7️⃣

호텔 예약 시스템

문제 이해 및 설계 범위 확정

비기능 요구사항

- 높은 수준의 동시성
  - 성수기, 대규모 이벤트 기간에는 일부 인기 호텔의 특정 객실을 예약하려는 고객이 많이 몰릴 수 있음
- 적절한 지연 시간
  - 사용자가 예약을 할 때는 응답 시간이 빠르면 이상적이겠지만 예약 요청 처리에 몇 초 정도 걸리는 것은 괜찮음

개략적 규모 추정

- 총 5000개의 호텔과 100만 개의 객실
- 평균적으로 객실의 70%가 사용 중
- 평균 투숙 기간은 3일
- 일일 예상 예약 건수 = 1백만 * 0.7 / 3 = 233333(올림하여 약 240000)
- 초당 예약 건수 = 240000 / 하루에 10^5초 = ~3
  - 초당 예약 트랜잭션 수는 그다지 높지 않음
- 시스템 내 모든 페이지의 QPS
  1. 호텔/객실 상세 페이지
     - 사용자가 호텔/객실 정보 확인 → 조회
     - QPS = 300
  2. 예약 상세 정보 페이지
     - 사용자가 날짜, 투숙 인원, 결제 방법 등의 상세 정보를 확인 → 조회
     - QPS = 30

3. 객실 예약 페이지
    - 사용자가 예약 버튼을 눌러 객실을 예약 → 트랙잭션
    - QPS = 3

⇒ 대략 10% 사용자만 다음 단계로 진행하기 때문에 ▽ 모양의 QPS 분포가 형성됨

⇒ 최종 예약 TPS(Transaction Per Second) = 3

개략적 설계안 제시 및 동의 구하기

API 설계

호텔 관련 API

API	설명
[GET] /v1/hotels/id	호텔의 상세 정보 반환
[POST] /v1/hotels	신규 호텔 추가 호텔 직원만 사용 가능
[PUT] /v1/hotels/id	호텔 정보 갱신 호텔 직원만 사용 가능
[DELETE] /v1/hotels/id	호텔 정보 삭제 호텔 직원만 사용 가능

객실 관련 API

API	설명
[GET] /v1/hotels/:id/rooms/id	객실 상세 정보 반환
[POST] /v1/hotels/:id/rooms	신규 객실 추가 호텔 직원만 사용 가능
[PUT] /v1/hotels/:id/rooms/id	객실 정보 갱신 호텔 직원만 사용 가능
[DELETE] /v1/hotels/:id/rooms/id	객실 정보 삭제 호텔 직원만 사용 가능

예약 관련 API

API	설명
[GET] /v1/reservations	로그인 사용자의 예약 이력 반환
[POST] /v1/reservations/id	특정 예약의 상세 정보 반환
[PUT] /v1/reservations	신규 예약
[DELETE] /v1/reservations/id	예약 취소

- 신규 예약 접수가 가장 중요한 기능
- 신규 예약 api를 호출할 때 reservationId라는 값을 보내는데 이는 이중 예약을 방지하고 동일한 예약은 단 한 번만 이루어지도록 보증하는 멱등 키임

데이터 모델

호텔 예약 시스템이 지원해야 하는 질의

1. 호텔 상세 정보 확인
2. 지정된 날짜 범위에 사용 가능한 객실 유형 확인
3. 예약 정보 기록
4. 예약 내역 또는 과거 예약 이력 정보 조회

시스템 규모가 크지는 않지만 대규모 이벤트가 있는 경우에는 트래픽이 급증할 수도 있기 때문에 대비해야 함

본 설계안에서는 관계형 데이터베이스를 사용할 것

- 읽기 빈도가 쓰기 연산에 비해 높은 작업 흐름을 잘 지원함
- ACID 속성을 보장함
  - 예약 시스템을 만드는 경우에 중요함 → 이중 처리 이슈
- 데이터를 쉽게 모델링 가능

