Contents

[Running the application 2](#_Toc452058547)

[REST API Details: 2](#_Toc452058548)

[Frameworks used 3](#_Toc452058549)

[Tables 3](#_Toc452058550)

[Entity Relationship Diagram 3](#_Toc452058551)

[Auditorium Data used in app 4](#_Toc452058552)

[Packages available 4](#_Toc452058553)

[Exception scenarios 5](#_Toc452058554)

[CODE FLOW 5](#_Toc452058555)

[NOTES 6](#_Toc452058556)

[Sample Requests and Responses 7](#_Toc452058557)

[numSeatsAvailable 7](#_Toc452058558)

[Request without levelId: 7](#_Toc452058559)

[Response 7](#_Toc452058560)

[Request with levelId: 7](#_Toc452058561)

[Response 7](#_Toc452058562)

[Sample Screenshot 7](#_Toc452058563)

[findAndHoldSeats 8](#_Toc452058564)

[Request 8](#_Toc452058565)

[Response 8](#_Toc452058566)

[Sample Screenshot 9](#_Toc452058567)

[reserveSeats 9](#_Toc452058568)

[Request 9](#_Toc452058569)

[Response 9](#_Toc452058570)

[Sample Screenshot 9](#_Toc452058571)

[Logic used for allocating Seats(High Level) 10](#_Toc452058572)

[Order of priority 10](#_Toc452058573)

Auditorium Ticketing Service

# Running the application

Go to the Project Root Directory and run the below command

1. mvn package && java -jar target/AuditoriumTicketingSystem-0.0.1-SNAPSHOT.jar

This Command will compile the code, run the tests and start tomcat server in port 8080 where the application is deployed. The backend system is **In Memory H2 Database**.

1. If you just want to build and run the test then just run mvn clean install
2. The application exposes a REST API which has 3 methods detailed below.

# REST API Details:

You can use POSTMAN to send the HTTP Requests

1. numSeatsAvailable without level -GET REQUEST- <http://localhost:8080/numSeatsAvailable/>

This will bring back the total number of available seats in the auditorium

* 1. numSeatsAvailable with level -GET REQUEST- <http://localhost:8080/numSeatsAvailable/1/>

This will bring back the total number of available seats in level 1. If we change the path param to 2, 3 or 4 it will bring back the total number of available seats in the corresponding levels

1. findAndHoldSeats -POST <http://localhost:8080/findAndHoldSeats>

The request should contain the below header for it to work

Content-type: application/json

Request body:

