W05
데이터 환경과 서버 구성

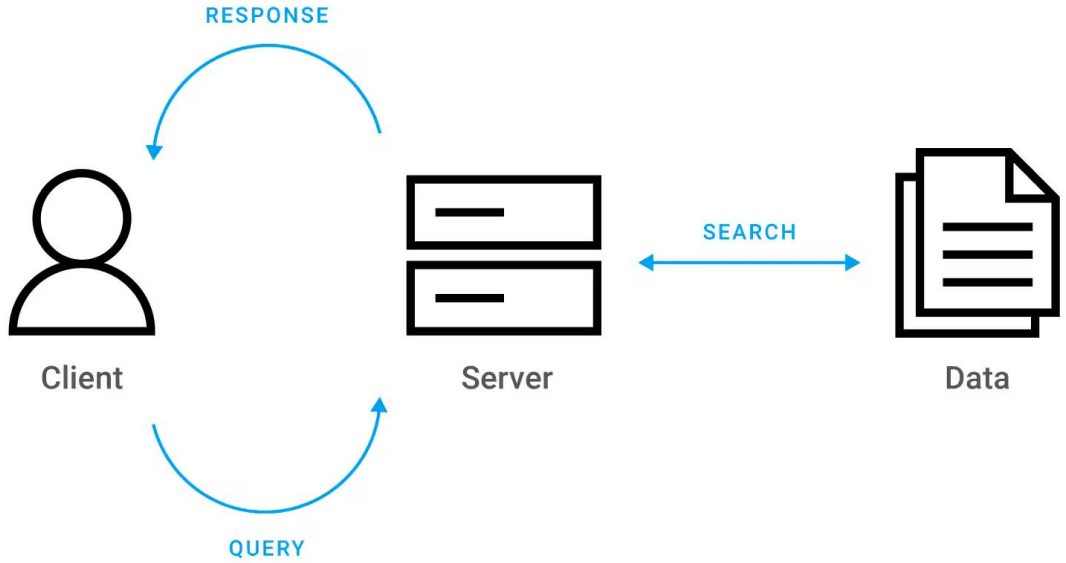
서버 없이 테스트하는 방법

데이터 입력은 어떻게 구성되어 있을까?

- 기능이 만들어지면서 테스트를 하려면 실제 데이터가 필요하다
- 서비스가 다 만들어지지 않은 상태인데 데이터를 어떻게 입력해야 할까?
- 임의로 폴더에 필요한 데이터를 넣어놓고 그때그때 사용하게 해야 할까?
- 아니면 실제 데이터 입력 시스템을 모두 설계하고 그 다음에 테스트를 해야할까?

서버 - 클라이언트 시스템

What is the client/server model?



Akamai

클라이언트는 어떤 일을 하나

클라이언트 = 사용자가 직접 보고 만지는 부분

- 웹 브라우저, 스마트폰 앱 등 사용자 화면
- 사용자의 입력(클릭, 타이핑)을 받아서 서버에 요청을 보냄
- 서버에서 받은 응답을 화면에 보여줌

서버는 어떤 일을 하나

서버 = 뒤에서 실제 일을 처리하는 부분

- 클라이언트의 요청을 받아서 처리
- 데이터베이스에서 데이터를 가져오거나 저장
- 계산, 검증 등 비즈니스 로직 실행
- 처리 결과를 클라이언트에 응답

API란?

- API = 클라이언트와 서버가 대화하는 약속된 방식
- "이렇게 요청하면, 이렇게 응답한다"는 규칙

예시:

- GET /users → 사용자 목록 달라
- POST /users → 새 사용자 만들어줘
- DELETE /users/3 → 3번 사용자 삭제해줘

CRUD란?

데이터를 처리하는데 필요한 행동은 어떤것들이 있을까?

예시) 메모장

- 메모를 새로 쓴다 → Create
- 메모 목록을 본다 → Read
- 메모 내용을 고친다 → Update
- 메모를 지운다 → Delete

CRUD란? = 서버 개발시에 데이터 처리 방식을 균일화하여 API 개발을 효율적으로 하기 위한 규칙

- Create: 만들기 (글쓰기, 회원가입)
- Read: 보기 (목록, 상세페이지)
- Update: 수정 (글 편집, 정보 변경)
- Delete: 삭제 (글 삭제, 탈퇴)

데이터 = 사용자 입력 + 데이터 베이스

사용자 입력: 사용자가 화면에서 직접 넣는 데이터

- 로그인 정보, 게시글 내용, 검색어 등

데이터베이스: 이미 저장되어 있는 데이터

- 기존 사용자 정보, 과거 게시글, 설정값 등

테스트할 때의 문제:

- 사용자 입력은 직접 넣으면 됨
- 근데 데이터베이스에 있어야 할 데이터는 어떻게 준비해야 하나?

