Walmart Ticket Services

Designed and Developed By

Sachin Garg

Contents

[System Introduction: 3](#_Toc443854672)

[Assumptions: 3](#_Toc443854673)

[Features: 3](#_Toc443854674)

[Technology: 3](#_Toc443854675)

[Sequence Diagram: 4](#_Toc443854676)

[Ticket Service Architecture: 8](#_Toc443854677)

[Controller Layer API’s: 9](#_Toc443854678)

[Service Layer API’s: 9](#_Toc443854679)

[DAO Layer API’s: 10](#_Toc443854680)

[Tables: 11](#_Toc443854681)

[Table Structure Diagram: 11](#_Toc443854682)

[Entity Classes: 11](#_Toc443854683)

[Unit Testing: 13](#_Toc443854684)

Walmart Ticket Services

# System Introduction:

Implement a simple ticket service that facilitates the discovery, temporary hold, and final reservation of seats within a high-demand performance venue. This service will perform some basic functionality as it is a part of assignment only. It is only POC right now. It is Spring Rest web service right now.

1. Check Number of Seats Available per level.
2. Find And Hold available seat.
3. Reserve seat

# Assumptions:

1. We have populated the data during server initialization to be utilized to perform application specific operations.
2. Max Level seats will be given preferences first then Min Level.
3. Holding time will be 20 milliseconds and cron will check status every 10 milliseconds.
4. Orchestra Level – This level will be identified by 1.
5. Main Level – This level will be identified by 2.
6. Balcony 1 – This level will be identified by 3.
7. Balcony 2 – This level will be identified by 4.

# Features:

1. Check Available Seats – Check number of available seats.
2. Hold – Hold seats based on number of seats and level.
3. Reserve – Reserve seat and change the status either to HELD or RESERVE.
4. Release Hold. – Release the hold after 20 seconds.

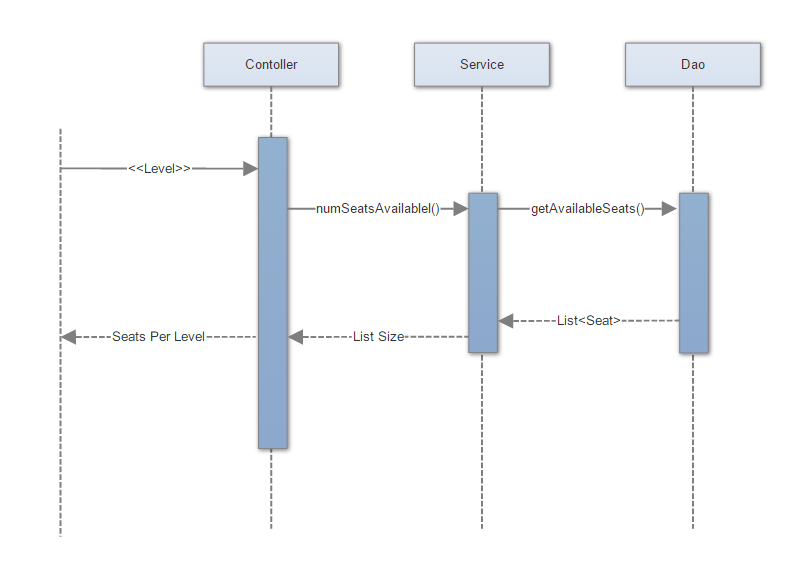
# Technology:

This assignment is tried and entirely on Java based spring annotation configuration with no use of xml.