{"email":"rohit\_vs@yahoo.com","numSeats": "2","minLevel":"1","maxLevel":"3"}

This will hold 2 seats for a user between levels 1 and 3 if its available

1. reserveSeats – PUT - http://localhost:8080/reserveSeats/1/rohit\_vs@yahoo.com

# Frameworks used

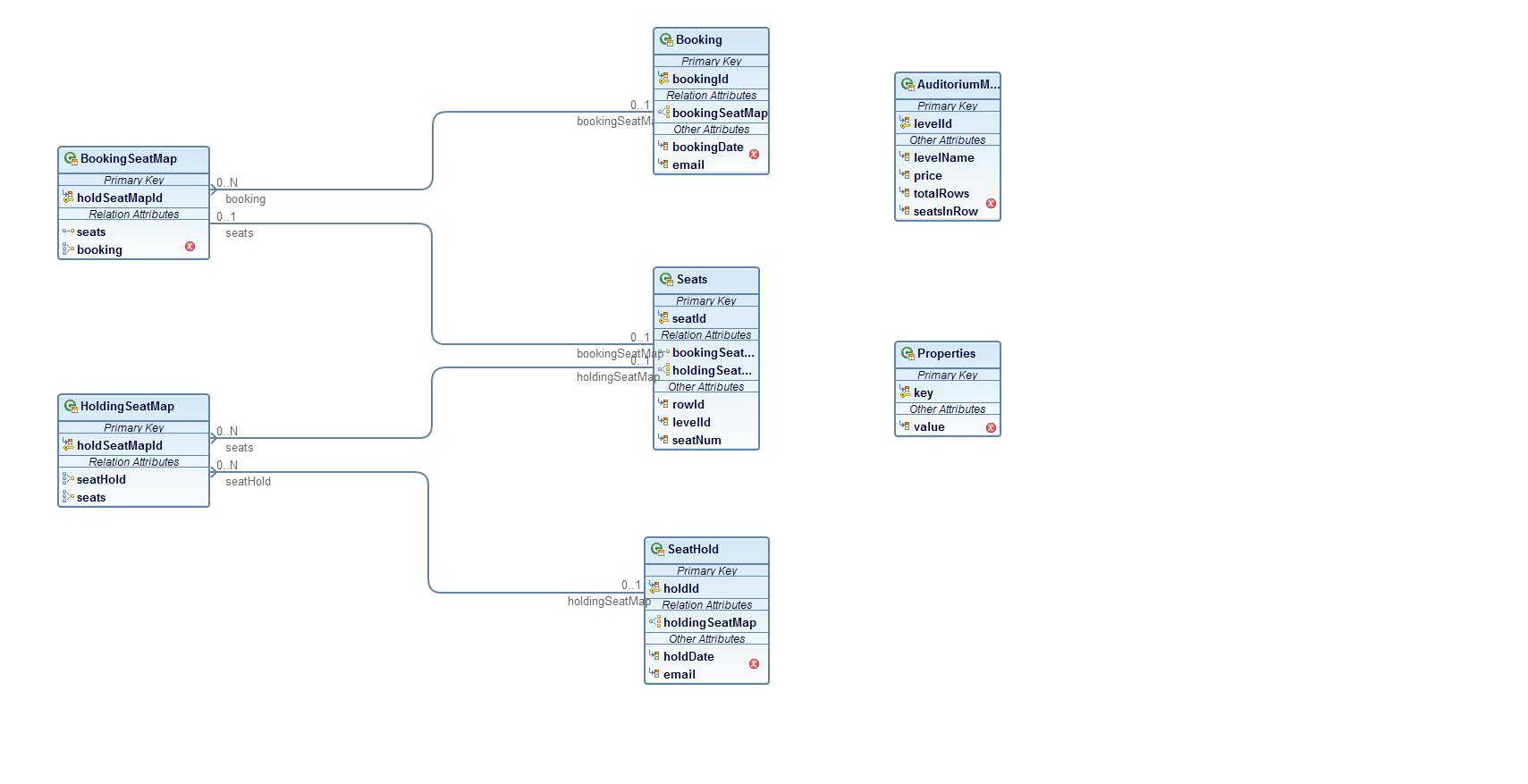
* 1. Spring Boot
  2. Spring Data
  3. JPA2/Hibernate (spring-boot-starter-data-jpa)
  4. H2 (In memory Database)
  5. RestAssured(For Testing)

# Tables

* 1. AUDITORIUM\_METADATA (level\_id, level\_name, price, total\_rows, seats\_in\_row)
  2. SEATS(seat\_id,level\_id,row\_id, seat\_num)
  3. SEAT\_HOLD(hold\_id, email, hold\_date)
  4. HOLDING\_SEAT\_MAP(holding\_seat\_map\_id,hold\_id,seat\_id)
  5. BOOKING (booking\_id, email, booking\_date)
  6. BOOKING\_SEAT\_MAP(booking\_seat\_map\_id,booking\_id,seat\_id)
  7. PROPERTIES(KEY\_NAME, VALUE)(Used for configuring hold time)

## Entity Relationship Diagram

An enlarged copy of this diagram is also placed in the root directory of the project.