스키마 설계

hotel
hotel_id (PK)

guest
guest_id (PK)

hotel
name
address
location

guest
first_name
last_name
email

room
room_id (PK)
room_type_id
floor
number
hotel_id
name
is_available

room_type_rate
hotel_id (PK)
date (PK)
rate

reservation
reservation_id (PK)
hotel_id
room_id
start_date
end_date
status
guest_id

- reservation 테이블의 status 필드
  - pending
  - paid
  - refunded
  - canceled

  ⇒ 호텔은 특정 객실이 아닌 객실 유형을 예약

  ⇒ 개선이 필요함

개략적 설계안

마이크로서비스 아키텍처를 사용

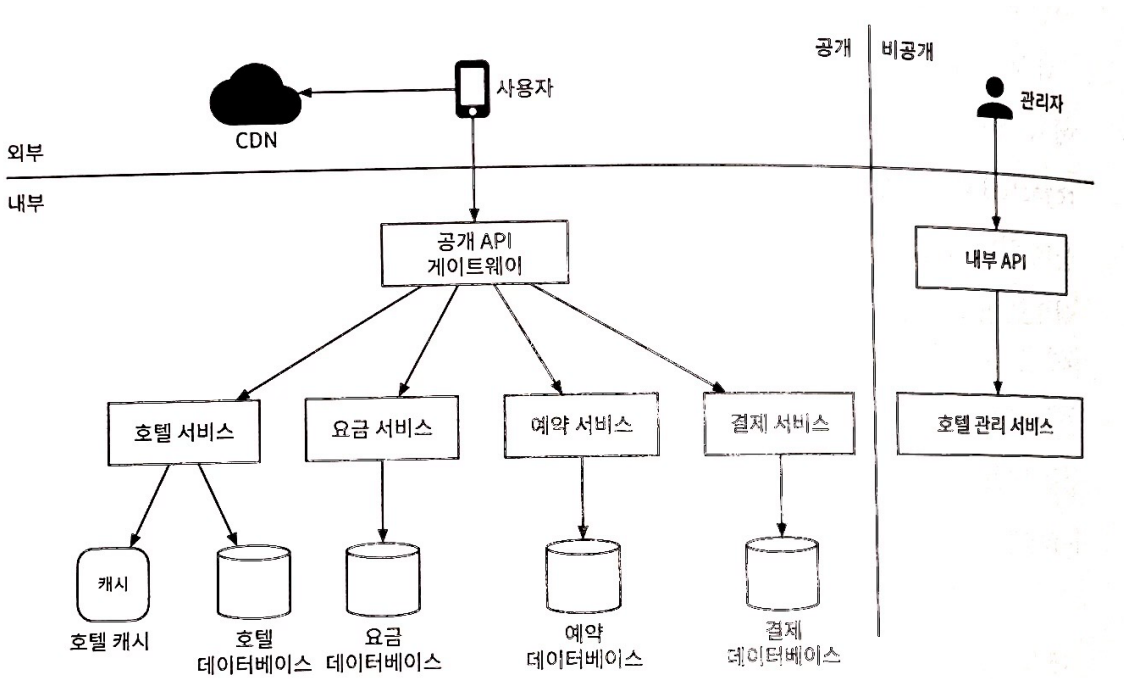

그림 7.4 개략적 설계안

- 사용자
- 관리자(호텔 직원)
- CDN
- 공개 API 게이트웨이
  - 처리율 제한, 인증 등의 기능을 지원하는 완전 관리형 서비스
  - 엔드포인트를 기반으로 특정 서비스에 요청을 전달할 수 있도록 구성
- 내부 API
  - 내부에서만 접속 가능한 사이트
  - VPN 등을 사용해 접속
- 호텔 서비스
  - 호텔과 객실에 대한 정보 제공
  - 캐싱
- 요금 서비스
  - 어떤 요금을 받아야 하는지 데이터 제공
  - 날짜에 따라 요금이 달라짐
- 예약 서비스

    - 예약 요청을 받고 객실을 예약
    - 객실이 예약되거나 최소될 때 잔여 객실 정보를 갱신
- 결제 서비스
    - 고객의 결제를 처리
    - 예약 상태를 업데이트
- 호텔 관리 서비스
    - 관리자만 사용 가능

