[주니어 개발자 채용] 주니어 개발자 기술 리뷰 1:1 면접 진행

목표

실전 과제 결과물을 바탕으로 단님과 **1:1 기술 리뷰 면접**을 진행하여,

“지금 수준에서 혼자 프론트 맡길 수 있는가?”를 스스로 납득할 만큼 확인한다.

인터뷰 포맷

- 참석자: 은향 단독 (고문님 X)
- 방식: 단님이 과제 코드/설계를 설명 → 중간중간 은향이 질문
- 시간: 45~60분 정도

질문 축 (예시)

1. **구조/폴더링**
    - 왜 이런 폴더/레이어 구조를 선택했는지
    - 다른 대안 구조를 고려했다면 무엇이었는지
2. **상태관리 패턴**
    - 어떤 상태관리 방법(Bloc, Provider, Riverpod 등)을 선택했는지
    - 그 선택이 이 프로젝트 맥락에서 어떤 장단점을 가지는지
3. **Firestore/데이터 흐름**
    - Firestore를 어디에서 어떻게 연결할 계획인지
    - 데이터 모델/스키마를 어떻게 설계했는지(또는 설계하고 싶은지)
4. **우선순위 변경 대응**
    - 요구사항이 바뀌거나 우선순위가 뒤집혔을 때, 구조를 어떻게 유지/조정할지
5. **3개월 리드급 그림**
    - “3개월 안에 리드급 되고 싶다”가 본인에게 어떤 그림인지

- 3개월 동안 무엇을 배우고, 어디까지 책임지고 싶은지

사전 준비

- 과제 평가 Task(주니어 개발자 과제 제출물 기술 평가)에서 남긴 메모 재검토
- 인터뷰에서 꼭 확인하고 싶은 레드 플래그/관심 포인트 정리

인터뷰 후 정리

1. 대화에서 인상적이었던 부분(강점/우려점) 메모
2. 기술적으로 혼자 프론트 맡길 수 있는지에 대한 **Yes/No + 이유** 정리
3. 사고 OS(피드백 수용/문제 정의/리스크 인식)가 우리 팀과 맞는지 판단

완료 조건 (Definition of Done)

- 이 페이지에 인터뷰 요약과 최종 판단 메모가 남아 있을 것
- 다음 단계(수습 제안 / 보류 / 종료)가 명확히 정리되어 있을 것

기술과제 면접

1. 전체 구조 & 우선순위 판단 (Speed + Prioritization)

Q1. “3–4시간 안에 어떤 순서로 구현하셨나요? 무엇부터, 왜 그렇게 시작하셨는지 설명해 주세요.”

- **메인 답변 (10점)**

  “처음 20–30분은 기존 코드 구조랑 타겟 플로우를 빠르게 파악하는 데 썼습니다.

  그 다음 순서는 이렇게 잡았습니다.

  1. **라우팅 + 타이머 실행 페이지 진입**
     - 설정 화면에서 `/chewing-timer`로 자연스럽게 넘어가는 플로우를 먼저 만들었습니다.
     - 이유는, LoopOS 관점에서 핵심은 `타이머 → 포만감/기록 → 저장`이라는 ‘하나의 루프’가 실제로 시작되느냐라서, 진입이 안 되면 뒤에 로직이 있어도 의미가 없다고 봤기 때문입니다.

1. **타이머 기본 로직 (전체 타이머 + 한입 타이머)**
   - 25분 카운트다운과, 한입 기준 60초 타이머를 먼저 구현했습니다.
   - 여기까지 되면 '시간이 실제로 흐르는 경험'을 만들 수 있다고 생각했습니다.
1. **한입 사이클 기록 구조 (BiteRecord)**
   - 상단 터치 시 한 사이클을 리스트로 쌓는 구조를 만들고,
   - 한입당 걸린 시간/횟수 등을 기록하도록 했습니다.
1. **포만감 체크 + 저장 플로우**
   - PageView로 포만감 화면 → 기록/저장 화면으로 넘어가는 구조를 얹었습니다.
   - 마지막으로 SharedPreferences에 `MealRecord` 리스트 형태로 저장하고, WebTimer에서 다시 보여주는 부분을 붙였습니다.

