백엔드 & 인프라 스터디 5 회차 문제

2025-11-16

작성자 : 김호중

상황 – JWT 기반 인증 시스템의 함정
당신은 온라인 쇼핑몰의 백엔드 개발자입니다. 이 쇼핑몰은 Django REST Framework 를 사용하며, JWT 로 사용자 인증을 처리합니다. Access Token 의 만료 시간은 30 분, Refresh Token 의 만료 시간은 14 일로 설정되어 있습니다.
어느 날, 한 고객으로부터 긴급 문의가 들어왔습니다. "제 계정이 해킹당해서 비밀번호를 바로 변경했는데, 해커가 계속 제 계정으로 주문을 하고 있어요!" 당신은 즉시 로그를 확인했지만, 해커의 모든 요청은 유효한 JWT 토큰으로 인증을 통과하고 있었습니다.
비슷한 사례가 또 발생했습니다. 한 직원이 퇴사하면서 계정을 비활성화했는데, 이 직원은 퇴사 후 일주일 동안 계속 시스템에 접근할 수 있었습니다.

문제
첫째, JWT 는 서버에 세션을 저장하지 않는 'Stateless(무상태)' 방식입니다. 이것이 왜 서버 확장에는 유리하지만, "토큰을 즉시 무효화할 수 없다"는 보안 문제를 일으키는지 설명하시오. 특히 서버가 JWT 를 검증할 때 실제로 무엇을 확인하는지, 그리고 비밀번호 변경이나 계정 비활성화 정보가 왜 즉시 반영되지 않는지 설명해야 합니다.

둘째, 많은 개발자들이 "Refresh Token 을 DB 에 저장하면 JWT 를 무효화할 수 있다"고 말합니다. 하지만 이 방법이 왜 Access Token 무효화 문제를 해결하지 못하는지 설명하시오. Access Token 의 남은 수명(30 분)이 공격자에게 어떤 위험을 주는지 구체적인 예시를 들어 설명하고, "Access Token 만료 시간을 1 분으로 줄이면 되지 않느냐"는 제안이 왜 현실적이지 않은지 사용자 경험과 서버 부하 측면에서 설명하시오.

셋째, JWT 무효화 문제를 해결하기 위한 두 가지 방법을 비교하시오. 방법 A: Redis 에 블랙리스트를 만들어서 무효화된 토큰의 ID 를 저장하는 방식. 이 방식이 JWT 의 Stateless 라는 장점을 어떻게 포기하게 되는지, 블랙리스트가 계속 커지는 문제를 어떻게 관리해야 하는지 설명하시오. 방법 B: Access Token 만료 시간을 짧게(3-5 분) 설정하고, 사용자가 활동할 때마다 자동으로 토큰을 갱신하는 방식. 이 방식이 프론트엔드 개발에 어떤 복잡성을 추가하는지 설명하시오.

넷째, JWT 의 Payload 는 암호화되지 않고 단순히 Base64 인코딩만 되어 있습니다. 즉, 누구나 디코딩해서 내용을 볼 수 있습니다. 한 개발자가 사용자 편의를 위해 JWT 에 user_id, email, phone_number, 주소 같은 정보를 모두 넣었습니다. 이것이 왜 개인정보 보호 관점에서 위험한지, 그리고 XSS 공격이 발생했을 때 어떤 문제가 생길 수 있는지 설명하시오. JWT 에 넣어도 안전한 정보와 절대 넣으면 안 되는 정보의 기준을 제시하시오.

다섯째, 당신의 시스템은 JWT 를 브라우저의 localStorage 에 저장하고 있습니다. 보안팀에서 이것이 XSS 공격에 취약하다고 지적했습니다. 한 개발자가 "그럼 HttpOnly Cookie 에 저장하면 되지 않느냐"고 제안했습니다. localStorage, HttpOnly Cookie, 메모리(JavaScript 변수)에 JWT 를 저장하는 세 가지 방식의 장단점을 XSS 와 CSRF 공격 가능성과 함께 비교하시오. 특히 SPA(Single Page Application)에서 새로고침 시 메모리의 토큰이 사라지는 문제를 어떻게 해결할 수 있는지 설명하시오.

상황 – Unix Signal 의 숨겨진 위험

당신은 실시간 로그 처리 시스템을 운영하는 백엔드 개발자입니다. log-processor 라는 Python 데몬이 Kafka 에서 로그를 읽어서 데이터베이스에 저장합니다. 배포할 때는 systemctl restart log-processor 를 실행하면, 프로세스가 현재 작업을 완료하고 깔끔하게 종료됩니다.

