Django 백엔드 스터디 2 회차 문제

2025-08-05

작성자 : 김호중

컴퓨터공학 – 서술형 (문제당 3 점)

1. REST 아키텍처 스타일의 핵심 제약 조건(Uniform Interface 등)을 나열하고 각 제약이 주는 이점을 서술하시오. (중요)

2. B-tree 인덱스 내부 동작 원리와, 선택도(selectivity)가 낮을 때 인덱스가 오히려 성능을 저하시킬 수 있는 이유를 설명하시오. (중요)

3. 로드밸런싱 알고리즘 Round-Robin, Least-Connections, IP Hash 세 알고리즘의 특징과 사용 시 고려해야 할 점을 비교하시오.

4. Docker 컨테이너가 VM 보다 경량화되는 원리를 네임스페이스·cgroups 관점에서 설명하고, 보안 격리 측면의 차이를 논하시오. (중요)

5. TCP 3-way/4-way 핸드셰이크 과정을 설명하고, 대기 연결이 많은 서버에서 SYN flood 를 완화하는 방법을 제시하시오.

6. 페이징 vs 세그멘테이션에 대해서 설명하시오

7. 프로세스 vs 스레드 두 개념의 차이와 주소 공간/스택/파일 디스크립터 등 어떤 자원을 공유하는지 설명하시오. (중요)

컴퓨터공학 – 객관식, 단답형, o/x (문제당 2 점)

1. Docker 컨테이너는 호스트 OS 커널을 공유한다. (O/X)

2. Docker 이미지는 변경 불가능(immutable)하다. (O/X)

3. OS 커널 모드(Kernel Mode)는 사용자 모드(User Mode)보다 높은 권한을 가진다. (O/X)

4. NAT 게이트웨이(NAT Gateway)는 사설 IP 를 공인 IP 로 변환해 주는 역할만 수행하며, 트래픽 필터링 기능은 없다. (O/X)

5. 마이크로서비스 아키텍처에서 API 게이트웨이(API Gateway)는 클라이언트 요청을 각 서비스로 라우팅할 뿐만 아니라, 인증·인가, 로깅, 속도 제한(rate limiting) 등의 기능도 제공한다. (O/X)

6. TCP 3-way Handshake 순서 중 첫 번째 메시지는?

A. SYN B. SYN-ACK

C. ACK D. FIN

7. B-Tree 인덱스의 특징으로 옳지 않은 것은?

A. 균형 트리 구조로 검색 시간이 로그(log) 스케일이다.

B. 중복된 값이 많은 컬럼에 부적합하다.

C. 대용량 범위 검색(range scan)에 효과적이다.

D. 해시 연산을 기반으로 동작한다.

8. Docker 컨테이너와 가상 머신의 가장 큰 차이점은?

A. 컨테이너는 하이퍼바이저를 사용한다.

B. 컨테이너는 별도의 커널을 포함하지 않는다.

C. 가상 머신은 호스트 커널을 공유한다.

D. 가상 머신은 이미지 레이어를 사용하지 않는다.

9. 4×4 미로에서 '1'은 이동 가능, '0'은 벽. (0,0)에서 (3,3)까지 최단 칸 수를 구하시오.

---

1 0 1 1

1 1 1 0

0 1 0 1

1 1 1 1

A. 7 B. 8

C. 9 D. 10

10. TCP 의 흐름 제어(flow control) 메커니즘은?

A. 슬라이딩 윈도우(Sliding Window)

B. 혼잡 제어(Congestion Control)

C. 다중 경로 전송(Multipath Transmission)

D. 체크섬(Checksum)

Django - 서술형 (문제당 3 점)

1. 뷰 함수에서 request.GET 과 request.POST 딕셔너리 객체가 각각 언제 사용되는지, 예시 상황을 들어 설명하시오. (중)

2. AbstractBase vs multi-table inheritance vs proxy model 세 가지 모델 상속 방식의 차이·장단점을 설명하고, 실제 서비스에서 적합한 사용 사례를 각각 제시하시오 (중상)

3. DRF 에서 Model → Serializer → ViewSet → Router 로 이어지는 데이터 흐름을 설명하고, ListCreateAPIView 와 ModelViewSet 중 하나를 선택해 간단한 예시 코드와 함께 장단점을 논하시오. (중상)

