2주차

1장. 사용자 수에 따른 규모 확장성

1. 단일 서버

실제 요청이 오는 경로

웹 어플리케이션

- 비즈니스 로직, 데이터 저장 등을 처리하기 위해 서버 구현용 언어(자바, 파이썬)를 사용
- 프렌젠테이션 용으로 클라이언트 구현용 언어(HTML, 자바스크립트)를 사용

모바일 어플리케이션

- 모바일 앱과 웹 서버 간의 통신을 위해 HTTP 프로토콜을 이용
    - HTTP 프로토콜을 통해 반환될 응답 데이터 포맷은 보통 JSON을 사용

---

2. 데이터베이스

사용자가 늘면 웹/모바일 트래픽 처리용 서버와 데이터베이스용 서버를 따로 둠

- 이 두 서버를 분리하면 각각을 독립적으로 확장해나가는 것이 가능

1) 어떤 베이스를 사용할 것인가?

관계형 데이터베이스(관리형 데이터베이스 관리 시스템), RDBMS

종류: MySQL, 오라클 데이터베이스, PostgerSQL

- 자료를 테이블과 열, 칼럼으로 표현

- SQL을 사용하여 여러 테이블에 있는 데이터를 관계에 따라 조인하여 합칠 수 있음

비관계형 데이터베이스

종류: CouchDB, Neo4j, Cassandra, HBase 등

분류: 키-값 저장소, 그래프 저장소, 칼럼 저장소, 문서 저장소

- 비관계형 데이터베이스는 일바적으로 조인 연산을 지원하지 않음

비관계형 데이터베이스 사용을 고려해야 하는 경우

- 아주 낮은 응답 지연시간이 요구됨
- 다루는 데이터가 비정형이라 관계형 데이터가 아님
- 데이터(JSON, YAML, XML)등을 직렬화하거나 역직렬화 할 수 있기만 하면 됨
- 아주 많은 양의 데이터를 저장할 필요가 있음

수직적 규모 확장 vs 수평적 규모 확장

수직적 규모 확장(Scale-up): 서버에 고사양 자원(더 좋은 CPU, 더 많은 RAM)을 추가하는 행위

- 서버로 유입되는 트래픽의 양이 적을 때 사용
    - 확장에 한계가 존재
    - 장애에 대한 자동복구 방안이나 자동화 제시 X

수평적 규모 확장(Scale-out): 더 많은 서버를 추가하여 성능을 개선하는 행위

- 대규모 어플리케이션을 지원할 때 사용

1) 로드밸런서

부하 분산 집합에 속한 웹 서버들에게 트래픽 부하를 고르게 분산하는 역할을 진행

로드밸런서 동작 과정

ⓐ 사용자가 로드밸런서의 공개 IP주소로 접속

ⓑ웹 서버는 클라이언트의 접속을 직접 처리하지 않음

- 서버 간 통신에 사설 IP 주소를 이용

ⓒ 로드밸런서는 웹 서버와 통신하기 위해 사설 IP 사용

부하 분산 집합에 또 하나의 웹 서버를 추가하면 장애를 자동복구하지 못하는 문제를 해결할 수 있으며 웹 계층의 가용성이 향상됨

- 서버 1이 다운되면 트래픽이 서버 2로 전송되어 웹 사이트 전체가 다운되는 일이 방지됨
- 로드밸런서를 사용하여 가파르게 증가하는 트래픽 분산이 용이

2) 데이터베이스 다중화

보통 마스터-슬레이브 관계를 설정하고 데이터 원본은 주 서버(마스터)에, 사본은 부 서버(슬레이브)에 저장하는 방식을 사용함

- 이때, 쓰기 연산은 마스터에서만 지원. 부 데이터베이스는 주 데이터베이스로부터 사본을 전달 받으며, 읽기 연산만을 지원함
- 데이터베이스를 변경하는 명령 - insert, delete, update- 은 주 데이터베이스로만 전달되어야 함
    - 대부분의 어플리케이션은 읽기 연산의 비중이 쓰기 연산보다 많음
        - 부 데이터베이스의 수 > 주 데이터베이스의 수