# Auditorium Data used in app

1. Number of Levels: 4
2. Number of rows in each level: 2
3. Number of seats in each row : 5
4. Total seats in Auditorium: 40
5. The response of the REST API will be based off of the above numbers
6. Hold Expiry time: 300 sec is configured in PRPERTIES table(This can be changed in the insertscript -data-dev.sql -placed in src/main/resources directory)

# Packages available

1. com.homework.tickets
2. com.homework.tickets.core
3. com.homework.tickets.controller
4. com.homework.tickets.dao
5. com.homework.tickets.exception
6. com.homework.tickets.model
7. com.homework.tickets.repository

# Exception scenarios

1. SeatsNotAvailableException

Thrown when there are no more seats available to be booked or to be held.

1. NotAuthorizedException

Thrown when reserveSeats call is made with a holdId belonging to person A and email belonging to person B

1. TicketHoldExpiredException

Thrown when reserveSeats is called with a holdId that has already expired. Age>300 sec

1. TicketHoldNotFoundException

Thrown when reserveSeats is called with a holdId that doesn’t exist

1. TicketingServiceApplicationException- general exception for all other error scenarios

# CODE FLOW

All the four REST calls go through a similar flow.

Sequence of classes

1. com.homework.tickets.controller.TicketingServiceController
2. com.homework.tickets.controller.TicketingServiceController
3. com.homework.tickets.controller.TicketingServiceDAOImpl

# NOTES

1. The application exposes a REST API which has four methods(detailed above).
2. The backend system is In Memory H2 Database.
3. There are 2 profiles for the data setup(Dev and Test). When the tests run, the test profile is utilized. The Application connects to H2 In Memory Database which has the tables and data that the application will utilize. For the tests, the tables will be pre-loaded with data in all tables.
4. When the application starts in dev profile, all the tables get created. Only AUDITORIUM\_METADATA, SEATS AND PROPERTIES Table will have data. Other tables will be empty. As and when the seats get hold or booked, corresponding tables will get created
5. When the tests run, it starts the app server in a random available port and sets up data in H2 in-memory database. All of the test data is available in data-test.sql.
6. **AuditoriumTicketingSystemApplicationTests** class runs the tests for numSeats and Exception scenarios
7. **TicketingSystemSimulatorTests** runs the tests for Holding and booking. It brings the database up in a particular state. There is just one test method in this class but it does a series of actions like holding, then booking, then hold again. Basically it simulates user behavior and then runs assert statements to make sure the output is as expected.