4. 커스텀 미들웨어를 작성해 Content-Security-Policy 헤더를 삽입하는 과정을 설명하고, 미들웨어 체인에서의 위치 선정이 보안·성능에 미치는 영향을 논하시오. (중상)

5. 다음 코드를 보고 어떤 상속인지 추론하고 그 근거를 작성하시오.(추상, 멀티테이블, 프록시)

(1)class TimeStamped(models.Model):

```
    created_at = models.DateTimeField(auto_now_add=True)
    class Meta:
        abstract = True

class Article(TimeStamped):
    title = models.CharField(max_length=200)
```

---

(2)

```
class Content(models.Model):
    title = models.CharField(max_length=200)


class Video(Content):
    duration = models.PositiveIntegerField()
```

---

(3)

```
class Order(models.Model):
    status = models.CharField(max_length=20)


class PaidOrder(Order):
    class Meta:
        proxy = True
        ordering = ['-id']
```

백엔드 – 논리적사고문제: 오픈뱅킹 계좌이체 이중지급 방지 설계 -toss 참고

우리 서비스가 오픈뱅킹 API 로 고객의 돈을 타행 계좌로 이체한다.

모바일 네트워크 재전송, 사용자 연타, 은행 웹훅 중복/역순 때문에 같은 이체가 두 번 나가는 사고가 가끔 발생한다.

목표로 삼은 것은 다음과 같다.

1. 한 번의 이체 의도 = 실제 출금 1 회(이중지급 0)

2. POST 재전송, 워커 재시작, 웹훅 중복/역순에서도 안전

3. 다운타임 없이 운영 가능(롤링 배포)

4. 사후 감사/정산(대사)이 가능해야 함

위와 같은 상황을 감안하여 다음 문제의도를 파악하고 해결하시오.

1. 이체 fsm 을 설계하시오. (조건부 업데이트를 중심으로 서술)

2. 처리 흐름에 대해 자세히 설명하시오.

3. 다음 상황을 가정하였을 때 해결방안을 생각하고 서술하시오.

상황 1: 우리가 요청을 보냈고, 은행이 성공 처리했지만 네트워크 끊김 → 우리는 결과를 못 받음.

상황 2: 은행이 응답을 못 줬지만 웹훅이 뒤늦게 옴.

필요 지식

상태 정의

PENDING: 요청만 접수. 장부에 보류 전표 기록(돈은 잠시 홀드).

DISPATCHED: 은행 API 호출 완료(응답 대기). external_ref 확정.

SETTLED: 성공 확정(보류 → 실제 출금/입금 전표로 전환). 종단

FAILED: 거절/오류 확정(보류 롤백 전표). 종단

EXPIRED: 일정 시간 내 확정 못함(타임아웃 대기). 종단 아님(늦은 웹훅 수용)

오픈뱅킹 계좌이체에서 이중지급을 막으려면, “같은 의도는 몇 번 실행돼도 결과가 한 번 한 것과 같아야 한다”는 멱등성 원칙을 서비스 전 구간에 깔아야 합니다. 사고가 나는 지점은 보통 세 군데입니다. 첫째, 사용자가 버튼을 연달아 누르거나 휴대폰 네트워크가 흔들리며 같은 요청이 여러 번 들어오는 입구 구간, 둘째, 우리 서버가 은행 API 를 호출하는 바깥 구간(여기는 최소 한 번 전달이 기본이라 중복이 쉽게 생깁니다), 셋째, 은행이 성공·실패를 알려주는 웹훅이 중복되거나 순서가 뒤집혀 돌아오는 되돌아오는 길입니다. 이 세 구간 어디서든 중복이 나도 최종 결과가 흔들리지 않게 하려면, “항상 같은 것을 같은 것으로 인식할 수 있는 키”와 “뒤늦게 온 신호는 무해하게 처리하는 상태 전이 규칙”, 그리고 “돈 흐름을 재현할 수 있는 장부”가 필요합니다.

입구에서는 클라이언트가 생성해 재사용하는 client_request_id 가 핵심입니다. 이 값은 같은 의도를 가리키는 표식이므로 데이터베이스에 유일 제약(UNIQUE)으로 두어야 합니다. 그러면 동일한 요청이 동시에 여러 번 도착해도 실제로는 한 행만 삽입되고 나머지는 “이미 있는 것”으로 처리됩니다. 중요한 점은 이 값을 서버가 임의로 만들지 않는 것입니다. 서버가 매번 새 값을 만들면 재시도 때 값이 달라져 멱등성이 깨집니다. 클라이언트가 만든 값을 끝까지 재사용해야 입구에서 중복이 차단됩니다.

