영화 예매 서비스 시스템

프로젝트 소개

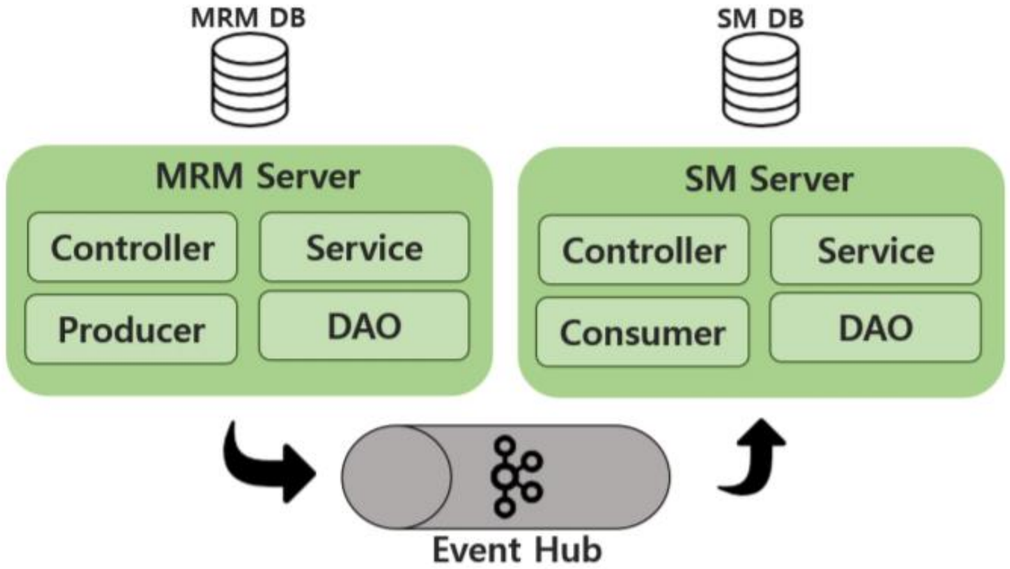
영화 예매 서비스 프로젝트

임의의 클라이언트 >> 임의의 클라이언트는 현재 시간을 기준으로 영화를 지속적으로 예매한다.

영화 예약 관리 서버
(MRM Server) >> 예매 가능 여부를 판단하고 가능하면 이벤트를 이벤트 허브에 적재한다.

좌석 관리 서버
(SM Server) >> 이벤트 허브로부터 이벤트를 읽어와 좌석 현황을 유지한다

Architecture 구성



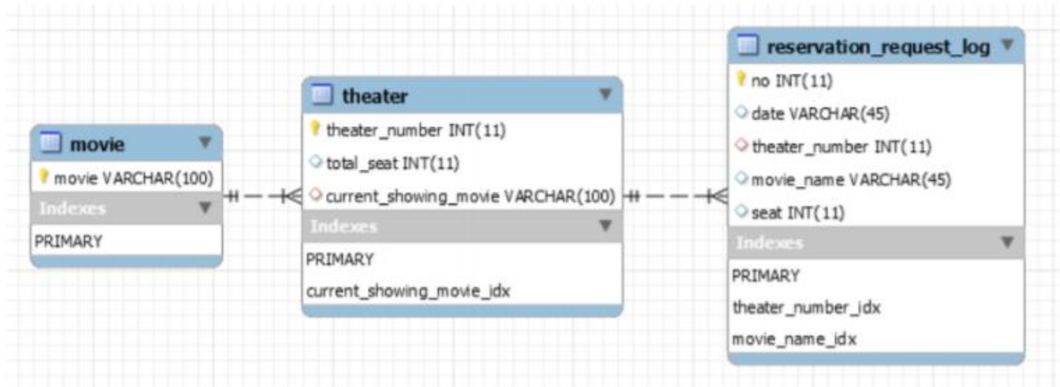
- **Event Hub**
  MRM Server와 SM Server 간의 이벤트를 공유하는 Event Hub로써 Zookeeper 1대, Kafka Broker 3대로 구성되어있다.

- **MRM Server**
  임의의 사용자로 부터 영화 예매 요청을 받는 서버로써 요청시 해당 상영관에 잔여 좌석 이 있는지, 현재 예매하려고 하는 좌석이 이미 예매가 되어있는지를 판단하며 예약이 가능한 경우 Event Hub로 이벤트를 보낸다.