데이터베이스 다중화의 장점

- 더 나은 성능
    - 데이터 변경 연산은 주 데이터베이스 서버로, 읽기 연산은 부 데이터베이스 서버로 분산됨 → 병렬로 처리될 수 있는 질의의 수가 늘어나므로 성능이 ↑
- 안전성
    - 데이터베이스 서버 가운데 일부가 파괴되어도 데이터가 보존됨
        - 데이터를 지역적으로 떨어진 여러 장소에 다중화할 수 있기 때문
- 가용성
    - 데이터를 여러 지역에 복제, 데이터베이스 서버에 장애가 발생해도 다른 서버의 데이터를 가져와 서비스

부 서버가 한 대 뿐인데 다운되었을 경우 → 주 데이터베이스 서버가 읽기 연산도 일시적으로 수행, 또한 즉시 새로운 데이터 베이스 서버가 해당 서버의 역할을 수행함

부 서버가 여러 대인데 다운 되었을 경우 → 나머지 부 데이터 베이스 서버로 읽기 연산 분산, 새로운 부 데이터베이스 서버가 장애 서버 역할을 대체

주 데이터베이스가 다운 되었을 경우 → 부 데이터베이스 중 하나가 새로운 주 서버가 됨 + 데이터베이스의 모든 연산인 일시적으로 새로운 주 서버 상에서 수행됨, 이후 부 서버 추가 생성

프로덕션 환경에서는 부 서버에 보관된 데이터가 최신 상태가 아닐 수도 있음 → 복구 스크립트를 사용하여 없는 데이터를 복구

캐시

캐시란?

: 값비싼 연산 결과 혹은 자주 참조되는 데이터를 메모리 안에 두어 뒤이은 요청이 빠르게 처리 될 수 있도록 하는 저장소

- 어플리케이션의 성능은 데이터 베이스를 얼마나 자주 호출하느냐에 크게 좌우됨
    - 캐시는 이런 문제를 완화할 수 있음

1) 캐시 계층

: 데이터가 임시 보관되는 곳, 데이터베이스 보다 빠른 속도를 가짐

별도의 캐시 계층을 두면 성능 개선 + 데이터베이스의 부하를 줄이고, 규모를 독립적으로 확장시키는 것이 가능해짐

읽기 주도형 캐시 전략

- 요청을 받은 웹 서버가 캐시에 응답이 저장 되어 있는지 확인
    - 저장 되어 있다면 해당 데이터를 클라이언트에 반환
    - 저장 되어 있지 않다면 데이터베이스 질의를 통해 데이터를 찾아 캐시에 저장한 후 클라이언트에 반환
- 읽기 주도형 캐시 전략 외에도 캐시할 데이터의 종류, 크기, 액세스 패턴에 맞추어 캐시 전략을 선택

캐시 서버들은 일반적으로 널리 쓰이는 프로그래밍 언어로 API를 제공하므로 사용 방법이 간단

▼ 캐시 사용 시 유의할 점

- 캐시는 어떤 상황에 바람직한가?
    - 데이터 갱신은 자주 일어나지 않지만 참조는 빈번하게 일어날 때 고려해볼 만함
- 어떤 데이터를 캐시에 두어야 하는가?
    - 캐시는 데이터를 휘발선 메모리에 두므로 영속적으로 보관할 데이터를 캐시에 두는 것은 바람직 하지 않음
        - 중요한 데이터는 지속적 저장소에 두기
- 캐시에 보관된 데이터는 어떻게 만료되는가?
    - 이에 대한 정책을 마련해두는 것이 좋은 습관. 만료 정책이 없으면 데이터는 캐시에 계속 남게 됨
    - 단, 만료기간이 너무 짧으면 데이터베이스를 너무 자주 읽게 되고 너무 길면 원본과 차이가 날 가능성이 존재하므로 주의
- 일관성은 어떻게 유지되는가?
    - 저장소의 원본을 갱신하는 연산과 캐시 갱신 연산이 단일 트랜잭션으로 처리되지 않으면 일관성이 깨질 수 있음
        - 여러 지역에 걸쳐 시스템을 확장해 나가는 경우, 캐시와 저장소 사이의 일관성 유지는 어려운 문제가 됨
- 장애는 어떻게 대처할 것인가?
    - 캐시 서버를 한 대만 두는 경우, 해당 서버는 단일 장애 지점이(SPOF)될 가능성이 있음
        - 단일 장애 지점: 장애가 발생할 경우 전체 시스템의 동작을 중단 시킬 수 있는 지점
    - SPOF를 피하기 위해서 어려 지역에 걸쳐 캐시 서버를 분산 시켜야 함
- 캐시 메모리는 얼마나 크게 잡을 것인가?
    - 캐시 메모리가 너무 작을 경우 → 액세스 패턴에 따라, 데이터가 너무 자주 캐시에서 밀려나 캐시 성능이 떨어짐