상세 설계

개선된 데이터 모델

스키마 설계

hotel
hotel_id (PK)
name
address
location

guest
guest_id (PK)
first_name
last_name
email

room
room_id (PK)
room_type_id
floor
number
hotel_id
name
is_available

객실에 관계된 정보를 담는다

room_type_rate
hotel_id (PK)
date (PK)
rate

특정 객실 유형의 특정 일자 요금 정보를 담는다

reservation
reservation_id (PK)
hotel_id
room_type_id
start_date
end_date
status
guest_id

room_id → room_type_id

특정 객실 유형을 저장

투숙객 예약 정보를 담음

room_type_inventory	
hotel_id	호텔 식별자
room_type_id	객실 유형 식별자
date	일자
total_inventory	총 객실 수에서 일시적으로 제외한 객실 수를 뺀 값 일부 객실을 유지보수를 위해 예약 가능 목록에서 빼 둘 수 있어야 함
total_reserved	지정된 hotel_id, room_type_id, date에 예약된 모든 객실의 수

호텔의 모든 객실 유형을 담는 테이블

예약 시스템에 아주 중요한 테이블

동시성 문제

같은 사용자가 예약 버튼을 여러 번 누를 수 있음

- 두 개의 예약이 만들어 질 것
- 클라이언트 측 구현
  - 클라이언트가 요청을 전송하고 난 후 버튼을 비활성화
  - 대부분의 이중 클릭 해결 가능
  - 안정적인 방법은 아님
  - 사용자가 자바스크립트를 비활성화 하면 우회 가능
- 멱등 API
  - 예약 API 요청에 멱등 키를 추가
  1. 예약 주문서 생성
  2. 예약 주문서 반환시 멱등 키(reservation_id) 추가

        - 전역적 유일성을 보증하는 ID 생성기가 만들어 낸 것
    3. 예약 전송시 멱등 키 추가
    4. 예약 재전송시 기본 키의 유일성 조건 위반

여러 사용자가 같은 객실을 동시에 예약하려 할 수 있음

- 트랜잭션 격리 수준이 높은 수준이 아님
- 비관적 락
    - 비관적 동시성 제어 방안
    - 사용자가 레코드를 갱신하려고 하는 순간 즉시 락을 걸어 동시 업데이트를 방지
    - SELECT ... FOR UPDATE
    - 장점
        - 애플리케이션이 변경 중이거나 변경이 끝난 데이터를 갱신하는 일을 막을 수 있음
        - 구현이 쉽고 모든 갱신 연산을 직렬화하여 충돌을 막음
        - 데이터에 대한 경합이 심할 때 유용함
    - 단점
        - 여러 레코드에 락을 걸면 교착 상태 발생 가능
        - 확장성이 낮음
        - 트랜잭션이 너무 오랫동안 락을 해제하지 않고 있으면 다른 트랜잭션은 락이 걸린 자원에 접근할 수 없는데 트랜잭션의 수명이 길거나 많은 엔티티에 관련된 경우, 데이터베이스 성능에 심각한 영향을 끼침
- 낙관적 락
    - 낙관적 동시성 제어
    - 여러 사용자가 동시에 같은 자원을 갱신하려 시도하는 것을 허용
    - 플로우
        1. 데이터베이스 테이블에 version이라는 새 열을 추가
        2. 레코드를 수정하기 전 레코드의 버전 번호를 읽음
        3. 레코드를 갱신할 때 애플리케이션 버전 번호에 1을 더한 다음 데이터베이스에 다시 기록

4. 유효성 검사(현재 버전 번호보다 1만큼 큰 값인지)

- 장점
  - 애플리케이션이 유효하지 않은 데이터를 편집하는 일을 막음
  - 데이터베이스 자원에 락을 걸 필요가 없음
  - 데이터에 대한 경쟁이 치열하지 않은 상황에 적합
- 단점
  - 데이터에 대한 경쟁이 치열한 상황에서는 성능이 좋지 못함

- 데이터베이스 제약 조건
  - 낙관적 락과 유사

```
CONSTRAINT 'check_room_count' CHECK(('total_inventory - to
```