이를 위해 코드에는 다음과 같은 로직이 있습니다.

python

```
def graceful_shutdown(signum, frame):
    # 현재 처리 중인 배치 완료
    # Kafka 오프셋 커밋
    # DB 연결 정리
    sys.exit(0)

signal.signal(signal.SIGTERM, graceful_shutdown)
```

최근 이 시스템을 Kubernetes 로 옮겼습니다. Pod 의 종료 대기 시간(terminationGracePeriodSeconds)은 30 초로 설정했습니다. 처음 며칠은 문제없었지만, 일주일 후 배포 중에 심각한 문제가 발생했습니다. 데이터베이스에 일부만 저장된 데이터가 발견되었고, 약 3,000 개의 로그가 Kafka 에서 읽혔지만 어디에도 저장되지 않고 사라졌습니다.

문제

첫째, SIGTERM 과 SIGKILL 의 차이점이 무엇이며, Kubernetes 가 30 초 후에 SIGKILL 을 보내는 이유는 무엇입니까? graceful_shutdown 함수가 30 초 안에 끝나지 못했을 때 정확히 무슨 일이 벌어지는지, 프로세스의 메모리와 DB 연결이 어떻게 되는지 설명하시오. 특히 데이터베이스 트랜잭션이 커밋되지 않은 상태에서 프로세스가 강제 종료되면 데이터가 어떻게 되는지 설명하시오.

둘째, 한 개발자가 "graceful_shutdown 함수 실행 시간을 측정해보니 평균 3 초인데, 왜 30 초 안에 못 끝난 거죠?"라고 물었습니다. 이 질문은 Signal 처리에 대한 중요한 오해를 보여줍니다. Signal 이 프로세스에 전달되었을 때 핸들러가 언제 실행되는지, 그리고 메인 스레드가 time.sleep(60) 같은 블로킹 작업을 하고 있을 때 Signal 핸들러는 어떻게 동작하는지 설명하시오.

셋째, log-processor 는 multiprocessing 을 사용합니다. 메인 프로세스가 Kafka 에서 데이터를 읽고, 4 개의 워커 프로세스가 실제 처리를 담당합니다. 메인 프로세스가 SIGTERM 을 받았을 때, 이 Signal 이 워커 프로세스들에게 자동으로 전달됩니까? 전달되지 않는다면 graceful_shutdown 에서 워커들을 어떻게 정리해야 하는지, pool.terminate()와 pool.close(), pool.join()의 차이는 무엇인지 설명하시오.

넷째, 또 다른 개발자가 "SIGTERM 받으면 그냥 즉시 종료하면 안 되나요? 어차피 Kafka 는 오프셋을 커밋 안 하면 다시 처리하잖아요"라고 제안했습니다. 이 방식이 왜 문제인지 설명하시오. 특히 시스템이 데이터베이스와 Elasticsearch 에 동시에 쓰기를 할 때, 한쪽은 성공하고 한쪽은 실패하는 상황에서 데이터 중복이나 불일치가 어떻게 발생하는지 설명하시오.

다섯째, 문제 해결을 위한 두 가지 접근법을 비교하시오.
접근법 A: terminationGracePeriodSeconds 를 300 초(5 분)로 늘려서 충분한 시간을 확보하는 방법. 이 방식이 Kubernetes 의 배포 속도와 장애 복구에 어떤 영향을 주는지 설명하시오.

접근법 B: 각 로그를 개별적으로 처리하고 즉시 오프셋을 커밋하도록 설계를 변경하는 방법. 이 방식이 시스템 처리 속도에 어떤 영향을 미치는지, 배치 처리의 효율성과 어떻게 상충되는지 설명하시오.