# Sample Requests and Responses

## numSeatsAvailable

This service takes an optional path variable which should be the level Id

### Request without levelId:

GET - <http://localhost:8080/numSeatsAvailable/>

### Response

{

"numSeats": 40

}

### Request with levelId:

<http://localhost:8080/numSeatsAvailable/1>

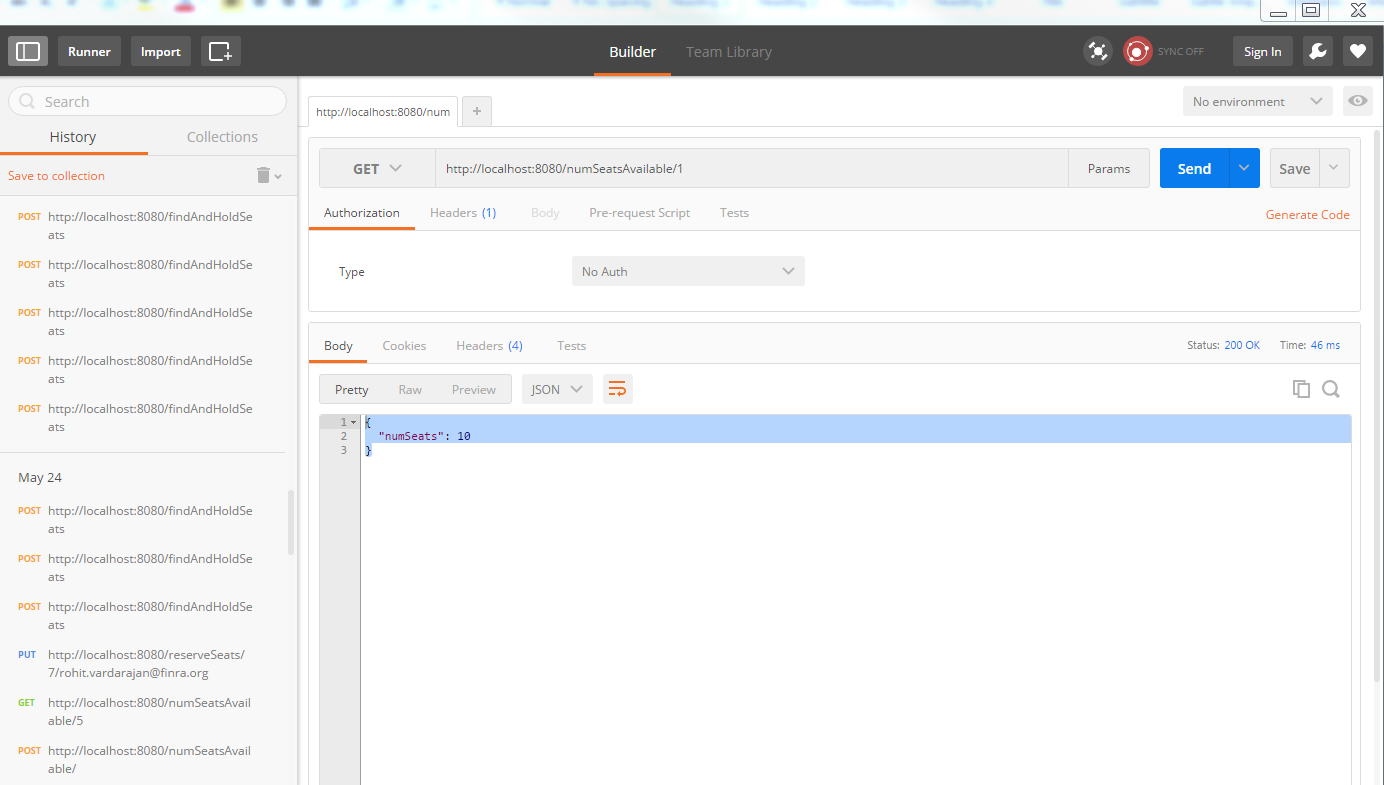
### Response

{

"numSeats": 10

}

### Sample Screenshot



## findAndHoldSeats

The request should contain the below header for it to work

Content-type: application/json

POST - http://localhost:8080/findAndHoldSeats