은행 호출 구간에서는 external_ref 가 중추 역할을 합니다. 어떤 은행은 멱등키를 지원하지만, 어떤 곳은 그렇지 않습니다. 지원 여부와 무관하게 우리가 같은 이체 의도에 대해 언제나 같은 참조값을 생성해 보내면, 은행은 중복 요청을 한 건으로 인식하거나 “이미 처리됨”으로 응답하게 됩니다. 이 참조값은 보낸 계좌, 받는 계좌, 금액, 통화, 가치일자 등 ‘불변’ 필드를 정규화해 연결한 다음 안전한 해시로 만든 결정적 값이어야 하고, 우리 쪽 DB 에도 UNIQUE 로 걸어 외부·내부 양쪽에서 중복을 동시에 막습니다. 여기서 실수는 시각·난수 같은 가변 요소를 섞는 것입니다. 그렇게 하면 재시도마다 값이 바뀌어 같은 의도를 다른 것으로 오인하게 됩니다.

웹훅 구간에서는 event_id 가 멱등의 기준입니다. 은행이 주는 이벤트 ID 를 기본키로 먼저 저장해 두면 같은 알림이 두세 번 와도 두 번째부터는 데이터베이스가 자동으로 거부합니다. 만약 은행이 이벤트 ID 를 주지 않으면, 웹훅 본문의 핵심 필드들을 정규화해 해시한 값을 우리 쪽 event_id 로 삼아 같은 효과를 냅니다. 중요한 순서는 “웹훅을 먼저 저장하고, 그다음 상태를 바꾼다”입니다. 반대로 하면 중복이나 역순 알림에 취약해집니다.

상태 관리의 요령은 “앞으로만 간다”와 “조건부로만 바꾼다”입니다. 이체의 상태를 PENDING(요청 접수, 장부 보류만 기록) → DISPATCHED(은행 호출 완료, 응답 대기) → SETTLED(성공 확정) 또는 FAILED(실패 확정)로 정의하고, 타임아웃 대기를 위해 EXPIRED 를 보조 상태로 둡니다. 그런 다음 상태 변경을 “현재가 X 일 때만 Y 로 바꿔라”라는 조건부 규칙으로 제한합니다. 예컨대 성공 웹훅이 오면 “지금이 DISPATCHED 이거나 EXPIRED 일 때만 SETTLED 로” 바꾸고, 이미 SETTLED 나 FAILED 라면 아무 것도 하지 않습니다. 이렇게 하면 성공 후 뒤늦게 실패 웹훅이 와도 종단 상태를 덮어쓰지 못하고, 성공·실패가 거의 동시에 들어와도 두 쪽 모두 조건을 확인하다가 한 번만 실제 변경이 일어납니다. EXPIRED 는 타임아웃일 뿐 종단이 아니므로, 뒤늦은 성공·실패가 오면 최종 확정으로 전이시킵니다. 핵심은 “시간 순서”에 의존하지 말고 “현재 상태 값”만으로 판단하는 것입니다. 웹훅은 순서를 보장하지 않기 때문입니다.

돈을 안전하게 다루려면 장부도 올바르게 써야 합니다. 성공 시에만 돈을 옮기면 깔끔할 것 같지만, 현실에서는 요청 직후부터 시스템 곳곳에서 상태를 보여줘야 하고 실패 시에도 깨끗하게 복구되어야 합니다. 그래서 먼저 보류 전표를 두 줄로 기록합니다. 출금 계정에 보류 증가, 내부 보류 계정에 대응하는 감소를 기록해 전체 합이 0 이 되도록 하는 ‘복식부기’ 방식입니다. 성공이 확정되면 보류 전표를 실제 출금·입금 전표로 전환하고, 실패가 확정되면 보류를 원위치시키는 롤백 전표를 남깁니다. 이렇게 하면 언제든지 과거 특정 시점의 금전 흐름을 재현할 수 있고, 회계·감사·정산(대사)에 필요한 근거가 됩니다.