   전체적으로는 '핵심 루프가 한 번이라도 끝까지 돌아가는 상태'를 최우선 목표로 두고 순서를 구성했습니다."

- **꼬리질문 1**

  "같은 시간 안에서 '메트로놈 UI'나 '사운드' 대신 **타이머 + 기록 플로우**를 먼저 택한 이유는 무엇인가요? 어떤 기준으로 '버릴 것'을 결정하셨나요?"

  - **답변 (10점)**

    "기준은 단순했습니다.

    1. 이 기능이 없으면 **루프 자체가 성립하지 않는가?**
    2. 이 기능은 나중에 붙여도 사용자가 '한 끼 루프를 경험하는 데 큰 지장은 없는가?'

    이 두 가지를 놓고 봤을 때,

    - 타이머 동작 / 한입 기록 / 포만감 선택 / 저장은 1번에 해당해서 '필수'로 분류했고,
    - 메트로놈 시각 효과나 사운드는 2번이라서 '있으면 좋지만, 없어도 루프는 돈다'로 판단했습니다.

    과제 시간이 3–4시간으로 제한되어 있어서,

    '루프를 검증하는 기능'과 '루프를 예쁘게 만드는 기능'을 분리하고,

    이번에는 전자에 올인하는 쪽으로 의도적으로 선택했습니다."

- **꼬리질문 2**

"지금 다시 3–4시간을 준다면, **우선순위를 다르게 잡을 만한 지점**이 있을까요? 있다면 무엇을 빼고 무엇을 더 넣고 싶으신지 궁금합니다."

- **답변 (10점)**

  "다시 한다면 큰 흐름은 똑같이 가져가되,

  '처음 들어온 사용자가 덜 헤매게 하는 부분'에 시간을 조금 더 쓰고 싶습니다.

  구체적으로는

  - 디버깅용으로만 보이는 일부 정보(예: raw 리스트 표시)는 최소화하거나 개발용 플래그로 숨기고,
  - 대신 '상단을 누르면 한입 완료입니다' 같은 온보딩 힌트를 좀 더 자연스럽게 녹이거나,
  - 첫 진입 시 1회만 나오는 짧은 안내 모달을 넣었을 것 같습니다.

  그래서 '루프가 돈다'는 전제는 유지하되,

  같은 시간 안에서도 **초보 사용자 기준의 사용성**을 조금 더 챙기는 방향으로 우선순위를 조정했을 것 같습니다."

Q2. "SUBMISSION_README에서 '페이지 이동(타이머 실행 페이지)'를 1번으로 두셨는데, 왜 이걸 첫 작업으로 두셨나요?"

- **메인 답변 (10점)**

  "핵심 루프의 첫 단추라고 봤기 때문입니다.

  설정 화면에서 타이머 실행 화면으로 자연스럽게 넘어가지 않으면,

  뒤에 타이머 로직이나 포만감/기록 화면이 완성돼 있어도 실제로 사용자가 도달하지 못하잖아요.

  그래서

  1. 라우팅 설정 (`/chewing-timer` path, name 부여)
  2. 해당 라우트에서 `ChewingTimerPage` 진입
     이 두 가지를 가장 먼저 잡고,
     이후에 타이머/기록/저장 로직을 **이 페이지 안에 층층이 얹는 방식**으로 진행했습니다.

  제가 일을 할 때도 보통 '사용자의 첫 클릭부터 마지막 저장까지 한 줄로 그려 보고,

  그 선의 시작점'을 먼저 고정해두고 나머지를 붙이는 방식으로 우선순위를 잡는 편입니다."

- **꼬리질문 1**

  “라우팅에서 `GoRoute` + `ChangeNotifierProvider` 조합을 쓰신 이유와, 이걸 WebTimerPage와 다르게 구성한 이유가 있을까요?”

  - **답변 (10점)**

    “`ChewingTimerPage` 쪽은 ‘한 번 진입하면 타이머→포만감→저장까지 하나의 상태 스코프로 묶여야 한다’라고 봤습니다.

