**Problem statement**

XYZ wants to build an online movie ticket booking platform that caters to both B2B (theatre partners) and B2C (end customers) clients.

**Note : Navneet’s comments are shaded in yellow. I’d like to showcase basic solution by sharing the screen during discussion.**

Key goals it wants accomplished as part of its solution:

* Enable theatre partners to onboard their theatres over this platform and get access to a bigger customer base while going digital.
* Enable end customers to browse the platform to get access to movies across different cities, languages, and genres, as well as book tickets in advance with a seamless experience.

**Technologies recommended**

* Language -Java and other add-on languages : Java,Spring boot , Spring cloud
* Frameworks- Any
* Database – h2 (in-memory database for demo purpose only.)
* Integration technologies- Microservices
* Cloud technologies- Spring cloud
* Preferred editor to build and present solution : Idea Intellej

*PS:* ***The given exercise will help you get prepared for technical discussion and demonstrate your current understanding on key architectural artifacts. It’s expected that you may not all sections hence can ignore them.***

**Evaluation criteria**

* Code artifacts (**APIs Contract, Design Patterns,** Any one Scenario Implementation)
* Design principles to address functional requirement and non-functional requirement
* DB & Data model
* Platform solutions detailing
* Solution completeness, presentation, and discussion.
* Solution coverage uniqueness and extensibility.

Note: Incomplete solution component would be discussed during discussion round. All sections are not mandatory.

*You can skip solution areas that you are not comfortable by making a note of it.*

**Functional features to implement** Good to have - Code Implementation (Read scenario)):

**Anyone of the following read scenarios:** (Only Service Implementation needed/ No UI required)

* Browse theatres currently running the show (movie selected) in the town, including show timing by a chosen date
* Booking platform offers in selected cities and theatres
  + 50% discount on the third ticket
  + Tickets booked for the afternoon show get a 20% discount

**Candidate Solution:**

* **Problem :** Browse theatres currently running the show (movie selected) in the town, including show timing by a chosen date

@RestController  
  
**public class** TheatreService {  
  
 @Autowired  
 Environment **environment**;  
 @Autowired  
 TheatreRepository **theatreRepository**;  
 @Autowired  
 ScreeningRepository **screeningRepository**;  
 @Autowired  
 MovieRepository **movieRepository**;  
 @RequestMapping(value = **"/bms/{theatreName}/{city}"**, method = RequestMethod.***GET***)  
 **public** String getTheatreByCityAndMovie(@PathVariable String theatreName, @PathVariable String city) {  
  
 *//List<Theatre> listOfTheatre = new ArrayList<>();* Theatre byTheatreNameAndTheatreCity = **theatreRepository**.findByTheatreNameAndTheatreCity(theatreName, city);  
  
 **return "Theatre ID: "** + byTheatreNameAndTheatreCity.getTheatreId() + **" PORT: "** + **environment**.getProperty(**"local.server.port"**);  
 }  
  
 @RequestMapping(value = **"/bms/{theatreName}/{city}/{movieName}/{screeninDate}/{screeningTime}"**, method = RequestMethod.***GET***)  
 **public** List<Screening> getTheatreAndTimingByCityAndMovie(@PathVariable String theatreName, @PathVariable String city, @PathVariable String movieName, @PathVariable String screeninDate, @PathVariable String screeningTime) {  
  
 List<Screening> listOfScreening = **new** ArrayList<>();  
 Theatre byTheatreNameAndTheatreCity = **theatreRepository**.findByTheatreNameAndTheatreCity(theatreName, city);  
  
  
 Screening byMovieNameAndTheatreIdAndScreeningDateAndScreeningTime = **screeningRepository**.findByMovieNameAndTheatreIdAndScreeningDateAndScreeningTime(movieName, byTheatreNameAndTheatreCity.getTheatreId(), Date.*valueOf*(screeninDate), Time.*valueOf*(screeningTime));  
  
 listOfScreening.add(byMovieNameAndTheatreIdAndScreeningDateAndScreeningTime);  
 **return** listOfScreening;  
 }  
  
 @RequestMapping(value = **"/insert/{theatreCity}/{theatreName}"**, method = RequestMethod.***GET***)  
 **public** String insertIntoTheatre(@PathVariable String theatreCity, @PathVariable String theatreName) {  
 Theatre theatre = **new** Theatre(theatreCity, theatreName);  
  
 **theatreRepository**.save(theatre);  
  
 **return "Row inserted successfully with id "** + theatre.getTheatreId();  
 }  
  
  
}

**Panel Feedback:**

**Anyone of the following write scenarios:** Good to have - Code Implementation (write scenario):

* Book movie tickets by selecting a theatre, timing, and preferred seats for the day
* Theatres can create, update, and delete shows for the day.
* Bulk booking and cancellation
* Theatres can allocate seat inventory and update them for the show

