**movie-booking**

**Project overview**

* Movie booking app

**Tech stack**

* Spring boot
* Restful API
  + CORS
  + Spring data with mongoDB repositories
  + Spring security,Oauth2,JWT,
* MongoDB
* Docker - Docker compose
* Testing : JUnit/Mockito/MockMVC/AssertJ/Hamcrest
* Java 8

Azure

Azure api gateway,

Ingress controller,

K8,Kubernetes(azure hosted,)Dapr,Azure alerts,Monitoring tools

Azure Keyvaults,RBAC,Active directory

Api tools

Swagger

Code quality

SonarQube

Build tools

Jenkins

Source repository(Git)

Testing tools

Selenium,Graphana,Postman

Design patterns

Circuit breaker(Hystrix)

Singleton

Per service database design pattern

API Gateway design pattern(Zuul/Azure api gateway/Ingress)

CQRS Design pattern/Messaging pattern (Even driven/event notification->Kafka)

**API docs /v1/api**

* Customer
  + GET /customers : queries all customer
  + GET /customers/id: queries the customer wiht mongo object id
  + POST /customers : save customer record json
  + DELETE customers/id: delete the customer by mongo object id
  + GET : customers/search/findByEmail?email=hienminhnguyen711@gmail.com: find customer by customer id

**How to run**

* Locally
  + mvn spring-boot:run
* Docker container
  + docker-compose up
  + Attach bash to the running Mongo container
    - docker exec -i -t springboot-mongo /bin/bash
* Swagger API docs
  + Open Swagger API doc : localhost:8080/swagger-ui.html

**Basic commands to run with mongoDB**

* show dbs
* use dbbasename
* show collections
* db.movie.find()

Graphical user interface, text, application

Description automatically generated

## Below are the list of API endpoints available for the user to make api integrations for there platform

### All the Admin or merchant api endpoint starts with the /api \*

### To add the City to the DB

URL: /api/city Method: POST Request: { "name":"Bengaluru", "pincode":"590056", "state":"Karnataka" } Response: { "name": "Bengaluru", "state": "Karnataka", "id": 2, "pincode": "590056" }

### To add theater to the BD

here theater reference to the key\_id of the city as primary key so make sure to pass the valid city id URL: /api/{CityID}/theater Method: POST Request: { "name":"Finox mall", "area":"Koramangala" } Response: { "t\_name": "Finox", "t\_area": "Kormangala", "t\_id": 3 }

### To add the Movie to the DB

URL: /api/movie Method: POST Request Body: { "\_name":"WAR 3", "\_director":"WAR 3 directors", "\_description":"description about the WAR 3 movie here" } Response Body: { "\_name": "WAR 3", "\_director": "WAR 3 directors", "\_description": "description about the WAR 3 movie here", "\_id": 6 }

### To add a show

URL:

* TO add a show pass the theater Id and Movie id as a URL parma and the show time in the request body \* URL: /api/{theater\_id}/{movie\_id}/show Method: POST Request: { "time":"4/4/2020 - 3:30 PM" } Response: { "show\_Id": 8, "show\_time": "1/4/2020 - 3:30 PM " }

### All the End user api will start with the /user

When user comes to the app make a get call to below API to fetch the name of city available on the app

URL: /user/city Method: GET Response Body: [ { "name": "Munavalli", "state": "Karnataka", "id": 1, "pincode": "591117" }, { "name": "Bengaluru", "state": "Karnataka", "id": 2, "pincode": "590056" } ]

To get the list of Theaters Available on the city using the city id got in the previous call

URL: /user/{cityID}/theater Method: GET Response Body: [ { "t\_name": "Finox", "t\_area": "Kormangala", "t\_id": 3 }, { "t\_name": "Bhoda theater", "t\_area": "Kengeri", "t\_id": 5 } ]

### after user selects the theater pass the theater id to the below api and in return you'll get a list of movie available for the movie \*

URL: /user/theater/{theaterID}/movie Method: GET Response Body: [ [ 6, "WAR 3", "description about the WAR 3 movie here" ] ]

DB Design



Sequence diagram of movie ticket booking app

Chart, box and whisker chart

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Document will be updated later as per the completion of project



1. **Integrate with payment gateways**

There are 3 ways to integrate with payment gateways

**1. On-site payments:** dealt with on their own servers where the checkout experience and installments taking care of all work through your system.