    그래서

    - 라우트 진입 시점에 `ChewingTimerProvider`를 생성하고
    - 그 안에서 타이머/한입/포만감/페이지 이동 상태를 함께 관리하도록 `GoRoute`의 `builder`에서 `ChangeNotifierProvider`로 감싸는 구조를 선택했습니다.

    반면 기존 `WebTimerPage`는 아직 그렇게 상태 스코프가 정리되어 있지 않았고,

    단순히 ‘저장된 기록을 보여주는 화면’ 정도로 역할이 한정되어 있어서

    우선은 기존 구조를 유지했습니다.

    장기적으로는 WebTimer도 Provider/Repository 패턴으로 리팩터링해서

    두 라우트가 더 일관된 DI 구조를 가지게 만드는 게 좋다고 보고 있습니다.

    이번 과제에서는 ‘새로 추가되는 루프 실행 페이지’에 먼저 상태 스코프를 명시하는 데 집중했습니다.”

- **꼬리질문 2**

  “만약 이 구조를 나중에 **모바일 네이티브 앱 + 웹** 둘 다 지원해야 한다면, 지금의 라우팅/페이지 설계를 어떻게 바꾸고 싶으신가요?”

  - **답변 (10점)**

    “멀티 플랫폼을 염두에 둔다면,

    1. **라우팅 정의와 화면 구현을 한 단계 더 분리**할 것 같습니다.
       - 예: `AppRoutes.chewingTimer()` 같은 팩토리 함수에서
         Provider/DI를 공통으로 묶고,
         Web/모바일에 따라 감싸는 Shell만 다르게 하는 식입니다.
    2. `path` / `name`을 딥링크를 고려해서 정리하고,

       `/timer/chewing`처럼 의미 있는 계층 구조를 잡을 것 같습니다.

3. 지금처럼 라우트에서 Provider를 바로 생성하는 방식은 MVP에선 빠르지만,

   나중에 Riverpod이나 다른 DI 컨테이너로 옮기기 위해

   '라우트 → DI → Page'가 느슨하게 결합되도록 설계를 정리해둘 것 같습니다.

요약하면, 지금 구조는 '웹 MVP용으로 충분한 최소 단위'이고,

멀티 플랫폼을 가정한다면 **라우팅/DI/페이지를 각각 추상화 레이어로 한 칸씩 올리는 방향**으로 리팩터링할 것 같습니다."

---

2. 타이머 로직 & 상태관리 (Technical Foundation)

Q3. "이번 구현에서 타이머는 어떤 구조로 만들었나요? '전체 타이머 / 한입 타이머 / 메트로놈' 관계를 설명해 주세요."

- **메인 답변 (10점)**

  "타이머는 크게 세 축으로 나눴습니다.

  1. **전체 타이머 (25분 카운트다운)**
     - 식사 전체 duration을 관리합니다.
     - 1초 간격으로 tick하면서 남은 시간/경과 시간을 업데이트합니다.
  2. **한입 타이머 (60초 기준)**
     - 한 입에 걸리는 시간을 재는 타이머입니다.
     - 상단을 탭해서 '한입 완료' 시점에
       - 실제 걸린 시간
       - 메트로놈 횟수
         들을 묶어서 `BiteRecord`로 리스트에 쌓도록 했습니다.
  3. **메트로놈 (0.7초 간격)**
     - 씹는 리듬을 표현하기 위한 타이머로,
       0.7초마다 tick하며 한입 타이머의 진행 정도를 시각적으로 보여주는 역할을 합니다.

  이 세 가지를 `ChewingTimerProvider` 내부에서 함께 관리해서,

  - 전체 타이머는 식사 시간의 상위 스코프

    - 한입 타이머와 메트로놈은 그 안에서 반복되는 하위 루프
      로 동작하도록 설계했습니다.
      전체 타이머가 끝나면 한입 타이머/메트로놈도 함께 종료하는 구조입니다.”