- **SM Server**
  Event Hub로 부터 좌석 예매 이벤트를 리스닝하며, 해당 좌석을 DB에 적재하여 좌석 예 매 정보를 저장한다. 또한 API를 통해서 과거 시점에 상영관의 예약 현황을 확인할 수 있 다.

Schema 구성

MRM Database

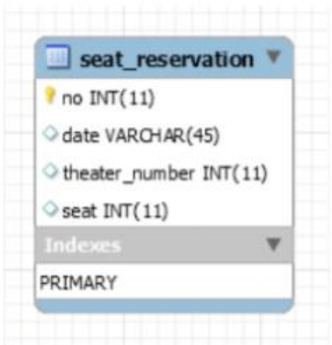


Movie 테이블에 필드는 영화 제목이 저장되는 movie 하나이다. 영화 제목은 중복이 되지 않고 현재 프로젝트에서 영화에 대한 부가적인 정보를 사용하지 않기 때문에 다음과 같이 설계 하였다.

Theater 테이블은 상영관 번호, 각 상영관 별 총 좌석 수, 현재 상영관에서 상영중인 영화와 같이 3개의 필드로 구성이 되어있다. 상영관 번호를 PK로 잡은 이유는 요구사항에서 영화관이 하나이기 때문이다.
만약 영화관도 여러 개라고 했을 경우, PK는 상영관 번호가 아닌 중복 없는 상영관 코드로 구성이 되어야 할 것이며 영화관 필드가 추가되어야 한다.(영화관 테이블 도 추가 구성 필요)

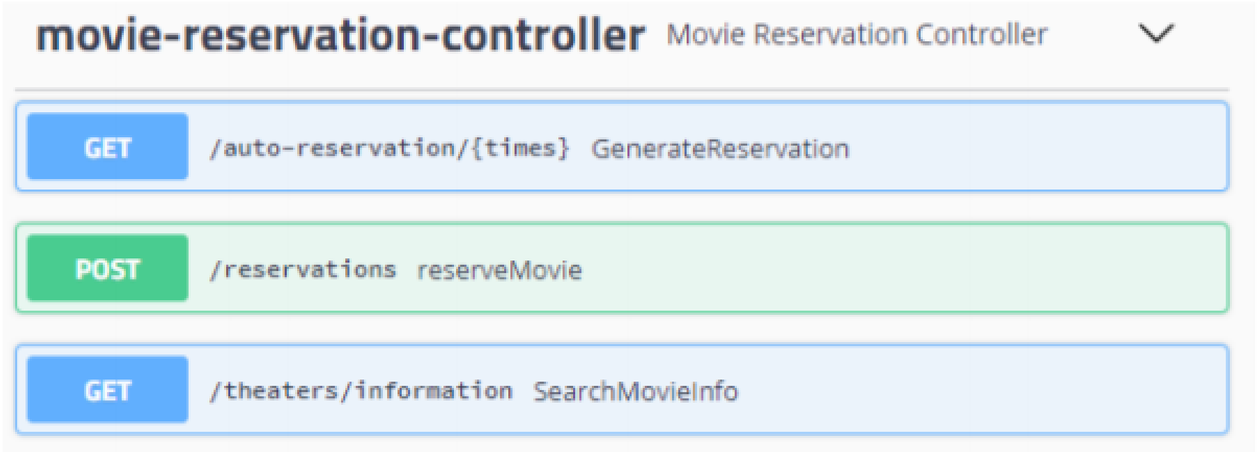
Reservation_request_log 테이블은 순차적으로 증가하는 no 필드가 PK이며 영화 예매 요청을 적재하는 테이블이다.

