Django 백엔드 스터디 4 회차 문제

2025-08-24

작성자 : 김호중

백엔드 – 논리적사고문제: Django 캐시 설계

트래픽이 많은 소셜 미디어 플랫폼에서 사용자의 피드(Feed)를 불러오는 API 의 응답 속도가 느려졌습니다. DB 쿼리가 병목의 원인으로 지목되어 캐시를 도입하기로 결정했습니다. 그러나 단순히 API 응답 전체를 캐싱했더니, 사용자가 새 글을 작성하거나 '좋아요'를 눌러도 피드에 즉시 반영되지 않는 데이터 정합성 문제가 발생했습니다. 캐시 만료 시간(TTL)을 짧게 설정하자 캐시 히트율(Hit Rate)이 급격히 떨어져 성능 개선 효과가 미미해졌습니다.

문제

캐시의 데이터 정합성과 성능(히트율) 이라는 두 마리 토끼를 어떻게 잡을 수 있을까요? 데이터 변경 시점에 캐시를 능동적으로 무효화하는 전략을 설명하시오. (Cache Invalidation)

이벤트나 특정 시간대에 사용자가 몰려 캐시된 데이터가 동시에 만료될 때 발생하는 '캐시 스탬피드(Cache Stampede)' 현상을 방지할 수 있는 DB 보호 설계를 제시하시오.

어떤 데이터를 캐시하고 어떤 데이터를 캐시하면 안 되는지, 캐시 대상 선정 원칙을 구체적인 기준(휘발성, 중요도 등)과 함께 서술하시오.

캐시 시스템의 운영 관측 가능성(Observability) 을 위해 추적해야 할 핵심 메트릭은 무엇이며, 이를 통해 어떤 문제를 진단할 수 있는지 설명하시오.

백엔드 – 논리적사고문제: Django 실시간서비스

실시간 경매 애플리케이션을 개발 중입니다. 사용자의 입찰(Bid)은 즉시 모든 참여자에게 공유되어야 하며, 최종 낙찰 정보는 절대로 유실되어서는 안 됩니다. 초기에는 Django 의 일반적인 HTTP 요청-응답 모델로 구현했으나, 클라이언트가 입찰 결과를 확인하려면 계속해서 서버에 요청을 보내야 하는 폴링(Polling) 방식으로 인해 불필요한 트래픽과 지연이 발생했습니다.

문제
HTTP 의 한계를 넘어 실시간 양방향 통신을 구현하기 위한 기술(WebSocket, SSE, Long Polling)들의 장단점을 비교하고, 경매 앱에 가장 적합한 기술과 그 선택 근거를 설명하시오.

수천 명 이상의 대규모 동시 접속을 안정적으로 처리하기 위한 서버 아키텍처(e.g., Django Channels)와 상태 관리(Connection State) 방안을 서술하시오.

네트워크 문제로 메시지가 유실되거나 중복 전송되는 경우를 방지하고 '최소 한 번 전송 보장(At-least-once delivery)' 을 구현할 수 있는 아키텍처 패턴을 제시하시오.

서버 재배포나 장애 시에도 연결 중단을 최소화하고, 클라이언트가 자동으로 재연결하여 상태를 복구할 수 있는 전략은 무엇인지 설명하시오.

백엔드 – 논리적사고문제: AIR flow 도입 근거

한 데이터 분석팀이 매일 새벽 수십 개의 파이썬 스크립트를 cron 을 이용해 순차적으로 실행하고 있습니다. 스크립트 간에는 암묵적인 의존성이 존재하며(A 스크립트가 만든 파일을 B 스크립트가 읽는 등), 중간에 하나라도 실패하면 전체 파이프라인이 멈추고 담당자는 수동으로 로그를 찾아 원인을 파악한 뒤 실패 지점부터 다시 실행해야 합니다. 과거 특정 날짜의 데이터를 재처리(Backfill)하는 작업은 상상조차 하기 어렵습니다.

문제
cron 과 비교했을 때, Apache Airflow 를 도입해야 하는 결정적인 기술적, 운영적 이유는 무엇인지 3 가지 이상 설명하시오.

Airflow 의 핵심 개념인 DAG 와 멱등성(Idempotency) 이 어떻게 데이터 파이프라인의 신뢰성과 재현성을 보장하는지 구체적인 예시를 들어 설명하시오.

과거 데이터 재처리(Backfill)나 특정 작업의 재실행이 cron 에서는 어렵지만 Airflow 에서는 간단한 이유를 실행 컨텍스트(Execution Context) 와 관련지어 서술하시오.

Airflow 도입이 항상 정답은 아닐 수 있습니다. Airflow 도입 시 마주할 수 있는 운영 복잡성과 학습 곡선의 문제를 언급하고, 이를 완화할 수 있는 전략을 제시하시오.

백엔드 – 논리적사고문제: 데이터 어노테이션 플랫폼으로의 확장

ML 모델 학습을 위해, 운영팀은 Django Admin 페이지에서 직접 이미지에 '정상' 또는 '불량' 태그를 다는 방식으로 데이터를 수집(어노테이션)하고 있습니다. 초기에는 문제가 없었지만, 데이터가 수십만 건으로 늘어나자 Admin 페이지가 극심하게 느려졌고, 여러 명의 작업자가 동시에 작업할 때 데이터 충돌이 발생하기 시작했습니다. 또한, 어떤 작업자가 어떤 데이터를 언제 수정했는지 추적하기 어려워 데이터의 신뢰성에 의문이 제기되었습니다.