상황 – Django REST Framework ViewSet 과신의 함정

당신은 전자상거래 플랫폼의 API 개발팀 리더입니다. 팀의 코딩 규칙은 명확했습니다. "모든 API 는 ViewSet 으로 만들고, Router 로 URL 을 자동 생성한다." 이 규칙 덕분에 코드가 일관적이었고, 신입 개발자도 쉽게 따라할 수 있었습니다.
핵심 리소스인 Order(주문)는 OrderViewSet 으로 구현되었습니다. 기본적인 CRUD 작업(list, retrieve, create, update, delete)과 함께, 비즈니스 로직을 위한 커스텀 액션들도 추가되었습니다. cancel(주문 취소), refund(환불), track(배송 추적) 같은 엔드포인트들입니다.
시간이 지나면서 요구사항이 복잡해졌습니다. "일반 사용자는 자기 주문만, 관리자는 모든 주문을 볼 수 있어야 한다", "주문 생성 시 재고 확인, 결제 처리, 포인트 차감을 모두 해야 한다", "주문 수정은 결제 대기 상태에서만 가능하다" 같은 비즈니스 규칙이 계속 추가되었습니다.
개발자들은 이 모든 로직을 OrderViewSet 안에 넣었습니다. 6 개월 후, OrderViewSet 은 1,000 줄이 넘는 거대한 클래스가 되었고, 아무도 전체 코드를 이해하지 못했습니다.

문제

첫째, ViewSet 의 기본 아이디어는 "하나의 리소스에 대한 모든 작업을 하나의 클래스에 모은다"는 것입니다. 이 접근법이 처음에는 좋지만, 비즈니스 로직이 복잡해지면 왜 문제가 되는지 설명하시오. 특히 OrderViewSet 이 "일반 사용자용 주문 조회", "관리자용 주문 관리", "통계 조회" 같은 서로 다른 역할을 모두 담당할 때, get_queryset 과 get_permissions 에 if 문이 가득 차게 되는 이유와, 이것이 코드 테스트와 유지보수에 어떤 문제를 일으키는지 설명하시오.

둘째, 한 개발자가 "ViewSet 은 같은 코드를 반복하지 않아도 돼서 좋지 않나요? 같은 Serializer 와 Queryset 을 여러 곳에서 쓸 수 있잖아요"라고 말했습니다. 이 주장의 문제점을 설명하시오. 예를 들어, 일반 사용자용 주문 목록과 관리자용 주문 목록이 완전히 다른 필터링, 정렬, 페이지 크기를 필요로 할 때, 이 둘을 하나의 ViewSet 으로 억지로 합치는 것이 왜 코드를 더 복잡하게 만드는지 설명하시오.

셋째, Router 가 자동으로 만드는 URL 패턴은 REST 원칙에 잘 맞습니다. 하지만 실제 비즈니스 API 는 종종 REST 를 벗어납니다. 예를 들어 POST /orders/{id}/cancel/은 RESTful 한가요? 이것은 새로운 리소스를 만드는 것이 아니라 주문의 상태를 바꾸는 "액션"입니다. 이런 액션 기반 API 를 ViewSet 의 @action 으로 구현할 때의 문제점을 설명하시오. 특히 cancel, refund, ship, deliver 같은 수십 개의 액션이 추가될 때, 이것이 왜 좋은 설계가 아닌지, 그리고 이 경우 별도의 APIView 로 분리하는 것이 왜 더 나은지 설명하시오.

넷째, 한 개발자가 "주문 목록 조회는 가능하지만 생성은 막고 싶다"고 요구했습니다. 이를 위해 create 메서드를 오버라이드해서 에러를 발생시키거나, http_method_names 를 제한했습니다. 이런 "기능을 제거하기 위해 코드를 추가하는" 상황이 왜 발생하는지 설명하시오. ViewSet 대신 GenericAPIView 를 직접 상속받아 필요한 기능만 조합하는 방법이나, APIView 를 사용하는 방법의 장단점을 비교하고, "언제 ViewSet 을 사용하지 말아야 하는가"에 대한 기준을 제시하시오.

다섯째, OrderViewSet 은 복잡한 관계를 다룹니다. Order 는 OrderItem 을 가지고, OrderItem 은 Product 를 참조하며, Order 는 User, Payment, Shipment 와 연결되어 있습니다. 주문 목록을 조회할 때 N+1 쿼리 문제가 발생해서, 100 개 주문을 조회하는데 1,000 개 이상의 SQL 쿼리가 실행됩니다. get_queryset 에서 select_related 와 prefetch_related 로 최적화하려 했지만, list 액션과 retrieve 액션이 필요로 하는 관계가 다르고, 커스텀 액션들도 각각 다른 관계를 필요로 합니다. 하나의 get_queryset 에서 if self.action == 'list' 같은 조건문으로 모든 케이스를 처리하는 것이 왜 문제인지 설명하고, 이를 해결하기 위한 방법(예: 읽기 전용 뷰 분리, CQRS 패턴)을 제시하시오.