**Candidate Solution:**

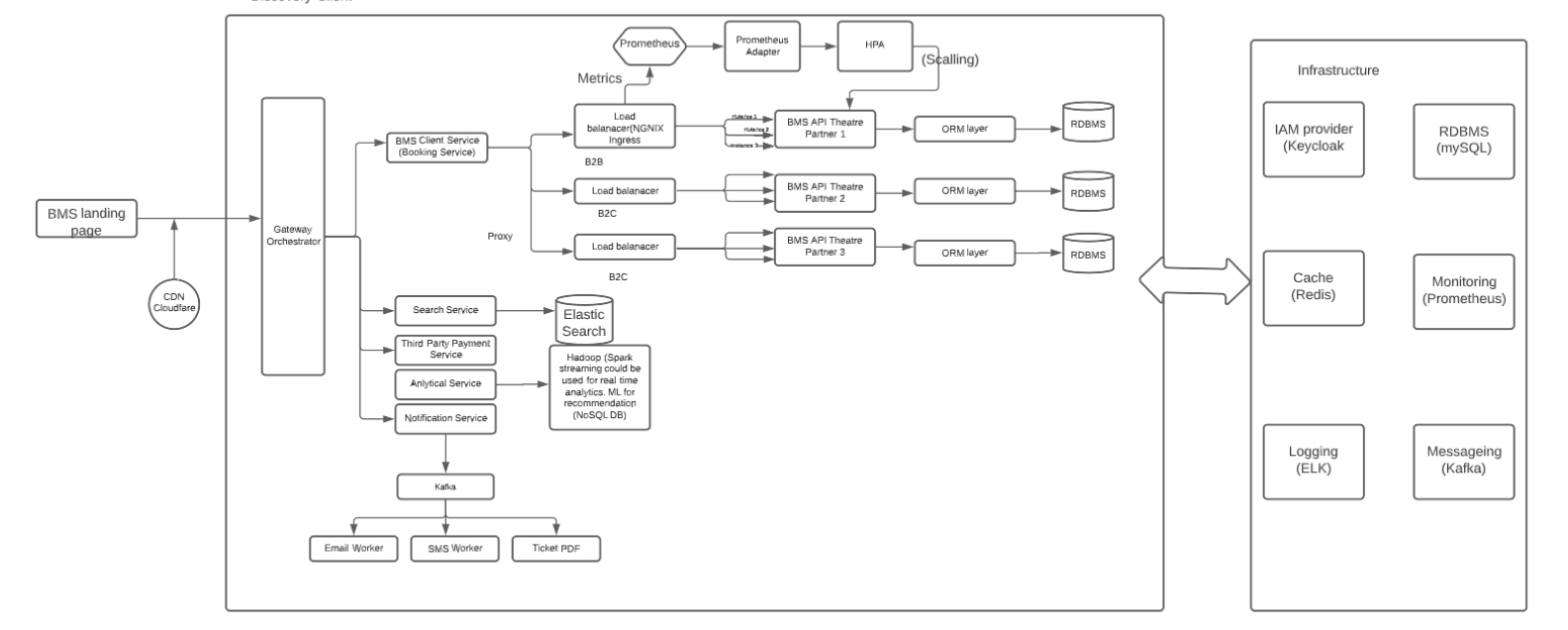
* **Probolem :** Theatres can create, update, and delete shows for the day.

@RestController  
**public class** ScreeningCRUDService {  
 Logger **logger** = LoggerFactory.*getLogger*(ScreeningCRUDService.**class**);  
 @Autowired  
 ScreeningRepository **screeningRepository**;  
  
 @Autowired  
 TheatreRepository **theatreRepository**;  
  
 @Autowired  
 ScreenRepository **screenRepository**;  
  
 @RequestMapping(value = **"/delete/{screeningDate}"**, method = RequestMethod.***GET***)  
 **public** String deleteShowForTheDay(@PathVariable Date screeningDate) {  
  
 List<Screening> byScreeningDate = **screeningRepository**.findByScreeningDate(screeningDate);  
 System.***out***.println(**"==="**+byScreeningDate.get(0));  
 **screeningRepository**.deleteAll(byScreeningDate);  
 **logger**.info(**"All shows are deleted for the day:: "**+screeningDate);  
 **return "All shows are deleted for the day:: "**+screeningDate;  
 }  
  