        - 캐시 메모리를 과할당하여 문제 해결
- 데이터 방출 정책은 무엇인가?
    - 캐시가 꽉 차버리면 추가로 캐시에 데이터를 넣어야 할 경우 기존 데이터를 내보내야 함 → 데이터 방출 정책
        - LRU: 마지막으로 사용된 시점이 가장 오래된 데이터를 내보내기
            - 가장 많이 사용됨
        - LFU: 사용된 빈도가 가장 낮은 데이터를 내보내는 정책
        - FIFO: 가장 먼저 캐시에 들어온 데이터를 가장 먼저 내보내는 정책

콘텐츠 전송 네트워크(CDN)

CDN이란?

정적 콘텐츠를 전송하는 데 쓰이는, 지리적으로 분산된 서버의 네트워크. 이미지, 비디오, CSS, JavaScript 파일 등을 캐시할 수 있음

CDN 동작 과정

사용자가 웹 사이트 방문 시, 사용자와 가장 가까운 CDN 서버가 정적 콘텐츠를 전달

ⓐ 사용자가 이미지 URL을 이용해 이미지에 접근

- 이때, URL은 CDN사업자가 제공함

ⓑ CDN 서버의 캐시에 해당 이미지가 없는 경우, 서버는 원본 서버(웹 서버 혹은 스토리지)에 요청하여 파일을 가져옴

ⓒ 원본 서버가 파일을 CDN 서버에 반환

ⓓ CDN 서버는 파일을 캐시하고 사용자 A에게 반환

ⓔ 다른 사용자가 같은 이미지에 대한 요청을 CDN 서버에 전송

ⓕ 만료되지 않은 이미지에 대한 요청은 캐시를 통해서 처리

▼ CDN 사용 시 고려해야 할 점

- 비용

- 적절한 만료 시한 설정
    - 시의성이 증요한 콘텐츠의 경우 만료 시점을 잘 정해야 함
    - 너무 길 경우 → 콘텐츠의 신선도가 떨어짐
    - 너무 짧을 경우 → 원본 서버에 빈번히 접속해야 함
- CDN 장애에 대한 대처 방안
    - CDN 자체가 죽었을 경우 웹 사이트/애플리케이션이 어떻게 동작해야 하는지 고려해야 함
        - 일시적으로 CDN이 응답하지 않을 경우 해당 문제를 감지, 원본 서버로부터 직접 곤텐츠를 가져오도록 클라이언트를 구성
- 콘텐츠 무효화 방법(만료되지 않은 콘텐츠를 CDN에서 제거하기)
    - CDN 서비스 사업자가 제공하는 API를 이용하여 콘텐츠 무효화
    - 콘텐츠의 다른 버전을 서비스하도록 오브젝트 버저닝이용
        - 콘텐츠의 새로운 버전을 지정하기 위해서는 URL 마지막에 버전번호를 인자로 주면 됨

무상태 웹 계층

웹 계층을 수평적으로 확장하기 위해서는 사용자 세션 데이터와 같은 상태 정보를 웹 계층에서 제거해야 함

- 데이터 베이스 등 지속성 저장소에 보관하고, 필요할 때 가져오는 것이 가장 좋음

⇒ 상태 정보를 지속성 저장소에 저정하도록 구성된 웹 계층이 무상태 웹 계층

상태 정보 의존적인 아키텍처

- 상태 정보를 보관하는 서버
    - 클라이언트 정보(상태)를 유지하여 요청들 사이에 공유되도록 함
    - 로드밸런서에 부담 + 로드밸런서 뒷단에 서버 추가 혹은 제거가 어려워짐
        - 서버 장애 처리 복잡도 역시 증가

