) Transaction Management
* Assume payment is handled by a third-party payment system. The outcome of the payment API will be either success or a failure. ===> (Done) Hystrix
* Seats are blocked on a first-come-first-served basis. Unfortunately, if the same seat is chosen by more than one user, then pick one user who blocks the max number of seats. If there is a tie between users, then pick one user randomly and reject everyone else.  =🡺 (Done)  Blocking Queue
* Scalable system - multiple instances of the application should be able to work together to book tickets ===> (Done)  Spring Boot Microservices
* Assume a single instance of a database is used for maintaining the state ===> (Done) - Mysql

**API:**

* Provide APIs to access this system's service  ===> (Done) – Spring Security
* Secure the APIs ===> (Done) – Spring Security

Various concepts for Transaction Mangement planned to used

**1)Using basic synchronization**

**2)ReadWriteLocks**

**3)CompletableFuture**

**4)@Transactional in spring**

@Transactional(isolation = Isolation.SERIALIZABLE)

@Transactional( propagation = Propagation.SUPPORTS , readOnly = true )

**5)Hibernate locks ( optimistic locking, pessimistic locking by version , timestamp) 2 phase commit, 3 phase commit.**

**6)Plain JDBC**

* Local Transactions
* Global / Distributed Transactions

**RestTemplate – synchronous Communication of microservices**

ReactiveCrudRepository

**WebClient – Asynchronous Communication of microservices**