 @RequestMapping(value = **"/create/{theatreName}/{city}/{screeningDate}"**, method = RequestMethod.***GET***)  
 **public** String createShowForTheDay(@PathVariable String theatreName, @PathVariable String city,@PathVariable Date screeningDate) {  
  
 List<Screen> listOfScreens = **null**;  
 List<Theatre> listOfTheatre = **new** ArrayList<>();  
 Theatre byTheatreNameAndTheatreCity = **theatreRepository**.findByTheatreNameAndTheatreCity(theatreName, city);  
 **if**(0L!= byTheatreNameAndTheatreCity.getTheatreId()) {  
 listOfScreens = **screenRepository**.findByTheatreId(byTheatreNameAndTheatreCity.getTheatreId());  
 }  
 **for** (Screen screen:listOfScreens){  
 Screening screening = **new** Screening();  
 screening.setBookedTickets(0);  
 screening.setScreeningTime(java.sql.Time.*valueOf*(**"10:00:00"**));  
 screening.setMovieName(**"Force"**);  
 screening.setScreeningDate(screeningDate);  
 screening.setTheatreId(byTheatreNameAndTheatreCity.getTheatreId());  
 screening.setScreenId(screen.getScreenId());  
 **screeningRepository**.save(screening);  
 }  
 **logger**.info(**"show is inserted for the day:: "**+screeningDate);  
 **return "show is inserted for the day:: "**+screeningDate;  
 }  
  
 @RequestMapping(value = **"/update/{screeningDate}/{time}"**, method = RequestMethod.***GET***)  
 **public** String updateShowForTheDay(@PathVariable Date screeningDate, @PathVariable Time time) {  
  
 List<Screening> byScreeningDate = **screeningRepository**.findByScreeningDate(screeningDate);  
 **for** (Screening screening : byScreeningDate){  
 screening.setScreeningTime(time);  
 **screeningRepository**.save(screening);  
 }  
  
  
 **logger**.info(**"show is inserted for the day:: "**+screeningDate);  
 **return "show has been :: "**+screeningDate;  
 }  
}

**Panel Feedback:**

**Discussion topics & Logical View:**

**Non-functional requirements-(**Mandatory **-**Design/Arch solution & Optional Implementation**):**



* Describe transactional scenarios and design decisions to address the same.
* Integrate with theatres having existing IT system and new theatres and localization(movies)
* How will you scale to multiple cities, countries and guarantee platform availability of 99.99%?
* Integration with payment gateways
* How do you monetize platform?
* How to protect against OWASP top 10 threats.

**Candidate Solution:**

**Panel Feedback:**

**Discussion topics & Logical View:**

**Platform provisioning, sizing & Release requirements:** (Mandatory-Architecture artifacts)

**Suggested Technologies:**

UI: ReactJS

Server: [Java](https://java.com/en/), [Spring Boot](https://projects.spring.io/spring-boot/), Spring Cloud, [Swagger](http://swagger.io/), [Hibernate](http://hibernate.org/)

Security: [Spring Security](https://projects.spring.io/spring-security/), Keycloak for IAM and Roles.

Database: [MySQL](https://www.mysql.com/)

Server: [Tomcat](http://tomcat.apache.org/)

Caching: In memory cache Redis

Notifications: [Kafka](https://www.rabbitmq.com/). A Distributed message queue for push notifications.

Payment API: Popular ones are [Paypal](https://developer.paypal.com/" \t "_blank) or  [Stripe](https://stripe.com/docs)

Deployment: [Docker](https://www.docker.com/)

Code repository: [Git](https://git-scm.com/)

Logging: [Log4J](https://logging.apache.org/log4j/2.x/)

Log Management: [Logstash, ELK Stack](https://www.elastic.co/products/logstash)

Load balancer: [Nginx](https://www.nginx.com/)

* Discuss your technology choices and decisions through key drivers
* Discuss database, transactions, and data modelling.
* Discuss enterprise systems that you may need to manage specific areas.
* Discuss hosting solution and sizing (Cloud / Hybrid/ Multi cloud)- Any
* Discuss release management across cities, languages etc
* Provide details on monitoring solution
* Discuss overall KPIs
* Create a high-level project plan and estimates breakup.

**Candidate Solution:**

**Panel Feedback:**

**Discussion topics & Logical View:**

**Product management and Stakeholder management**

* Please talk about stakeholder management instances
  + What decisions and actions were taken for decision closure?
* Overall technology management
* Enabling team and introducing efficiencies
* Delivery planning and estimates

**Candidate Solution:**

**Panel Feedback:**

Disclaimer:

*This document is meant to assess your technical skills and is classified as "Sapient confidential". This document by any means shall not be used/shared without permission from Sapient, non-adherence to this can get your candidature blocked for employment with Sapient.*