무상태 아키텍처

- 웹 서버는 상태 정보가 필요한 경우 저장소로부터 데이터를 가져옴
    - 상태 정보가 웹 서버로부터 물리적으로 분리된 상태
        - 단순, 안정적, 규모 확장 용이

데이터 센터

다중 데이터 센터 아키텍처 구축을 위해 필요한 기술

- 트래픽 우회
    - 올바른 데이터 센터로 트래픽을 보내는 효과적인 방법을 찾아야 함
        - GeoDNS가 사용자에게 가장 가까운 데이터 센터로 트래픽을 보낼 수 있도록 도와줌
- 데이터 동기화
    - 데이터 센터마다 별도의 데이터베이스를 사용할 경우, 장애가 자동을 복구되어 트래픽이 다른 데이터베이스로 우회해도 해당 데이터 센터에서는 찾고자 하는 데이터가 없을 수 있음
        - 데이터를 여러 데이터 센터에 걸쳐 다중화 → 문제 해결
- 테스트와 배포
    - 웹 사이트 혹은 애플리케이션을 여러 위치에서 테스트해보기
    - 자동화된 배포 도구: 모든 데이터 센터에 동일한 서비스가 설치되도록 하는 데 중요한 역할

메시지 큐

시스템을 더 큰 규모로 확장하기 위해서는 시스템의 컴포넌트를 분리, 각기 독립적으로 확장될 수 있도록 해야 함 → 메시지 큐를 사용해서 달성

메시지 큐란?

메시지의 무손실을 보장 및 비동기 통신을 지원하는 컴포넌트

- 메시지 큐에 일단 보관된 메시지는 소비가자 꺼낼 때 까지 안전히 보관 → 메시지 무손실 보장

메시지 큐 기본 아키텍처

- 생산자 혹은 발행자라 불리는 입력 서비스가 메시지를 만들어 메시지 큐에 발행
- 큐에는 보통 소비자 혹은 구독자라 불리는 서비스 혹은 서버가 연결되어 있음
    - 메시지를 받아 그에 맞는 동작을 수행

메시지 큐를 이용하면 서비스 혹은 서버 간 결합이 느슨해져 규모 확장이 보장되어야 하는 안정적 애플리케이션을 구성하기 좋음

- 생산자: 소비자 프로세스가 다운되어 있어도 메시지 발행 가능
- 소비자: 생산자 서비스가 가용 상태가 아니어도 메시지 수신 가능

로그, 메트릭 그리고 자동화

소규모 웹 사이트를 만들 때, 로그나 메트릭, 자동화는 하면 좋지만 꼭 할 필요는 없음. 그러나 웹 사이트와 함께 사업 규모가 커진다면 이런 도구에 필수적으로 투자해야 함

로그

- 에러 로그를 모니터링하는 것은 무척 중요 → 시스템의 오류와 문제들을 보다 쉽게 찾아낼 수 있기 때문
    - 에러 로그는 서버 단위로 모니터링 할 수도 있지만, 로그를 단일 서비스로 모아주는 도구를 활용하면 더 좋음

메트릭

- 메트릭을 잘 수집하면 사업 현황에 관한 유용한 정보를 얻을 수 있고, 시스템의 현재 상태를 손쉽게 파악할 수도 있음
- 메트릭 종류
    - 호스트 단위 메트릭: CPU, 메모리, 디스크 I/O에 관한 메트릭이 해당
    - 종합 메트릭: 데이터베이스 계층의 성능, 캐시 계층의 성능
    - 핵심 비즈니스 메트릭: 일별 능동 사용자, 수익, 재방문

자동화

- 시스템이 크고 복잡해지면 생산성을 높이기 위해 자동화 도구를 활용해야 함
    - 지속적 통합, 빌드, 테스트, 배포 등의 절차를 자동화 → 개발 생산성 향상 가능

데이터베이스의 규모 확장

저장할 데이터가 많아지면 데이터베이스에 대한 부하도 증가하므로 데이터베이스를 증설할 방법을 찾아야 함

수직적 확장(Scale up)

: 기존 서버에 더 많은 고성능의 자원을 증설하는 방법

- 데이터베이스 서버 하드웨어에는 한계가 존재 → CPU, RAM 등을 무한 증설 불가능
    - 사용자가 계속 늘어날 경우, 한 대 서버로는 감당이 어려워짐
- SPOF로 인한 위험성 증가
- 비용이 많이듬..