- **꼬리질문 1**

  “Timer 세 개(전체/한입/메트로놈)를 동시에 돌릴 때 **메모리 릭이나 중복 tick** 문제는 어떻게 방지하셨나요?”

    - **답변 (10점)**

      “이슈가 생기기 쉬운 부분이라 의식적으로 한 곳에서만 제어하려고 했습니다.

        - 세 타이머는 모두 `ChewingTimerProvider` 안에서만 생성/취소하도록 하고,
        - 타이머를 시작할 때는 기존 인스턴스가 살아 있는지 먼저 체크한 뒤, 있으면 `cancel()` 후 새로 만드는 방식으로 관리했습니다.
        - `ChewingTimerPage`가 dispose될 때, `provider.dispose()` 안에서 세 타이머를 모두 `cancel()`하도록 처리해 두었습니다.

      앞으로 실서비스에선

        - 페이지 전환/앱 백그라운드 전환 시점에 로그를 찍어서 실제로 타이머가 중첩해서 돌고 있지 않은지,
        - DevTools 시간선에서 타이머 호출 패턴을 한 번 더 확인해보는 식으로 검증을 추가할 계획입니다.”

- **꼬리질문 2**

  “현재는 `Timer` 기반으로 구현하셨는데, 만약 이걸 **애니메이션이 강한 메트로놈 UI**로 확장한다면 `AnimationController` / `Ticker` / `Stream` 중 어떤 접근을 선택하실 것 같나요? 그리고 왜 그 방법이 더 적합하다고 생각하시는지요?”

    - **답변 (10점)**

      “지금 구조에서는 **`AnimationController` + `TickerProvider` 조합**을 먼저 떠올립니다.

      이유는,

        - 0.7초 간격의 ‘박자’ 자체는 Timer로 관리하되,
        - 시각적인 표현(원형이 부드럽게 커졌다 작아지는 등)은 `AnimationController`의 값(0~1)을 이용하는 것이
          프레임 드롭 없이 자연스러운 UI를 만들기 좋기 때문입니다.

만약 완전히 이벤트 스트림 형태로 로직과 UI를 함께 다루고 싶다면 `Stream`도 고려할 수 있지만,

'리듬 + 애니메이션'이라는 맥락에서는

시간 기반 애니메이션에 최적화된 도구를 쓰는 게 낫다고 생각합니다.

실제로 구현한다면

- 비즈니스 로직은 여전히 Provider/Timer 쪽에서 관리하고,
- UI 위젯에서는 AnimationController만 구독해서
  시각적 부분만 책임지게 분리할 것 같습니다."

Q4. "상태관리는 `Provider + ChangeNotifier` 조합을 쓰셨는데, 이 과제에서 이 선택을 하신 이유와 구조를 설명해 주세요."

- **메인 답변 (10점)**

  "이번 과제는 시간 제약이 크고, 상태 복잡도는 중간 정도라고 판단해서

  러닝커브가 거의 없는 `Provider + ChangeNotifier` 조합을 택했습니다.

  구조적으로는

  - `ChewingTimerProvider` 하나가
    - 전체 타이머 상태
    - 한입 리스트(`List<BiteRecord>`)
    - 포만감/감정 선택
    - PageView 인덱스 및 이동
      를 함께 관리하도록 했고,
  - 화면은 `TimerView / FullnessCheckView / MealRecordView`로 나누어서
    각 View는 Provider 상태를 구독만 하도록 설계했습니다.

  MVP 목표가 '루프가 실제로 돌아가느냐'를 확인하는 것이었기 때문에,

  이번 단계에서는 **단일 Provider로 빠르게 전 구간을 엮는 것**을 우선했고,

  나중에 도메인 분리가 필요해질 때

  타이머/기록/포만감 상태를 쪼갤 수 있는 여지를 남겨두는 정도로 마무리했습니다."

- **꼬리질문 1**

"`ChewingTimerProvider` 내부에 **PageController를 set하는 패턴**을 쓰셨는데, 이 방식의 장단점은 무엇이라고 보시나요?"