  - 사용자가 객실을 예약하려 하면 total_reserved의 값이 커져 제약 조건을 위반하게 됨
  - 장점
    - 구현이 쉬움
    - 데이터에 대한 경쟁이 심하지 않을 때 잘 동작함
  - 단점
    - 데이터에 대한 경쟁이 심하면 실패하는 연산 수가 엄청나게 늘어날 수 있음
    - 버전을 통제하기 어려움
    - 제약 조건을 허용하지 않는 데이터베이스도 있음

시스템 규모 확장

데이터베이스 샤딩

- 대부분의 질의가 hotel_id를 필터링 조건으로 사용하기 때문에 샤딩 조건으로 쓰면 좋음
- 16개의 샤드로 분산
  - QPS 30000 = 30000/16 = QPS 1875
  - 서버 한 대로 감상 가능

캐시

- 호텔 잔여 객실 데이터는 현재와 미래의 데이터만 중요함(=과거의 어떤 객실을 예약하지 않음)
- 낡은 데이터는 자동적으로 소멸되도록 TTL을 설정하면 좋음
- Redis의 TTL과 LRU 캐시 교체 정책을 사용하면 메모리를 최적으로 활용 가능
- 잔여 객실 캐시
  - 모든 잔여 객실 관리에 필요한 질의는 잔여 객실 캐시로 옮김
  - 사전에 잔여 객실 정보를 캐시에 미리 저장해 두어야 함

캐시가 주는 새로운 과제

- 잔여 객실 데이터에 대한 변화를 데이터베이스에 먼저 반영하기 때문에 캐시에는 최신 데이터가 없을 가능성이 있음
- 잔여 객실이 없는데 캐시 질의 결과에는 여전히 남은 객실이 있다고 나오거나 그 반대 등의 문제가 있을 수 있음

⇒ 데이터베이스가 최종적으로 잔여 객실을 확인하도록 하면 문제가 되지 않음

- 장점
  - 읽기 질의를 캐시가 처리하기 때문에 데이터베이스의 부하가 크게 줄어듦
  - 읽기 질의를 메모리에서 실행하므로 높은 성능을 보장할 수 있음
- 단점
  - 데이터베이스와 캐시 사이의 데이터 일관성을 유지하는 것은 어려움
  - 데이터 불일치가 사용자 경험에 어떤 영향을 끼치게 될지 신중하게 따져보아야 함

마이크로서비스 아키텍처에서의 데이터 일관성 문제에 대한 해결 방안

각 마이크로서비스가 독자적인 데이터베이스를 갖추고 있음

데이터 일관성의 문제 발생

논리적으로는 하나의 원자적 연산이 여러 데이터베이스에 걸쳐 실행되는 일을 피할 수 없음

= 하나의 트랜잭션으로 데이터 일관성을 보증하는 기법을 사용할 수 없음

- 2단계 커밋(2PC)
  - 여러 노드에 걸친 원자적 트랜잭션 실행을 보증하는 데이터베이스 프로토콜
  - 모든 노드가 성공하든 아니면 실패하든 둘 중 하나로 트랜잭션이 마무리되도록 보증
  - 비중단 실행이 가능한 프로토콜이 아니기 때문에 어느 한 노드에 장애가 발생하면 해당 장애가 복구될 때까지 진행 중단
  - 성능이 뛰어난 프로토콜은 아님
- 사가
  - 각 노드에 국지적으로 발생하는 트랜잭션을 하나로 엮은 것
  - 각각의 트랜잭션은 완료되면 다음 트랜잭션을 시작하는 트리거로 쓰일 메시지를 만들어 보냄
  - 한 트랜잭션이라도 실패하면 사가는 그 이전 트랜잭션의 결과를 전부 되돌리는 트랜잭션을 순차적으로 실행
  - 각 단계가 하나의 트랜잭션이라서 결과적 일관성에 의존하는 것

~~본 설계안에서는 동일한 관계형 데이터베이스에 저장하는 실용적인 접근 방식을 선택함~~

마무리

이번 장도 성공적으로 마무리한 여러분, 축하한다!

스스로를 마음껏 격려하도록 하자!