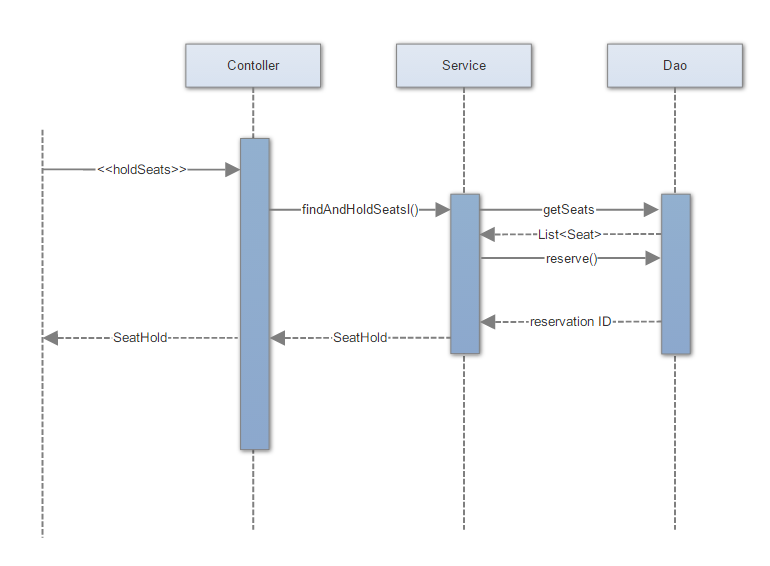
JDK 1.8, Spring Rest service, Jetty, Mockito, Embedded HsqlDB, Hibernate, JPA, Maven, Postman, log4j, eclipse mars.

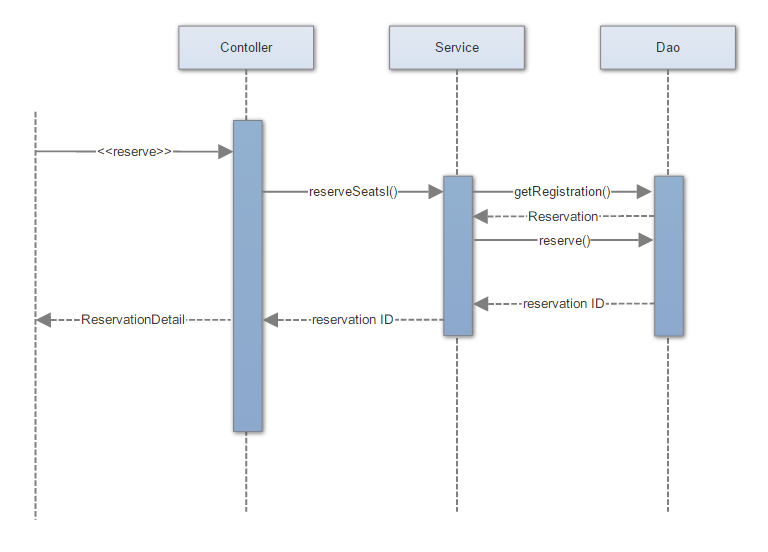
# Sequence Diagram:

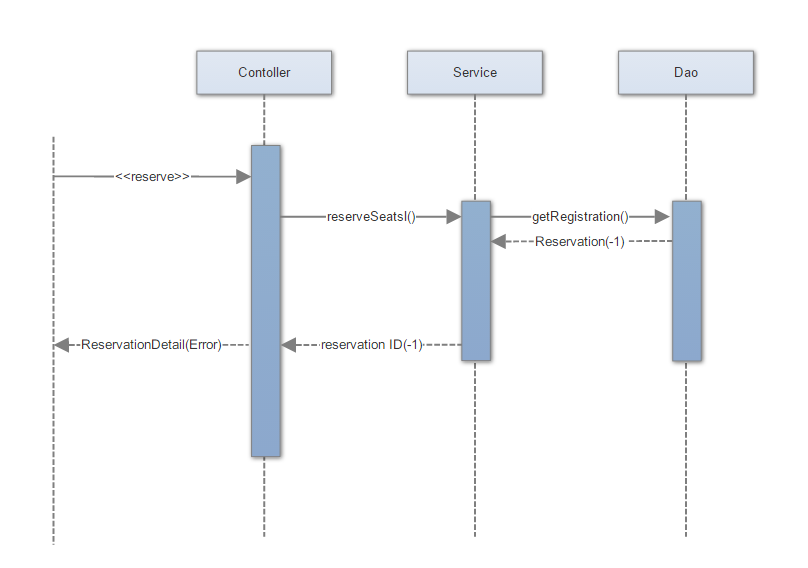
Get Available Seats



Hold Seats



Reserve Seats

Reservation Exception Scenario

# Ticket Service Architecture:

**Data Base**

**DAO Layer**

**Service Layer**

**Controller**

Get Available Seats Request

Seats/Level Response

Hold Seats Request

Seat Hold Response

Reserve Request

Reservation Detail Response

# Controller Layer API’s:

Controller class is ApplicationController, which will receive all http operations i.e. get/post and will parse it to service layer interface, TicketService.