테스트를 위해 필요한 부분만 구현해야 한다

"다 만들고 테스트하자"는 불가능하다

- 서버 완성 → DB 완성 → API 완성 → 그 다음 테스트?
- 이러면 중간에 문제 발견해도 너무 늦음
- 수정 비용이 기하급수적으로 증가

해결책: 단계별 접근

- UI만 확인할 거면?‘
- 데이터 흐름 테스트할 거면?
- API 호출 방식까지 테스트할 거면?
- 실제 저장/조회 테스트할 거면?
- 배포 전 최종 테스트할 거면?

방법 0. 하드 코딩

코드 안에 데이터를 직접 넣어놓는 방식

- const users = [
  - { id: 1, name: "홍길동" },
  - { id: 2, name: "김철수" }
-]

언제 사용하나:

- UI 레이아웃만 빠르게 확인할 때
- "이 화면에 데이터 3개 있으면 어떻게 보이지?" 테스트
- 장점: 가장 빠름. 바로 확인 가능.
- 한계: 데이터 바꾸려면 코드 수정해야 함. 저장/수정/삭제 동작 테스트 불가.

방법 1. Mock 데이터

가짜 데이터를 별도 파일로 분리해서 사용

- data/mockUsers.json
- 파일에 테스트용 데이터를 만들어두고, 코드에서 이 파일을 불러다 씀

언제 쓰나:

- 여러 화면에서 같은 데이터를 써야 할 때
- 다양한 케이스(데이터 0개, 100개, 특수문자 포함 등) 테스트할 때
- 장점: 데이터만 바꾸면 됨. 코드 수정 불필요. 테스트 케이스별 데이터 세트 관리 가능.
- 한계: 아직 "요청-응답" 흐름은 테스트 못함. 실제 API 호출 방식이 아님.

방법 2. Mock API

가짜 서버를 만들고 실제가 API 호출하는 방식으로 테스트

- json-server 같은 도구로 가짜 서버를 만들고
- 클라이언트는 진짜 서버인 것처럼 fetch('/api/users') 호출 가능

언제 사용하나:

- 클라이언트의 API 호출 로직을 테스트할 때
- 프론트엔드와 백엔드를 따로 개발할 때
- 외부 API(결제, 소셜로그인 등)를 흉내낼 때
- 장점: 실제 API 호출 방식 그대로 테스트. 나중에 진짜 서버로 바꿔도 코드 수정 최소화.
- 한계: 가짜 서버라서 복잡한 로직(검증, 계산 등) 테스트는 어려움.

방법 3. 로컬 DB

내 컴퓨터에 실제 데이터베이스를 설치해서 사용

- SQLite 같은 가벼운 DB를 로컬에 설치
- 실제 저장/조회/수정/삭제가 동작함.

언제 사용하나:

- 데이터 저장/조회 로직이 제대로 동작하는지 테스트할 때
- DB 구조(스키마)를 실험하고 확정할 때
- 인터넷 없이 개발할 때
- 장점: 실제 DB 동작 테스트 가능. 데이터가 진짜 저장됨. 구조 변경 실험이 자유로움.
- 한계: 내 컴퓨터에만 있음. 다른 사람과 공유 안 됨. 배포 환경과 설정이 다를 수 있음.

방법 4. 클라우드 서버(실제 구현)

인터넷에 있는 실제 서버와 데이터베이스 사용

- Supabase, Firebase, AWS 등 클라우드 서비스에 DB를 만들고 연결

언제 사용하나:

- 배포 전 최종 테스트
- 여러 사람이 같은 데이터로 테스트해야 할 때
- 실제 서비스 환경과 동일한 조건으로 테스트할 때
- 장점: 실제 서비스와 동일한 환경. 어디서든 접근 가능. 협업 가능.
- 한계: 인터넷 필요. 비용 발생 가능. 설정 복잡. 데이터 잘못 넣으면 복구 어려움.

서버를 먼저 다 만들어서 테스트할 수는 없다

실제 서버를 모두 구현해놓고 테스트하기는 사실상 불가능하다.

- 서버 구현에만 몇 주가 걸릴 수 있음
- 그동안 프론트엔드 개발자은 멈춰있어야
- 다 만들고 테스트했더니 구조가 잘못됐으면? → 처음부터 다시

단계별 접근의 이점:

- 빠른 피드백: 작은 단위로 자주 테스트
- 위험 분산: 문제를 일찍 발견
- 병렬 작업: 프론트/백엔드가 각자 진행 가능

질문 1. 테스트 단계 이해

“지금 단계에서 테스트를 하기 위해 입력 데이터를 어떤 방식으로 구성하는것이 좋을까? 서버 구현 정도와 테스트 데이터 연결, 이후 개발 로드맵을 고려해 최적의 방식이 뭔지 검토해줘"

관계형 데이터베이스

데이터를 저장할 때 어떤 문제가 있을까?

- 예시) 게시판
  - 사용자 정보: 이름, 이메일
  - 게시글 정보: 제목, 내용, 작성자
  - 댓글 정보: 내용, 작성자, 어떤 글에 달린 건지
- "작성자"가 중복됨. 사용자 이름이 바뀌면 게시글이랑 댓글에 있는 이름도 다 바꿔야 하나?