수평적 확장(Scale out)

- 데이터베이스의 수평적 확장은 샤딩이라고 부름
    - 더 많은 서버를 추가하여 성능을 확장 시키는 것
    - 샤딩: 대규모 데이터베이스를 샤드라고 부르는 작은 단위로 분할하는 기술
        - 모든 샤드는 같은 스키마를 사용, 샤드에 보관되는 데이터 사이에 중복은 없음

샤딩 전략 시 샤딩 키를 어떻게 정하느냐가 중요한 점이 됨

- 샤딩 키(파티션 키): 데이터가 어떻게 분산될지 정하는 하나 이상의 칼럼
    - 샤딩 키를 통해 올바른 데이터베이스에 질의를 보내고 데이터 조회 혹은 변경을 처리 → 효율을 높일 수 있음
- 데이터를 고르게 분할할 수 있을 지 <<가 가장 중요한 포인트

샤딩 문제점

- 데이터 재 샤딩
  - 재 샤딩이 필요한 경우
    - 데이터가 너무 많아져서 하나의 샤드로 더 이상 감당하기 어려울 때
    - 샤드 간 데이터 분포가 균등하지 못하여 어떤 샤드에 할당된 공간 소모가 다른 샤드에 비해 빨리 진행될 때
      - 샤드 소진 현상이 발생할 경우, 샤드 키를 계산하는 함수를 변경하고 데이터를 재 배치해야 함
- 유명인사 문제
  - 핫스팟 키 문제, 특정 샤드에 질의가 집중 되어 서버에 과부하가 걸리는 문제
- 조인과 비정규화
  - 데이터베이스를 여러 샤드 서버로 쪼개면 여러 샤드에 걸친 데이터를 조인하기 힘듦
    - 데이터베이스를 비정규화하여 하나의 테이블에서 질의를 수행할 수 있도록 해야 함

백만 사용자, 그리고 그 이상

시스템의 규모를 확장하는 것은 지속적이고 반복적인 과정

수백만 이상을 지원하려면 새로운 전략을 도입하고 지속적으로 시스템을 가다듬어야 함

2장. 개략적인 규모 추정

개략적 규모 추정을 효과적으로 해 내려면 규모 확장성을 표현하는 데 필요한 기본기(2의 제곱수, 응답 지연 값, 가용성에 관계된 수치들)에 능숙해야 함

2의 제곱수

분산 시스템에서 다루는 데이터 양은 엄청나게 커질 수 있으나, 계산법은 기본을 벗어나지 않음

- 제대로 된 계산 결과를 얻으려면 데이터 볼륨의 단위를 2의 제곱수로 표현하면 어떻게 되는지 알아야!

2의 x 제곱	근사치	이름	축약형
10	1천(thousand)	1킬로바이트(Kilobyte)	1KB
20	1백만(million)	1메가바이트(Megabyte)	1MB
30	10억(billion)	1기가바이트(Gigabyte)	1GB
40	1조(trillion)	1테라바이트(Terabyte)	1TB
50	1000조(quadrillion)	1페타바이트(Petabyte)	1PB

모든 프로그래머가 알아야 하는 응답지연 값

연산명	시간
L1 캐시 참조	0.5ns
분기 예측 오류(branch mispredict)	5ns
L2 캐시 참조	7ns
뮤텍스 락/언락	100ns
주 메모리 참조	100ns
Zippy로 1 KB 압축	10,000ns = 10μs
1 Gbps 네트워크로 2 KB 전송	20,000ns = 20μs
메모리에서 1 MB 순차적으로 read	250,000ns = 250μs
같은 데이터 센터 내에서의 메시지 왕복 지연시간	500,000ns = 500μs
디스크 탐색(seek)	10,000,000ns = 10ms
네트워크에서 1 MB 순차적으로 read	10,000,000ns = 10ms
디스크에서 1 MB 순차적으로 read	30,000,000ns = 30ms
한 패킷의 CA(캘리포니아)로부터 네덜란드까지의 왕복 지연시간	150,000,000ns = 150ms

ns=나노초, μs=마이크로초, ms=밀리초
1나노초 = 10^{-9}초
1마이크로초 = 10^{-6}초 = 1,000나노초
1밀리초 = 10^{-3}초 = 1,000마이크로초 = 1,000,000나노초

- 메모리는 빠르지만 디스크는 여전히 느림

- 디스크 탐색은 가능한 피하라
- 단순한 압축 알고리즘은 빠름
- 데이터를 인터넷으로 전송하기 전에 가능하면 압축
- 데이터 센터는 보통 여러 지역에 분산, 센터들 간에 데이터를 주고 받는 데는 시간이 걸림

가용성에 관계된 수치들

고가용성이란?