문제

Django Admin 이 대규모 데이터 조회/수정 작업에 부적합한 이유를 ORM 의 동작 방식과 HTTP 요청/응답 모델 관점에서 설명하시오. (e.g., 전체 queryset 로딩 문제, 페이지네이션의 한계)

이 문제를 해결하기 위해, Django 백엔드와 상호작용하는 별도의 라벨링용 프론트엔드(e.g., React, Vue)를 구축한다고 가정합시다. 이 때, 효율적인 데이터 라벨링을 위해 백엔드는 어떤 API 를 제공해야 할까요? (e.g., 작업 할당 API, 데이터 잠금(Locking) API, 통계 API)

데이터의 재현성과 버전 관리를 위해, 라벨링된 데이터를 어떻게 저장하고 관리하는 것이 좋을까요? "라벨"을 원본 데이터 테이블의 컬럼으로 추가하는 대신, 별도의 'Annotation' 모델로 분리하여 (user, data, label, created_at)을 저장하는 방식의 장점을 설명하시오.

궁극적으로, 이러한 내부 시스템을 데이터 버전 관리 도구(e.g., DVC)와 통합하여, 특정 버전의 '코드'가 특정 버전의 '데이터셋'으로 학습되었음을 보장하는 파이프라인을 어떻게 설계할 수 있을지 아이디어를 제시하시오.

백엔드 – 논리적사고문제: ORM 의 한계 분석 쿼리부하 분리를 위한 데이터웨어하우스 구축

고객사의 데이터를 분석하여 리포트를 제공하는 B2B SaaS Django 애플리케이션이 있습니다. 고객이 리포트 페이지에 접속할 때마다 Django ORM 은 복잡한 annotate, aggregate, Subquery 를 사용하여 프로덕션 PostgreSQL DB 에서 실시간으로 데이터를 집계합니다. 서비스가 성장하며 데이터가 수억 건 쌓이자, 이 리포트 쿼리 하나가 DB CPU 점유율을 90%까지 치솟게 만들어 전체 서비스 장애를 유발하는 상황에 이르렀습니다.

문제

OLTP(Online Transaction Processing)용으로 설계된 프로덕션 DB 에 **OLAP(Online Analytical Processing)**성 쿼리를 실행하는 것이 왜 치명적인 안티패턴인지 인덱스 전략, 데이터 스캔 범위, 락(Lock) 경합의 관점에서 설명하시오.

이 문제를 해결하기 위해, Django 의 프로덕션 DB 변경 사항을 CDC(Change Data Capture) 도구(e.g., Debezium)를 이용해 실시간으로 데이터 웨어하우스(DW)(e.g., BigQuery, Snowflake)에 복제하는 아키텍처를 설계하시오. 주기적인 배치(Batch) ETL 방식과 비교하여 CDC 방식이 갖는 장점은 무엇인가요?

분석용 DB 가 분리된 후, Django 백엔드는 리포트 페이지를 어떻게 렌더링해야 할까요? ① Django 가 직접 DW 에 쿼리하기, ② 미리 집계된 결과를 캐시나 별도 테이블에 저장해두기. 두 방식의 **기술적 트레이드오프(지연 시간, 비용, 복잡성)**를 비교하여 설명하시오.

새로운 아키텍처에서, 특정 고객이 "과거 3 년 치 데이터를 다시 집계해달라"고 요청하는 경우, 프로덕션 DB 에 영향을 주지 않고 안전하게 **데이터를 재처리(Backfill)**하는 절차를 구체적으로 제시하시오.

백엔드 – 논리적사고문제: 대용량 데이터 처리 파이프라인 설계

매일 1 억 건의 사용자 행동 로그(CSV 파일)를 분석하여 비즈니스 지표를 추출하는 파이프라인이 있습니다. 주니어 시절에는 Pandas 스크립트를 작성하여 100 만 건 샘플 데이터로 멋지게 처리했지만, 실제 데이터에 적용하자 메모리 초과(Out of Memory)로 실패하거나 하루가 지나도 작업이 끝나지 않습니다.

문제
단일 머신의 메모리 한계를 극복하기 위한 분산 처리 프레임워크(e.g., Spark, Dask)의 필요성은 무엇이며, 이들이 데이터를 처리하는 기본 원리(e.g., 파티셔닝, 지연 연산)는 어떻게 다른지 설명하시오.

단순 CSV 가 아닌 컬럼 기반 파일 포맷(e.g., Parquet, ORC)을 사용해야 하는 이유를 '스키마 진화(Schema Evolution)'와 '열 선택(Projection Pushdown)' 관점에서 설명하시오.

데이터 파이프라인의 신뢰성을 보장하기 위한 '데이터 품질(Data Quality)' 검증 계층을 어떻게 설계할 수 있을까요? (e.g., Great Expectations 같은 도구를 사용한 자동화된 데이터 테스트)

동일한 작업을 재실행해도 항상 같은 결과를 보장하는 멱등성(Idempotent) 있는 데이터 파이프라인을 설계하는 원칙은 무엇인지 설명하시오. (e.g., INSERT OVERWRITE 패턴)