**2. Checkout on-site, payment off-site:** Through this strategy, the front-end checkout will occur on your site, yet the Payment taken care of happens through the entryway's back end.

Like a redirected Payment gateway, this strategy can further develop the Payment processes while ensuring extended security rearward.

**3. Redirects:** Redirect every now and again integrates decisions for elective payment methodologies, similar to an association allowing the use of PayPal. Right when the entry takes a client to a PayPal payment page to manage the all-out trade, it transforms into a Redirect.

Redirects partake in the advantage of ease for the retailer. In any case, they similarly mean less control for the merchant - and a second step for clients.

Multiple payment gateways are

1.Paytm

2.SbiePay(Internet Banking, card payment,UPI,Neft,CSCS,Cash..etc)

SBIePay is a Payment Aggregation service provided by the most trusted Banking Brand in India

<https://www.sbiepay.sbi/secure/home>

1.Paytm

The flow Digram for Integration With Paytm can be explained as follows-

A diagram of a payment gateway

Description automatically generated

1. **How to protect against OWASP top 10 threats.**

**Top 10**

**1.Injection(SQL Injects need to be avoided)**

**2.Broken Authentication-Session Management(Proper handling)**

**3.Cross Site Scripting (XSS)**

**4.Insecure Direct Object references**

**5.Security Misconfiguration**

**6.Sensitive Data Exposure**

**7.Missing Function Level Access Control**

**8.Cross Site Request Forgery(CSRF)**

**9.Using components with Known Vulneribilities**

**10.Unvalidated Redirects and Forwards**

**11.CORSS**

1. **How do you monetize platform?**

**There are multiple monitoring tools**

**Jmeter**

**Graphana**

**Kubenetes(Liveness/Readyness)**

**Azure monitoring Tools ( logs**

**CPU,Memory(Azure Alerts)**

**Eureka Service Registry**

**Hystrix Dashboard(status of services)**

**API Gateway monitoring tools**

1. **How will you scale to multiple cities, countries and guarantee platform availability of 99.99%?**

**Admin screens for Cities,Countries, to increase, update, delete..etc**

1. **Transactional Scenarios**

Spring Boot implicitly creates a proxy for the transaction annotated methods. So for such methods, the proxy acts like a wrapper, which takes care of creating a transaction at the beginning of the method call and committing the transaction after the method is executed

Booking Transaction involves

(Seating micro service,

Theatre Micro service,

Movie micro service,

Payment Micro service)

**How to handle those cases where two people tries to access the same seat at the same time?**

Lets take SeatBook and Transactions class which will be called from the main class.

Here, We expand a bit the payment process.

In SeatBook class we will have reference to Transaction class also.

Now to ensure when two persons are trying to access the same seat almost at the same time then we would use Synchronized method of Thread class and will call a thread belong to each log in user.and Optimistic lock.

If two transactions are executing concurrently - till the first transaction is committed the existing records cannot be changed by second transaction but new records can be added. After the second transaction is committed, the new added records get reflected in first transaction which is still not committed.

**What is Transaction Isolation?**  
Transaction Isolation defines the database state when two transactions concurrently act on the same database entity. It involves locking of database records. So it describes the behaviour or state of the database when one transaction is working on database entity and then some other concurrent transaction tries to simultaneously access/edit the same database entity.  
four isolation levels, Isolation is one of the ACID (Atomicity, Consistency, Isolation, Durability) properties.

Using Spring we can change the isolation level to suit our business logic.

Nothing but read should be consistent with Update.

Example scenarios

A diagram of a process

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**Database Isolation levels**

CREATE TABLE employee (

empId VARCHAR(10) NOT NULL,

empName VARCHAR(100) NOT NULL

);

Some of the SQL commands I have used for implementing Isolation Levels are

//Show existing transaction isolation level if mysql version >= 8

**SELECT @@TRANSACTION\_ISOLATION;**

//Set transaction isolation level to serializable. Using same syntax

//we can set it to other isolation level.

**SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

//By default auto commit is enabled for mysql transaction. So we will disable it.

**SET AUTOCOMMIT=0;**

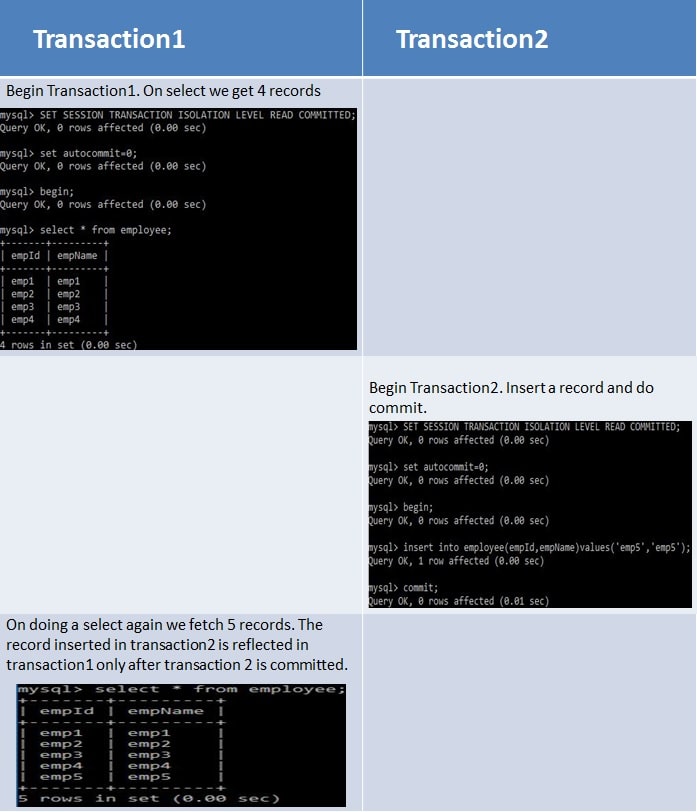
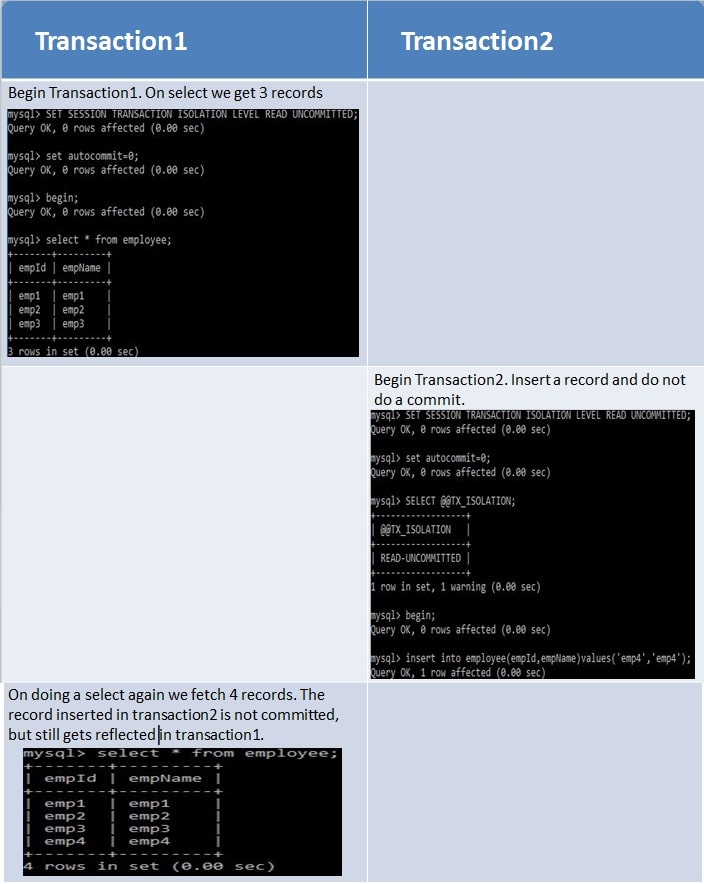
//Start transaction