### Request

{"email":"rohit\_vs@yahoo.com","numSeats":"2","minLevel":"1","maxLevel":"3"}

### Response

{

"@id": 1,

"holdId": 1,

"holdDate": 1464291195511,

"email": "rohit\_vs@yahoo.com",

"holdingSeatMap": [

{

"@id": 2,

"holdSeatMapId": 1,

"seatHold": 1,

"seats": {

"seatId": 1,

"rowId": 1,

"levelId": 1,

"seatNum": 1,

"holdingSeatMap": [],

"bookingSeatMap": null

}

},

{

"@id": 3,

"holdSeatMapId": 2,

"seatHold": 1,

"seats": {

"seatId": 2,

"rowId": 1,

"levelId": 1,

"seatNum": 2,

"holdingSeatMap": [],

"bookingSeatMap": null

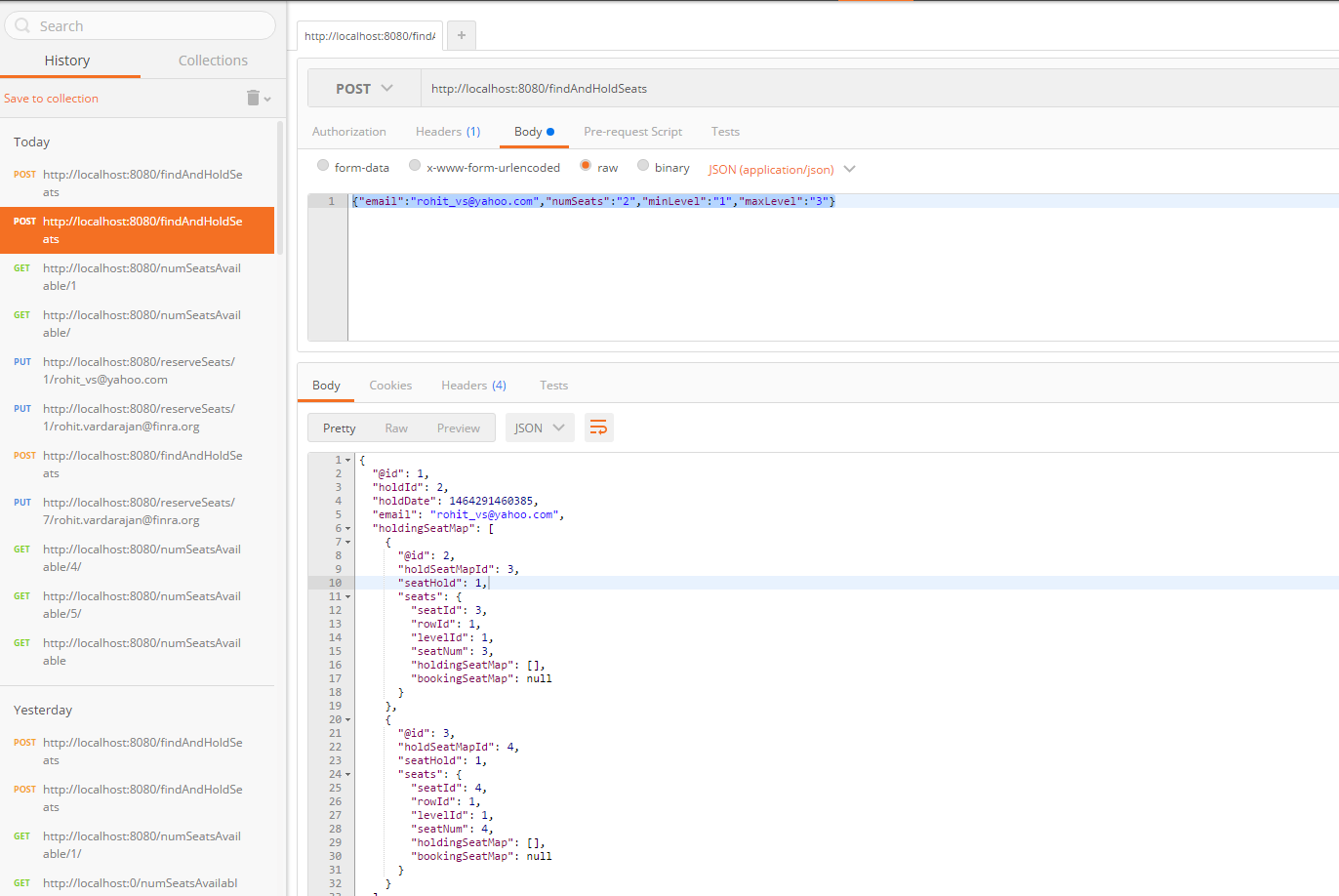
}

}

]