SM Database

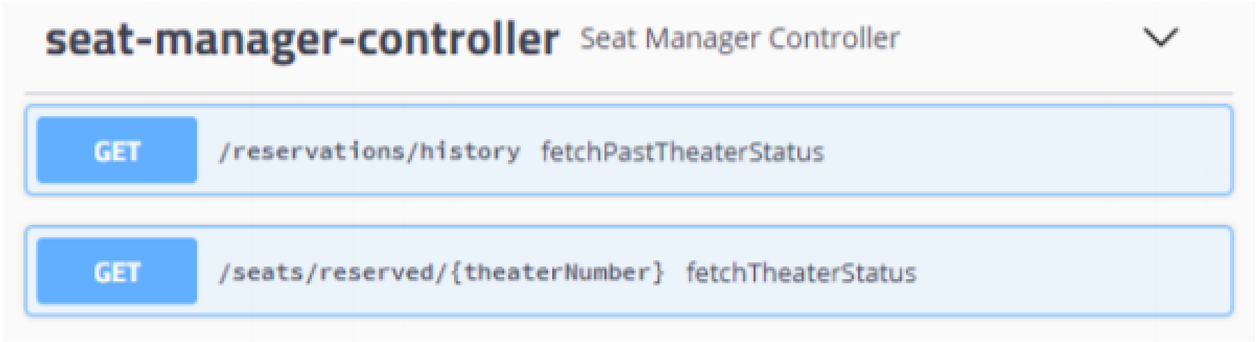


Seat_reservation 테이블은 순차적으로 증가하는 no 필드가 PK이며 상영관의 좌석 예매 정보를 저장하고있는 테이블이다. 좌석에 관한 부가적인 정보가 있었으면 좌석 테이블이 추가 될 수 있지만 현재는 위와 같이 구성되어있다.

API 구성



MRM Server API



SM Server API

주요 이슈 및 해결 - Infinite recursion (StackOverflowError)

JPA 엔티티가 양방향 참조를 하는 경우 'Lombok @ToString'을 사용할 때,

한 엔티티가 toString을 하기 위해 다른 엔티티를 참조하고 또 그 엔티티가 다시 엔티티를 참조하여 무한 루프가 일어 나서 생기는 에러이다.

해당 에러는 다음과 같이 해결할 수 있다.

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString(exclude = {"reservationList"})
@Table(name = "theater")
public class TheaterEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "theater_number")
    private int theaterNumber;

    @OneToMany(mappedBy = "theaterInfo", fetch = FetchType.EAGER)
    private List<SeatReservation> reservationList;

    private int totalSeat;

    private String currentShowingMovie;
}
```

주요 이슈 및 해결 - Infinite recursion (StackOverflowError) cont'd

그러나 이 문제는 **Controller에서 이 객체를 ObjectMapper가 Json으로 변환을 할 때** 도 같은 오류가 발생한다.

해당 문제를 해결하려면 참조가 되는 객체의 앞 단과 뒷 단을 두 개의 어노테이션을 통해 정의를 해주어야 한다.

사용한 어노테이션은 다음과 같다

- JsonManagedReference : 참조가 되는 앞부분을 의미하며 정상적으로 직렬화를 수행을 한다.
- JsonBackReference : 참조가 되는 뒷부분을 의미하고 직렬화를 수행하지 않는다.

```
/* 해당하지 않는 부분은 생략하였다.*/
public class TheaterEntity {

    @JsonManagedReference
    @OneToMany(mappedBy = "theaterInfo", fetch = FetchType.EAGER)
    private List<SeatReservation> reservationList;
}

public class SeatReservation {

    @JsonBackReference
    @ManyToOne(targetEntity = TheaterEntity.class)
    @JoinColumn(name = "theater_number", insertable = false, updatable = false)
    private TheaterEntity theaterInfo;
}
```

주요 이슈 및 해결

- Spring Kafka Consumer JsonDeserialize - addTrustedPackages

해당 에러의 이유는 Producer가 Kafka Broker로 객체 형태의 메시지를 직렬화 하여 보낼 때

Consumer 쪽에서는 신뢰할 수 없는 패키지(클래스)여서 에러가 발생했다.
해결 방안은 JsonDeserializer객체에 addTrustedPackges 메소드로 신뢰하는 패키지 명을 추가를 해준 뒤

Consumer properties에 추가를 해주면 되었다. 해당 소스 코드는 다음과 같다.

```
@Bean
    public ConsumerFactory<String, ReservationLogEntity> consumerFactory() {
        JsonDeserializer<ReservationLogEntity> deserializer = new JsonDeserialize
r<>(ReservationLogEntity.class);
        deserializer.setRemoveTypeHeaders(false);
        // kr.co.leadsoft packages는 무조건 신뢰!
        deserializer.addTrustedPackages("kr.co.leadsoft");
        deserializer.setUseTypeMapperForKey(true);

        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserialize
r.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, deserializer);
        return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(),
                                                 new ErrorHandlingDeserializer<>
(deserializer));
    }
```