지금까지 우리가 피 튀기며 논의했던 모든 수정사항(구간 예매, 오프-바이-원 수정, 보상 트랜잭션 등)을 하나도 빠짐없이 반영한 최종 기술 명세서(Technical Specification Document)입니다.

# [Project Spec] 대용량 트래픽 처리를 위한 고속열차 예매 시스템 (Project: Life-Line)

## 1. 개요 (Overview)

* **프로젝트 명:** High-Concurrency Train Reservation System (가칭: Rail-X)
* **핵심 목표:**
  1. **대용량 처리:** 수십만 동시 접속(Thundering Herd) 상황에서도 서버 다운 없는 대기열 제어.
  2. **데이터 무결성:** **구간 예매(Segment Booking)** 로직 완벽 구현 및 중복 예약 0건 보장.
  3. **보안:** 매크로/암표 방지 및 비회원 개인정보 보호.
* **벤치마킹:** 코레일(Let's Korail) - *하지만 성능은 그 이상을 지향함.*

## 2. 시스템 아키텍처 (System Architecture)

전통적인 Monolithic의 한계를 넘어서되, 초기 개발 속도를 위해 **Modular Monolith**로 시작하여 **MSA**로 확장이 용이한 구조를 채택합니다.

### 2.1. 트래픽 제어 계층 (Traffic Control Layer)

* **CDN / LB:** 정적 리소스 캐싱 및 L7 로드밸런싱.
* **Waiting Queue (대기열 시스템):**
  + **Tech:** Redis Sorted Set (Score: Timestamp).
  + **Logic:** 접속 시 대기표 발급 $\rightarrow$ 내 순서 도달 시 입장 토큰(Access Token) 발급 $\rightarrow$ 토큰 보유자만 API 호출 허용.

### 2.2. 애플리케이션 계층 (Application Layer)

* **Backend:** Java 21, Spring Boot 3.3.
* **Security:** Spring Security + JWT (Stateless).
* **Async:** Apache Kafka (예매 완료 후 알림톡 발송, 이메일 전송, 통계 데이터 집계).

### 2.3. 데이터 계층 (Data Layer)

* **Main DB:** MySQL 8.0 (Master-Slave Replication).
* **Cache & Lock:** Redis Cluster (세션, 대기열, **좌석 비트맵**, 분산 락).

## 3. 데이터베이스 설계 (Database Design)

수정된 **'구간 예매'** 로직과 \*\*'대용량 데이터'\*\*를 고려한 최종 스키마입니다.

### 3.1. 핵심 테이블 상세

1. **users (통합 사용자)**
   * user\_id (PK), role (MEMBER/NON\_MEMBER), access\_code (비회원용 고정길이 랜덤값 + Index).
2. **schedules (운행 스케줄 - 조회 최적화)**
   * schedule\_id (PK), route\_id, train\_id.
   * departure\_date, departure\_time.
   * **Partitioning:** departure\_date 기준 **월(Month)** 단위 Range Partitioning.
   * **Index:** (departure\_date, route\_id, departure\_time) - 검색 쿼리 최적화.
3. **tickets (티켓 - 동시성 격전지)**
   * ticket\_id (PK), reservation\_id (FK).
   * schedule\_id (FK), seat\_id (FK).
   * start\_station\_idx, end\_station\_idx (**구간 정보**).
   * **Constraint:** 유니크 인덱스 제거 $\rightarrow$ 애플리케이션 레벨 검증 및 일반 인덱스 (schedule\_id, seat\_id) 사용.

## 4. 핵심 비즈니스 로직 (Core Business Logic)

**이 프로젝트의 심장부입니다.** 이곳에서 성능과 정합성이 결정됩니다.

### 4.1. 구간 관리 및 비트마스킹 (Segment Bitmasking)

* **개념:** $N$개의 역이 있다면 구간(Segment)은 $N-1$개.
* **매핑 규칙 (Off-by-One 방지):**
  + 출발역 Index: $S$, 도착역 Index: $E$.
  + **점유해야 할 비트:** $S$ 부터 $E-1$ 까지.
  + *예: 서울(0) $\to$ 대전(1) $\to$ 동대구(2).* 서울$\to$대전 예매 시 \*\*0번 비트만 점유\*\*. (1번 비트는 대전$\to$동대구 구간이므로 건드리지 않음).

### 4.2. Redis Lua Script (Atomic Reservation)

* **목적:** 조회(Read)와 수정(Write) 사이의 틈(Race Condition)을 원천 차단.
* **로직:**
  1. 입력받은 구간($S \sim E-1$)에 해당하는 비트 마스크 생성.
  2. Redis의 현재 좌석 상태(Current) 조회.
  3. Current & Mask == 0 인지 확인 (겹치는 구간 없는지).
  4. 참이면 Current | Mask 연산 후 저장 (점유 성공).
  5. 거짓이면 실패 리턴.

### 4.3. 보상 트랜잭션 (Compensation Logic)

* **시나리오:** Redis 선점 성공 $\rightarrow$ DB Insert 실패 (네트워크/DB부하).
* **대응:**
  + try-catch 블록에서 DB 예외 발생 시 catch 문 진입.
  + 즉시 Redis에 롤백 명령 수행 (선점했던 비트 다시 0으로 해제).
  + 사용자에게는 "일시적 오류" 메시지 전달.
* **Reconciliation (재대조):** 5분 주기 배치가 Redis 비트맵과 DB 티켓 정보를 대조하여 유령 좌석 강제 해제.

## 5. 보안 및 부정 예매 방지 (Security & Anti-Fraud)

### 5.1. 인증/인가

* **Spring Security Filter Chain:** JWT 토큰 검증.
* **비회원:** 이름 + 전화번호 + 비밀번호(4자리) + 발급된 랜덤코드(10자리)로 조회 토큰 발급.

### 5.2. 매크로 방어

* **Rate Limiting:** IP당 API 호출 횟수 제한 (Bucket4j + Redis).
* **CAPTCHA:** 예매 요청 전, 혹은 의심스러운 패턴 감지 시 슬라이드 퍼즐 강제.
* **Honeypot:** 봇이 자동으로 채우는 숨겨진 필드(<input type="hidden">)에 값이 들어오면 즉시 차단.

## 6. 개발 환경 및 기술 스택 (Tech Stack)

|  |  |  |
| --- | --- | --- |
| **구분** | **기술 스택** | **버전/비고** |
| **Language** | Java | 21 (LTS) |
| **Framework** | Spring Boot | 3.3.x |
| **Database** | MySQL | 8.0 (InnoDB) |
| **Cache/NoSQL** | Redis | 7.x (Cluster Mode) |
| **ORM** | Spring Data JPA | + QueryDSL 5.0 |
| **Message Queue** | Apache Kafka | 3.6.x |
| **Build Tool** | Gradle | 8.x |
| **Test** | JUnit 5 | + k6 (부하 테스트) |