요청을 DB 에 기록하고 외부를 호출하는 순서에서 “DB 에는 들어갔는데 외부 호출이 유실됐다” 같은 어긋남을 없애려면 트랜잭셔널 아웃박스 패턴을 씁니다. 이체 생성 트랜잭션 안에서 아웃박스 테이블에 ‘이체를 디스패치하라’는 레코드를 같이 저장하고, 커밋 이후 별도의 워커가 이 레코드를 읽어 은행 API 를 호출합니다. 이렇게 하면 DB 기록과 외부 호출 사이가 트랜잭션 경계로 묶여 일관성이 생기고, 프로세스가 중간에 죽어도 워커가 다시 읽어 안전하게 재시도합니다. 이때도 동일한 external_ref 로 호출하므로 재시도는 자연스럽게 멱등하게 수렴합니다.

읽기/쓰기 분리 환경에서는 POST 직후 GET 이 방금 쓴 데이터를 못 보는 일이 흔합니다. 이건 이중지급과 직접 관련은 없지만 사용자 경험과 운영 판단을 흐릴 수 있어, 생성 직후 상세

조회는 일시적으로 프라이머리에서 읽거나, 화면에 “처리 중” 배지를 띄우는 방식으로 착시를 줄입니다. “왜 방금 송금이 안 보이죠?” 같은 문의를 줄이고, 불필요한 재시도를 막는 효과가 있습니다.

분산락을 쓰지 않는 것도 중요한 선택입니다. 락은 멋져 보이지만, 락 분실·만료·네트워크 분할 등 새로운 실패 지점을 도입합니다. 반대로 데이터베이스의 유일 제약과 결정적 참조값, 그리고 조건부 상태 전이만으로도 대부분의 중복·경쟁·역순 문제를 더 단순하고 견고하게 막을 수 있습니다. “입구 멱등(요청 ID)”, “외부 멱등(대외 참조)”, “이벤트 멱등(웹훅 ID)”—이 세 축이 겹겹이 방어막을 만들고, 상태 전이 규칙이 뒤늦게 도착한 신호를 무해하게 흡수합니다.

운영 단계에서는 재시도·타임아웃의 기본값을 현실적으로 잡는 게 필요합니다. 은행 API 의 p90 응답시간에 소폭 여유를 더한 수준으로 타임아웃을 두고, 지수 백오프와 지터를 섞어 2~3 회만 재시도합니다. 총 대기시간은 사용자 경험을 해치지 않는 선으로 제한합니다. 타임아웃이 누적되면 DISPATCHED 상태가 길게 머물 텐데, 임계 시간을 넘기면 EXPIRED 로 전환해 사용자에게 ‘보류 중’임을 명확히 알려주고 뒤늦은 확정을 수용합니다. 그리고 매일 대사를 돌려 은행 거래목록과 우리 장부를 external_ref 기준으로 맞춰봅니다. 불일치가 0 이 아니라면 즉시 원인을 찾고, 실제로 은행 측 중복 출금이 발생했다면 장부와 프로세스로 차액 회수를 진행합니다. 모니터링 지표로는 유일 제약 충돌로 막은 중복 건수, 대사 불일치 건수, 이체 승인 p95, DISPATCHED 체류 시간 분포 등을 보는 것이 현실적입니다.

요약하면, 이중지급 방지는 복잡한 트릭이 아니라 기본의 철저함입니다. 같은 의도를 같은 것으로 식별하는 키를 세 구간에 정확히 심고, 상태는 앞으로만 조건부로 움직이게 만들며, 돈의 움직임은 보류→확정/롤백으로 두 줄씩 남깁니다. DB 커밋과 외부 호출은 아웃박스로 엮고, 읽기 일관성과 재시도·타임아웃은 사용자 경험과 운영 편의를 해치지 않는 범위로 다룹니다. 이렇게 설계하면 중복 호출·네트워크 오류·웹훅 역순이 뒤엉켜도 최종 결과는 “한 번 보낸 것과 같은” 단단함으로 수렴합니다.

백엔드 – 논리적사고문제: 외부 API 집계 게이트웨이 설계

(예: FX/Weather/News 3 종 상류 API 집계)

우리 서비스는 2~3 개의 외부 API(상류)를 병렬 호출해 단일 응답으로 제공하는 게이트웨이(/v1/summary)를 운영한다.