1. GetAvailableSeats: This web service will expose Seats per Level object which will containing information of level and seat numbers. Input parameter for it will be Level ID using Http Get Operation. Return Value will be int.

Get Method

@RequestMapping("getAvailableSeats/{id}")

@ResponseBody

**public** SeatsPerLevel getAvailableSeats(@PathVariable Long id)

1. HoldSeats: This web service will receive numSeats, min level, max level, customer email as a part of Http POST Operation and will return SeatHOLD object which will have Seat ID and Seat Numbers.

Post Method

@RequestMapping(value = "hold", method = RequestMethod.***POST***)

@ResponseBody

**public** SeatHold holdSeats(@RequestBody HoldRequest request)

1. Reserve Seats: This service will receive Reserve Request object which will containing seat hold ID and customer email as a part of Http POST Operation and will return Reservation Detail with Reservation ID, email address and Message.

Post Method

@RequestMapping(value = "reserve", method = RequestMethod.***POST***)

@ResponseBody

**public** ReservationDetail reserve(@RequestBody ReserveRequest reserveRequest)

# Service Layer API’s:

Interface for Service layer is TicketService

1. numSeatsAvailable: This is a service to expose number of available seats per level. Input parameter for it will be Level ID using Http Get Operation. Return Value will be int.

Interface Method

**public** **int** numSeatsAvailable(Optional<Integer> venueLevel);

1. findAndHoldSeats: This service will receive numSeats, min level, max level, customer email as a part of Http POST Operation and will return SeatHOLD object which will have Seat ID and Seat Numbers.

Interface Method

**public** SeatHold findAndHoldSeats(**int** numSeats, Optional<Integer> minLevel, Optional<Integer> maxLevel,String customerEmail);

1. reserveSeats: This service will receive seat hold ID and customer email as a part of Http POST Operation and will return Reservation ID.

Interface Method

String reserveSeats(**int** seatHoldId, String customerEmail);

1. holdLogin: This service will invoke as part of scheduler which will get activated after each 10 milliseconds and will check the status of seat on hold. If hold status is greater than 20 milliseconds then it will remove the hold status.

Interface Method

**public** **void** holdLogin();

# DAO Layer API’s:

Interface for Service layer is ApplicationDao

1. getAvailableSeats: This method will return List of seats available per level. Input parameter for it will be Level ID and which will follow back from service layer to controller api.

Interface Method

List<Seat> getAvailableSeats(Optional<Integer> venueLevel);

1. getSeats: This service will receive min level/max level as a List of long value and will return list of seats as per the value.

Interface Method

**public** List<Seat> getSeats(List<Long> seatIds);

1. getAllSeats: This method will return List of seats.

Interface Method

**public** List<Seat> getAllSeats();

1. reserve: This method will reserve the seat/seats and change the status to HELD/RESERVE.

Interface Method

**public** **long** reserve(Reservation reservation);

1. getRegistration: This method will get reservation based on hold id, email and which is having a status HELD..

Interface Method

**public** Reservation getRegistration(Long id, String customerEmail);

1. getHeldRegistration: This method will return list of reservation which are in hold state for more than or equal to 20 milliseconds.

Interface Method

**public** List<Reservation> getHeldRegistration(Long olddate);

1. removeReservations: This method will remove reservations and release the holded seats.

Interface Method

**public** **void** removeReservations(List<Reservation> reservations) ;

1. saveOrUpdate: This method will persist Seat Object.

Interface Method

**public** **void** saveOrUpdate( Seat entity);

1. saveOrUpdate: This method will persist Venue level object.