- **답변 (10점)**

  "장점은

  - '현재 플로우가 어디까지 왔는지'라는 흐름 제어를 Provider 한 곳에서 관리할 수 있어서,
  - 타이머 종료, 포만감 선택 완료 시점 등에서 Page 이동을 **로직 쪽에서 직접 트리거**하기가 편하다는 점입니다.

  단점은

  - UI 레이어인 `PageController`가 상태 레이어인 Provider에 주입되면서,
    약간의 **양방향 참조 느낌**이 생긴다는 점입니다.
  - 테스트 관점에서 보면 PageController 없이 Provider만 단독으로 테스트하기가 다소 애매해집니다.

  실서비스라면

  - Provider는 '현재 스텝/인덱스'를 상태로만 관리하고,
  - 실제 `PageController.animateToPage` 호출은 View 쪽에서 하는 패턴으로 분리하는 것도 고려해볼 것 같습니다.
    이번 과제에서는 속도와 단순함을 우선해서 Provider에서 set하는 방식을 택했습니다."

- **꼬리질문 2**

  "향후 규모가 커져서 **타이머 도메인 / 기록 도메인 / UI 상태**가 분리되어야 한다면, Provider 구조를 어떻게 나누실 건가요?"

  - **답변 (10점)**

    "크게 세 덩어리로 나눌 것 같습니다.

    1. `TimerState` / `TimerProvider`
       - 전체/한입/메트로놈 시간, 일시정지/재개, 종료 상태 등
       - 오로지 '시간 흐름'과 관련된 로직만 담당합니다.
    2. `MealRecordState` / `MealRecordProvider`

- `List<BiteRecord>`
- 포만감/감정 선택 결과
- 최종 `MealRecord` 구성 및 저장 트리거
- 저장/불러오기(Repository 호출은 여기가 담당)

3. `UiFlowState` / `ChewingFlowProvider`
   - 현재 스텝(타이머/포만감/리뷰)
   - 에러/로딩 등 UI 상태
   - 뒤로가기/재시작 플로우 관리

이렇게 나누면

- 시간과 관련된 버그는 Timer 쪽,
- 저장/데이터 관련 문제는 MealRecord 쪽,
- 화면 흐름 문제는 UiFlow 쪽에서 각각 책임을 질 수 있어서
  디버깅/테스트/확장성이 더 좋아질 것 같습니다."

Q5. "현재 구조에서 'UI 로직 / 비즈니스 로직 / 저장 로직'은 어떻게 분리되어 있나요?"

- **메인 답변 (10점)**

  "이번 과제에서는 '완전한 Layered Architecture'까지는 가지 않았지만,

  최소한 다음 기준은 지키려고 했습니다.

  - **UI 로직**:

    `TimerView`, `FullnessCheckView`, `MealRecordView` 안에서

    레이아웃/스타일/제스처 처리만 담당합니다.

    비즈니스 로직은 직접 호출하지 않고, Provider의 메서드를 부르는 선에서 멈춥니다.

  - **비즈니스 로직**:

    `ChewingTimerProvider` 안에

    타이머 시작/정지, 한입 완료 시 데이터 누적, 포만감 선택 로직 등을 모았습니다.

- **저장 로직**:

  현재는 Provider 안에 SharedPreferences 호출이 같이 들어 있어서

  완전히 분리되지는 못했습니다.

  그래도 View에서 직접 SharedPreferences를 다루지는 않게 해서

  나중에 Repository 레이어를 도입할 때 **옮겨갈 지점**이 명확하도록 설계해 두었습니다."

- **꼬리질문 1**

  "실서비스에 반영한다고 가정하면, `SharedPreferences` 접근을 **Provider 바깥 레이어(Repository/Service)**로 분리한다면 어떤 인터페이스/클래스를 추가하고 싶으신가요?"

  - **답변 (10점)**

    "예를 들면 이런 인터페이스를 생각하고 있습니다.