}

### Sample Screenshot



## reserveSeats

### Request

PUT- <http://localhost:8080/reserveSeats/4/test@yahoo.com>

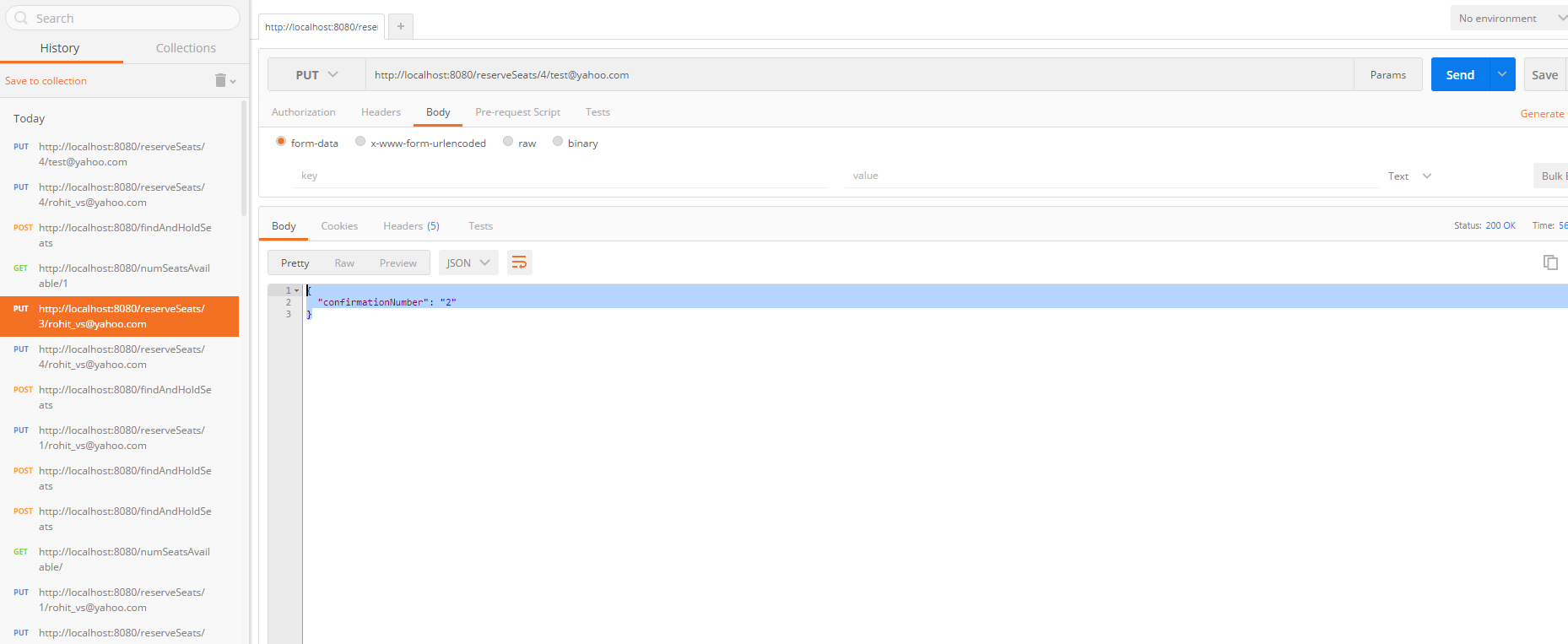
### Response

{

"confirmationNumber": "2"

}

### Sample Screenshot



# Logic used for allocating Seats(High Level)

All of the below logic is in com.homework.tickets.core.GreedySeatAssigner. assignSeatsInSameLevel

and TicketingServiceDAOImpl.findAndHoldSeats

## Order of priority

1. Assign seats in the lowest level possible that the customer requested
2. Assign seats that are next to each other(in the same row) so that the customer sits together with his family/friends
3. If #2 not possible, assign first available seats in the lowest level possible. In this case the seatids can be far apart but in the same level.
4. If no seats are found that would allow customer to be seated together in the same level, then allocate seats across levels.

**Use Case: User tries to hold 3 seats in within level 1 and level 3**

1. The system searches seats from level 1 to level 3
2. In each level it searches seats from row 1
3. It tries to find first available 3 continuous seats in row 1
4. If it is unable to find continuous seats in row 1 it moves on to the next row and applies the same logic in next row
5. If it is unable to find continuous seats in every row of level 1, it assigns the first 3 available seats in level 1.(the seats will not be together in this case but will be in same level)
6. If the system doesn’t find the seats in level 1 at all then it moves on to the next level and applies the same logic(steps 1-5)
7. If the system is not able to find seats in any level then it searches all levels 1 to 3 and allocates the 1st 3 seats(The seats will be in different levels in this case)
8. If the system is unable to find seats across all levels, it throws a SeatNotAvailableException.