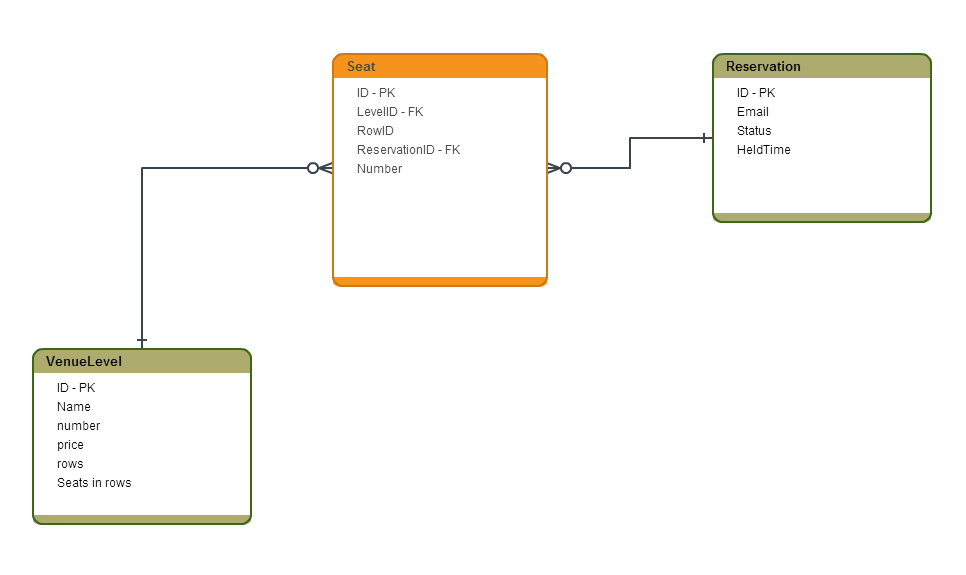
Interface Method

**public** **void** saveOrUpdate( VenueLevel entity);

# Tables:

1. Seat – Store all seat related information and reservation ID.
2. VenueLevel – Store all level related information, level, price, rows, seats.
3. Reservation – Store reservation related information with status.

# Table Structure Diagram:

****

# Entity Classes:

1. Seat: This entity class will maintain seats related details. Like ID(Auto Generated), Level ID, Row ID, Reservation(which is joined column with Reservation ID as a foreign key), Seat Number.

/\*\* The id. \*/

@Id

@GeneratedValue

**private** Long id;

/\*\* The level id. \*/

**private** **long** levelId;

/\*\* The row id. \*/

**private** **long** rowId;

/\*\* The reservation. \*/

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "reservationId")

**private** Reservation reservation;

/\*\* The number. \*/

**private** **int** number;

1. VenueLevel: This entity class will maintain venue level related details. Like ID(Auto Generated), Level Name, Level Number, Price per Level, Rows per level, Seats per row per level.

/\*\* The id. \*/

@Id

@GeneratedValue

**private** Long id;

/\*\* The name. \*/

**private** String name;

/\*\* The number. \*/

**private** **int** number;

/\*\* The price. \*/

**private** Double price;

/\*\* The rows. \*/

**private** Integer rows;

/\*\* The seats in row. \*/

**private** Integer seatsInRow;

1. Reservation: This entity class will maintain reservation related details. Like ID(Auto Generated), email, status(Held, Reserved), heldTime.

/\*\* The id. \*/

@Id

@GeneratedValue

**private** Long id;

/\*\* The email. \*/

**private** String email;

/\*\* The status. \*/

**private** ReservationStatus status;

/\*\* The held time. \*/

**private** Long heldTime;

# Unit Testing:

Unit cases are written using Mockito and covering basic test scenarios are covered

Junit Tests to test

1. End to End flow which also includes DAO flow too.
2. Controller java flow
3. Service layer flow

PostMan script to test

1. GetAvailableSeats API to verify Get operation
2. Hold API to verify Post Operation
3. Reserve API to verify Post Operation.

Web Service API URLS

1. Get Operation- Get Available Seats

URL - <http://localhost:8080/api/getAvailableSeats/1>

1. POST Operation- Find and Hold Request

URL - <http://localhost:8080/api/hold>

Input Request:

{"numberOfSeats":1,"minLevel":1,"maxLevel":2,"email":"sachin@sachin.com"}

1. POST Operation- Reserve ticket Request

URL - <http://localhost:8080/api/reserve>

Input Request:

{"id":2,"email":"sachin@sachin.com"}