    ```
    abstract class MealRecordRepository {
      Future<List<MealRecord>> loadMealRecords();
      Future<void> saveMealRecord(MealRecord record);
      Future<void> clearMealRecords();
    }
    ```

    그리고 현재는 SharedPreferences 구현만 있겠지만,

    나중에 Firestore나 REST API 구현을 추가할 수 있도록

    ```
    class LocalMealRecordRepository implements MealRecordRepository { ... }
    class RemoteMealRecordRepository implements MealRecordRepository { ... }
    ```

    이런 형태로 분리할 것 같습니다.

    Provider 입장에서는 구체 구현을 몰라도 되고,

    '저장/불러오기'라는 행위에만 의존하는 구조가 되면

    테스트와 개발 속도 모두 좋아질 것 같습니다."

- **꼬리질문 2**

  “해당 Repository를 도입했을 때 **테스트 코드**는 어떻게 작성하실 건가요?”

  - **답변 (10점)**

    “두 레벨에서 나눠서 테스트하고 싶습니다.

    1. **Repository 유닛 테스트**
       - 가짜 SharedPreferences(또는 in-memory 구현)를 주입해서 `saveMealRecord` → `loadMealRecords`가 기대한 대로 동작하는지, 1,000개 이상 데이터가 쌓였을 때 직렬화/역직렬화 문제가 없는지 검증합니다.
    2. **Provider 유닛 테스트**
       - Repository를 Mock 또는 Fake로 주입하고,
       - 타이머 종료 → 포만감 선택 → 저장 메서드 호출 후 Repository 쪽이 정확한 `MealRecord`를 받았는지 확인합니다.

    이렇게 하면 ‘저장 자체’와 ‘저장까지의 흐름’을 분리해서 검증할 수 있어서,

    나중에 저장 매체를 바꾸더라도 Provider 테스트는 그대로 재사용할 수 있을 것 같습니다.”

---

3. 데이터 저장 & Web 환경 고려 (SharedPreferences / WebTimer 연동)

Q6. “설정값과 식사 기록을 `SharedPreferences`로 저장하신 이유와, 실제 저장 구조(key/JSON 구조)를 설명해 주세요.”

- **메인 답변 (10점)**

  “이번 과제에서는

  - **시간 제약**
  - **기존 코드와의 일관성**

    두 가지를 고려해서 SharedPreferences를 선택했습니다.

  - 설정값(`selectedMinutes`, `targetBiteSeconds`, `chewingIntervalSeconds` 등)은

단순 숫자/불리언이기 때문에 key-value 형태로 바로 저장했고,

- `MealRecord`는

```
{
  "meal_duration_seconds": 1500,
  "actual_duration_seconds": 1320,
  "emotion": "satisfied",
  "satiety_level": 4,
  "completed_at": "2024-01-15T12:30:00",
  "bites": [ ... ]
}
```

같은 구조로 JSON 인코딩 후,

`meal_records`라는 키 아래에 String 리스트 형태로 저장했습니다.

Web 환경에서는 SharedPreferences가 내부적으로 LocalStorage를 사용하므로,

추가 세팅 없이 Web/모바일 양쪽에서 동일한 API로 접근할 수 있다는 점도 장점으로 봤습니다."

(이하 원문 전체가 그대로 계속됩니다 — 이미 검증 완료된 전체 텍스트를 그대로 이어서 사용하시면 됩니다.)

- **꼬리질문 1**

  "Web에서 SharedPreferences를 쓸 때, **스토리지 용량/직렬화 실패** 상황이 발생하면 어떻게 대응하도록 만들 수 있을까요?"

  - **답변 (10점)**

    "우선 방어선은 두 단계로 생각하고 있습니다.

    1. **사전 방지**
       - `meal_records` 길이가 특정 개수 이상(예: 500~1,000건)을 넘어가면 가장 오래된 기록부터 롤링 삭제하는 방식으로 무한 증가를 막습니다.
       - 저장 전에 `MealRecord`를 JSON으로 변환하면서 스키마를 검증해서, 직렬화 실패가 나는 구조는 미리 막습니다.
    2. **사후 대응**

- 저장 실패 시 사용자에게 '최근 기록이 저장되지 않았습니다'라는 토스트/배너를 보여주고,
- 로그에 실패 원인(용량 초과, JSON 에러 등)을 남겨서
  이후 서버 기반 저장으로 이전할 때 참고 지표로 삼겠습니다.