관계형 데이터베이스(RDB) = 데이터를 테이블로 나누고, 관계로 연결해서 중복을 없애는 방식

- 사용자 테이블: 작성자_id, 이름, 이메일
- 게시글 테이블: 게시글_id, 제목, 내용, 작성자_id (사용자 id 참조)
- 댓글 테이블: 댓글_id, 내용, 작성자_id, 게시글_id

관계형 데이터베이스

제품목록

제품ID(고유값)

제품ID	제품명	가격
Flower001	장미 100송이	100,000
Flower002	축화화환 3단	99,000
Flower003	검정리본 근조화환	125,000
Flower004	동양란 기본	68,000

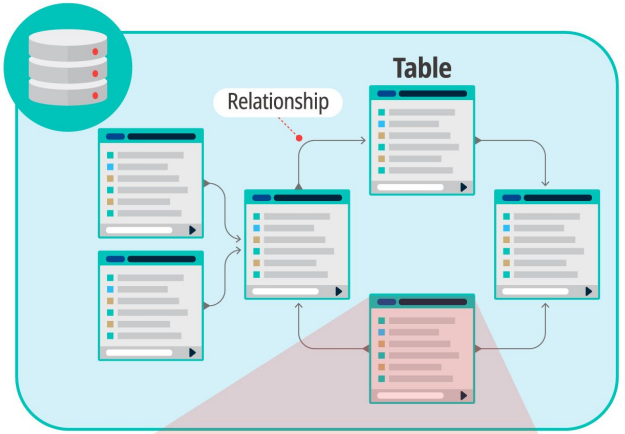
고객목록

고객ID(고유값)

고객ID	고객명	연락처
Cust001	강은주	010-2345-5555
Cust002	박하은	0101234-9876
Cust003	김태진	010-1333-4949
Cust004	황아름	010-2239-2202

판매기록

제품ID	고객ID	수량
Flower002	Cust001	5
Flower004	Cust003	1
Flower003	Cust002	1
Flower001	Cust002	3
Flower004	Cust001	3
Flower001	Cust004	1
Flower003	Cust003	5



ID	Last Name	Family Name	Sex
001	Kim	Chul-soo	M
002	Park	Yoo-mi	F
003	Lee	Ji-soo	F

RDB는 데이터 구조를 바꾸기 힘들다

한번 정한 구조(스키마)를 바꾸는 건 비용이 크다

- 예시: 사용자 테이블에 "전화번호" 컬럼을 추가하고 싶다
- 테이블 구조 변경 필요 (마이그레이션)
- 기존 데이터는 전화번호가 없음 → 어떻게 처리?
- 이 컬럼을 쓰는 모든 코드 수정 필요

DB 구조는 신중하게 결정해야 함

- 확정 전에 로컬 DB로 충분히 실험
- “DB 마이그레이션 필요합니다" : DB 구조 변경 작업이 필요하다

DB가 확정되어야 서버 개발이 가능하다

서버 코드는 DB 구조에 의존한다

- 예를 들어 "사용자 정보 저장" 기능을 만들려면:
- 어떤 필드가 있는지 (이름, 이메일, 비밀번호...)
- 어떤 필드가 필수인지
- 다른 테이블과 어떻게 연결되는지

이런 것들이 정해져야 서버를 개발할 수 있다

- Mock/로컬 DB로 구조 실험
- 이 데이터베이스 구조가 확정이 되었다
- 그 다음 서버 개발 및 배포 가능

DB를 만든 이후의 테스트

구조가 확정되면, 테스트용 데이터가 필요하다

시드 데이터(Seed Data):

- DB 초기 세팅용 샘플 데이터
- 테스트할 때마다 같은 상태에서 시작할 수 있음
- "DB 리셋하고 시드 데이터 다시 넣어줘"로 초기화 가능

질문 2. 데이터 환경 구성

"현재 프로젝트의 DB 구조를 검토하고, 테스트에 필요한 시드 데이터를 만들어줘."

"정상 케이스와 예외 케이스(빈 값, 특수문자, 대량 데이터 등)를 포함해서 다양한 상황을 테스트할 수 있게 구성해줘."

실제 서버가 구현이 된 이후

실제 서버에 배포하면 고려해야 하는 것들:

- 인증: 로그인, 토큰, 권한 체크
- 네트워크: 지연, 타임아웃, 연결 끊김
- 보안: HTTPS, 데이터 암호화
- 동시성: 여러 사용자가 동시에 접근

오늘 해결해야 할 문제

1. 현재 프로젝트의 데이터 환경 상태가 단계인지 파악합니다.
2. 지금 테스트하려는 것에 맞는 데이터 환경인지 검토합니다.
3. 필요하다면 AI와 함께 적절한 단계로 전환합니다.
4. 테스트용 데이터(Mock 또는 시드)를 구성하고 기능을 테스트합니다.
5. (선택) 로컬에서 작업하면서 DB 구조 변경이 필요한지 검토하고, 필요한 경우 AI와 함께 변경을 진행합니다.

Q&A