: 시스템이 오랜 시간 동안 지속적으로 중단 없이 운영될 수 있는 능력

- 고가용성을 표현하는 값은 퍼센트로 표현됨
  - 100%는 단 시스템이 단 한번도 중단된 적이 없음을 의미
  - 대부분의 시스템은 99~100% 사이의 값을 가짐

SLA(Service Level Agreement)

: 서비스 사업자가 보편적으로 사용하는 용오, 서비스 사업자와 고객 사이에 맺어진 합의를 의미

- SLA에는 서비스 사업자가 제공하는 서비스 가용 시간이 공식적으로 기술 되어 있음
- 가용 시간은 관습적으로 숫자 9를 사용하여서 표현

9의 개수와 시스템 장애 시간 사이의 관계 표

가용률	하루당 장애시간	주당 장애시간	개월당 장애시간	연간 장애시간
99%	14.40분	1.68시간	7.31시간	3.65일
99.9%	1.44분	10.08분	43.84분	8.77시간
99.99%	8.64초	1.01분	4.38분	52.60분
99.999%	864.00밀리초	6.05초	26.30초	5.26분
99.9999%	86.40밀리초	604.80밀리초	2.63초	31.56초

예제3: 트위터 QPS와 저장소 요구량 측정

가정

- 월간 능동 사용자는 3억 명
- 50%의 사용자가 트위터를 매일 사용
- 평균적으로 각 사용자는 매일 2건의 트윗을 올림
- 미디어를 포함하는 트윗은 10%
- 데이터는 5년간 보관

추정

QPS(Query Per Second) 추정

- 일간 능동 사용자 = 3억 * 50% = 1.5억
- QPS: 1.5억 x 2트윗 / 24시간 / 3600초 = 약 3500
- 최대 QPS: 2x QPS = 7000

미디어 저장을 위한 저장소 요구량

- 평균 트윗 크기
  - tweet_id에 64바이트
  - 텍스트에 140 바이트
  - 미디어에 1mb
- 미디어 저장소 요구량: 1.5억 x 2 x 10%x 1mb = 30TB
- 5년간 미디어를 보관하기 위한 저장소 요구량: 30tb x 365 x 5 = 약 55pb

팁

올바른 절차를 밟느냐가 결과를 내는 것 보다 중요~

- 근사치를 활용한 계산
- 가정들을 적어두기
- 단위를 붙이기
- 많이 출제 되는 개략적 규모 추정 문제
    - QPS, 최대 QPS, 저장소 요구량, 캐시 요구량, 서버 수 추정 등

3장. 시스템 설계 면접 공략법

시스템 설계 면접은 두 명의 동료가 모호한 문제를 풀기 위해 협력하여 그 해결책을 찾아내는 과정에 대한 시뮬레이션임

효과적 면접을 위한 4단계 접근법

1단계 - 문제 이해 및 설계 범위 확정

- 요구 사항을 완전히 이해하지 않고 답을 내놓는 행위는 아주 부정적인 신호임
    - 깊이 생각하고 질문해서 요구사항과 가정을 분명히 하기
- 엔지니어가 가져야 할 가장 중요한 기술
    - 올바른 질문을 하기
    - 가정 하기
    - 시스템 구축에 필요한 정보 모으기
- 면접관이 질문에 대해 스스로 어떤 가정을 하기를 원한다면 가정을 적어두기
- 요구사항을 정확히 이해하는 데 필요한 질문 하기
    - 구체적으로 어떤 기능들을 만들어야 하나?
    - 제품 사용자 수는 얼마나 되나?
    - 회사의 규모는 얼마나 빨리 커지리라 예상?
    - 회사가 주로 사용하는 기술 스택은 무엇인가? 설계를 단순화하기 위해 활용할 수 있는 기존 서비스는 무엇?