**BEGIN**

//Commit transaction

**COMMIT**

**The following are the types of Transaction Isolation Levels-**

* + **SERIALIZABLE**If two transactions are executing concurrently then it is as if the transactions get executed serially i.e the first transaction gets committed only then the second transaction gets executed. This is total isolation. So a running transaction is never affected by other transactions. However this may cause issues as performance will be low and deadlock might occur.
  + **REPEATABLE\_READ**  
    If two transactions are executing concurrently - till the first transaction is committed the existing records cannot be changed by second transaction but new records can be added. After the second transaction is committed, the new added records get reflected in first transaction which is still not committed. For MySQL the default isolation level is REPEATABLE\_READ.  
    However the REPEATABLE READ isolation level behaves differently when using mysql. When using MYSQL we are not able to see the newly added records that are committed by the second transaction.  
    
  + READ\_COMMITTED  
    If two transactions are executing concurrently - before the first transaction is committed the existing records can be changed as well as new records can be changed by second transaction. After the second transaction is committed, the newly added and also updated records get reflected in first transaction which is still not committed.  
    
  + READ\_UNCOMMITTED  
    If two transactions are executing concurrently - before the first transaction is committed the existing records can be changed as well as new records can be changed by second transaction. Even if the second transaction is not committed the newly added and also updated records get reflected in first transaction which is still not committed.  
    
  + Summary
  + Dirty Reads - Suppose two transactions - Transaction A and Transaction B are running concurrently. If Transaction A modifies a record but not commits it. Transaction B reads this record but then Transaction A again rollbacks the changes for the record and commits it. So Transaction B has a wrong value.
  + Non-Repeatable Reads - Suppose two transactions - Transaction A and Transaction B are running concurrently. If Transaction A reads some records. Transaction B modifies these records before transaction A has been committed. So if Transaction A again reads these records they will be different. So same select statements result in different existing records.
  + Phantom Reads - Suppose two transactions - Transaction A and Transaction B are running concurrently. If Transaction A reads some records. Transaction B adds more such records before transaction A has been committed. So if Transaction A again reads there will be more records than the previous select statement. So same select statements result in different number records to be displayed as new records also get added.

|  |  |  |  |
| --- | --- | --- | --- |
| * + Isolation Level | * + Dirty Reads | * + Non-Repeatable Reads | * + Phantom Reads |
| * + SERIALIZABLE | * + This scenario is not possible as the second transaction cannot start execution until the first is committed. They never execute parallelly but only sequentially | * + This scenario is not possible as the second transaction cannot start execution until the first is committed. They never execute parallelly but only sequentially | * + This scenario is not possible as the second transaction cannot start execution until the first is committed. They never execute parallelly but only sequentially |
| * + REPEATABLE\_READ | * + This scenario is not possible as any existing record change gets reflected only if the transaction is committed. So other transaction will never read wrong value. | * + This scenario is not possible since any record can be changed only after a transaction has been committed. So multiple select statements before transaction commit will always return same existing records. | * + This scenario is possible as other transactions can insert new records even if first transaction commit has not taken place. |
| * + READ\_COMMITTED | * + This scenario is not possible as any existing record change gets reflected only if the transaction is committed. So other transaction will never read wrong value. | * + This scenario is possible as other transactions can modify existing records even if first transaction commit has not taken place. | * + This scenario is possible as other transactions can insert new records even if first transaction commit has not taken place. |
| * + READ\_UNCOMMITTED | * + This scenario is possible as any record can be read by other transactions even if the first transaction is not committed. So if first transaction rollbacks the record changes then other transactions will have wrong values | * + This scenario is possible since any record can be changed even if a transaction is not committed. | * + This scenario is possible as any record can be inserted even if a transaction is not committed. |

* + Implement Transaction Isolation using Spring Boot
  + When using Transaction Isolation with Spring Boot, the default transaction isolation taken is that of the underlying database. So for our spring boot application the default transaction isolation will be REPEATABLE\_READ since we are using MySQL database
  + .As - [Spring Boot Transaction Management](https://www.javainuse.com/spring/boot-transaction) are transactions and implemented declarative transaction management. We can change the transaction isolation level as required!

1. Project Plan



1. Estimates