클라이언트 재시도, 모바일 네트워크 재전송, 상류의 레이트리밋(429), 지연/타임아웃, ETag 기반 304 Not Modified, 캐시 만료·무효화 타이밍 차이로 인해 부분 실패가 빈번하다.

목표는 다음과 같다.

한 번의 집계 의도 = 동일 파라미터 요청에 대해 일관된 단일 응답(멱등, 불필요한 상류 재호출 최소화)

타임아웃, 재시도, 회로차단을 통한 안정성 확보 + 부분 실패·폴백 시에도 유의미한 응답 제공

캐시 키 설계와 무효화/재검증(ETag/TTL) 로 효율/적중률 극대화

레이트리밋으로 상류 보호, 롤링 배포 중에도 다운타임 없이 동작

사후 감사/대사가 가능하도록 호출/판단 근거가 로그·메타데이터로 남아야 함

위와 같은 상황을 감안하여 다음 문제의도를 파악하고 해결하시오.

1. **게이트웨이 FSM 을 설계하시오. (조건부 업데이트를 중심으로 서술)**

1-1 집계 요청 단위의 상태를 정의하고, 상태 전이 규칙과 원자적(조건부) 업데이트 기준을 제시하라.

1-2 상류별 서브상태(예: INIT/IN_FLIGHT/SUCCESS/FAILED/STale/CB_OPEN)와 전체 응답 상태의 매핑 규칙을 정의하라.

1-3 캐시에 쓰는 시점/조건과, 동시에 도착하는 복수 요청에서 캐시 라인 갱신 충돌을 피하는 방법을 기술하라.

1-4 회로차단기(Open/Half-Open/Closed)와 FSM 의 상호작용(예: CB_OPEN 일 때 호출 우회+캐시 폴백)을 명시하라.

2. 처리 흐름에 대해 자세히 설명하시오.

2- 1 파라미터 정규화(정렬·소문자·기본값 대입) → 캐시 키 생성 → 캐시 조회 → 미스 시 병렬 상류 호출(http 타임아웃·재시도·지터) → ETag/If-None-Match 재검증 전략 → 부분 실패 판단과 폴백 순서(신선 캐시 > 오래된 캐시(stale) > 축약 응답/생략) → 응답 조립(partial/stale/sources[]/trace_id 메타 포함) → 캐시 저장(TTL/무효화 규칙) → 구조적 로깅/대사용 이벤트 기록의 전체 단계를 구체적으로 기술하라.

2-2 전역 시간 예산(p95 600ms 가정) 하에서 상류 read_timeout, 재시도 최대 횟수, 회로차단 임계값을 수치로 제시하고 근거를 설명하라.

2-3 레이트리밋(예: 토큰 버킷)의 키 스코프(클라이언트별/엔드포인트별/상류별)와 초과 시 처리(대기·즉시 429·폴백)를 결정하고 이유를 밝히라.

3. **다음 상황을 가정하였을 때 해결방안을 생각하고 서술하시오.**

상황 1: 3 개 상류 중 A, C 는 성공했으나 B 가 read timeout. Retry-After 는 없으며, B 의 stale 캐시(나이 10 분, 권장 TTL 5 분)가 존재한다.

전역 예산을 초과하지 않으면서 사용자가 체감 품질을 유지하도록, 응답 본문 구성 기준(어떤 필드를 생략/표기), partial/stale 플래그, sources[] 메타데이터 설계, 캐시 재적재(SWR 등) 전략을 제시하라.

동일 파라미터의 동시 다발 요청이 몰릴 때 B 재호출 폭주를 어떻게 억제할지(예: 캐시 라인락/싱글플라이트) 설명하라.

상황 2: News 상류가 ETag 를 지원한다. 우리 게이트웨이는 If-None-Match 로 조회했고 **304 Not Modified를 받았다. 그런데 우리 쪽 캐시 키 정규화 미흡(파라미터 순서 차이)으로 동일 콘텐츠가 서로 다른 키로 저장되어 적중률이 낮고 메모리도 낭비되고 있다.**

캐시 키 규칙(버전, 정렬, 소문자화, locale·lang 반영, 상류 etag 편입)을 재정의하고, 무효화/합치기(Migration) 절차를 제안하라.

304 수신 시 TTL 갱신 여부, stale-while-revalidate 와의 조합, 롤링 배포 중 키 스키마 변경의 호환성(구·신 공존) 전략을 설명하라.