⇒ 요구사항을 이해하고 모호함을 없애는 것이 중요

2단계 - 개략적인 설계안 제시 및 동의 구하기

개략적인 설계안을 제시하고 면접관의 동의를 얻는 것이 중요!

- 설계안에 대한 최초 청사진을 제시하고 의견을 구하기
- 핵심 컴포넌트를 포함하는 다어그램 그리기
  - 클라이언트, API, 웹 서버, 데이터 저장소, 캐시, CDN, 메시지 큐 등을 포함
- 최초 설계안이 시스템 규모에 관계된 제약 사항을 만족하는 지를 개략적으로 계산해보기

⇒ 해당 단계에서 API 엔드 포인트나 데이터베이스 스키마를 보여야 하는 지는 질문에 따라 다름

3단계 - 상세 설계

3단계까지 왔다면 우리는 면접관과 다음의 목표를 달성한 상태가 됨

- 시스템에서 전반적으로 달성해야 할 목표와 기능 범위 확인
- 전체 설계의 개략적 청사진 마련
- 해당 청사진에 대한 면접관의 의견 청취
- 상세 설계에서 집중해야 할 영역들 확인

면바면이지만, 우리는 이제 설계 대상 컴포넌트 사이의 우선순위를 정해야 함

- 만약 출제 문제가 단축 URL 생성기 설계에 관한 것이라면, 면접관은 해시 함수 설계를 구체적으로듣기를 원할 것
  - 사소한 세부사항 설명에 능력을 보일 수 있는 기획을 놓치지 말자

4단계 - 마무리

해당 단계에서는 면접관이 설계 결과물에 관련된 몇가지 후속 질문을 던질 수도 있고, 우리 스스로 추가 논의를 진행하도록 할 수도 있음

- 면접관이 시스템 병목 구간, 혹은 더 개선 가능한 지점을 찾아내라고 주문할 수 있음
    - 비판적 사고 능력을 보이고 좋은 인상을 남길 수 있는 기회
- 내 설계를 다시 한번 요약해주는 것도 도움이 됨
    - 여러 해결책을 제시한 경우 특히 중요
- 오류 발생시 무슨 일이 생기는지를 따져보는 것도 흥미로움
- 운영 이슈
    - 메트릭 수집 방법
    - 모니터링 여부
    - 배포 방법
- 미래에 발생할 수 있는 규모 확장 요구에 대한 대처 방안
- 그외 세부적 개선사항 제안

면접에서 해야 할 것

- 질문을 통해 확인하기
- 문제의 요구사항 이해하기
- 정답이나 최선의 답이 없음을 명심하기
    - 요구사항을 정확하게 이해했는지 확인하기
- 면접관이 나의 사고 흐름을 이해할 수 있도록 하기 - 면접관과의 소통
- 여러 해법 함께 제시하기
- 개략적 설계에 면접관이 동의한다면 각 컴포넌트 세부사항 설명 시작
    - 가장 중요한 컴포넌트부터 설명
- 면접관의 아이디어 이끌어내기
- 포기하지 말기

면접에서 하지 말아야할 것

- 전형적인 면접 문제에 대해서도 대비하지 않은 상태로 면접장 가지 말기
- 요구사항이나 가정이 분명하지 않은 상태에서 설계 제시 하지 말기

- 처음부터 특정 컴포넌트의 세부사항을 너무 깊이 설명하지 말기
    - 개략 설계 마친 후 세부사항으로 나아가기
- 진행 중에 막힌 부분에 대해 힌트 요청 주저하지 말기
- 소통 주저하지 말기
- 면접관이 끝났다고 말하기 전까지 면접 끝났다고 생각하지 말기

시간 배분

시스템 설계 면접은 보통 매우 광범위한 영역을 다루고, 45분 혹은 한 시간도 충분하지 않을 수 있음

→ 시간관리 매우 중요

45분이라는 가정하에

1단계 - 문제 이해 및 설계 범위 확정: 3 ~ 10분

2단계 - 개략적 설계안 제시 및 동의 구하기 : 10 ~ 15분

3단계 - 상세 설계: 10~ 25분

4단계 - 마무리: 3 ~ 5분