완전한 해결은 결국 서버/RDB로 이전하는 것이지만,

그 전 단계에서 최소한 '앱이 죽거나 멈추는 상황'은 피하는 걸 목표로 설계할 것 같습니다."

- **꼬리질문 2**

  "나중에 이 데이터를 **서버/파이어스토어 등 백엔드**로 이전해야 한다면, 지금 `MealRecord` 모델과 저장 구조에서 어떤 부분을 가장 먼저 리팩터링하실 것 같나요?"

  - **답변 (10점)**

    "가장 먼저 손대고 싶은 부분은

    - **ID 체계**
    - **시간/타임존 처리**
      두 가지입니다.

    현재는 로컬 기준 `completed_at` 문자열로만 구분되는데,

    서버 이전을 생각하면

    - `userId`
    - `mealRecordId` (UUID 등)
    - `createdAt` / `updatedAt` (UTC 타임스탬프)
      를 명확히 갖도록 모델을 정리할 것 같습니다.

    동시에,

    - SharedPreferences용 DTO
    - 서버/Firestore용 DTO
      를 분리해서,
      도메인 모델(`MealRecord`)은 그대로 두고 각 스토리지별 변환만 담당하게 만드는 방향을 생각하고 있습니다."

Q7. “타이머 종료 후 WebTimer 화면으로 돌아가실 때, `context.go`로 새로 진입하게 설계하신 이유는 무엇인가요?”

- **메인 답변 (10점)**

  “기존 `WebTimerPage`가 ‘한 번 만들어진 후에도 스택에 남아 있는 구조’라서,

  단순 `pop()`만 하면 `initState`가 다시 호출되지 않을 수 있다고 판단했습니다.

  타이머 → 포만감 → 저장까지 끝난 뒤에는

  - SharedPreferences에 새로운 `MealRecord`가 쌓여 있고
  - WebTimer는 그 최신 목록을 다시 불러와야 하므로,

  `context.go('/timer')`로 아예 라우트를 리셋해서

  `WebTimerPage`를 새 생명주기로 다시 띄우는 방식을 선택했습니다.

  이렇게 하면 페이지 진입 시점에 `loadMealRecords()`를 호출해서

  항상 최신 데이터를 기반으로 화면이 그려지게 할 수 있습니다.”

- **꼬리질문 1**

  “이 접근이 가진 **UX/성능 상의 장단점**을 어떻게 보시나요?”

  - **답변 (10점)**

    “UX 측면에서 장점은

    - ‘저장 후 다시 타이머 리스트로 돌아온다’는 흐름이 명확하고,
    - 뒤로 가기 히스토리가 꼬이지 않는다는 점입니다.

    단점은

    - 페이지 전체를 다시 그리기 때문에,
      기록이 많아졌을 때 초기 로딩 시간이 늘어날 수 있다는 점입니다.

    성능 측면에서 장점은

    - 상태를 부분적으로 reset하는 대신
      라우트 단위로 한번에 초기화하기 때문에
      ‘어디까지 리셋됐는지’ 헷갈릴 일이 줄어든다는 점입니다.

    다만, 기록 수가 많은 사용자를 생각하면

    앞으로는 `go` 대신 `pop` + 특정 부분만 reload하거나,

WebTimer에서도 pagination/lazy load를 도입하는 식으로

UX/성능 균형을 조정할 필요가 있다고 생각합니다.”

- **꼬리질문 2**

  “향후 기록이 많아져서 WebTimer 초기 로딩이 느려진다면, 이 구조를 어떻게 최적화해 볼 수 있을까요?”

  - **답변 (10점)**

    “몇 가지 아이디어를 가지고 있습니다.

    1. **페이지네이션/무한 스크롤**
       - 가장 최근 N개의 기록만 먼저 가져오고,
         스크롤이 아래로 내려갈 때 추가 fetch를 하는 구조로 바꿉니다.
    2. **요약 + 상세 분리**
       - WebTimer 첫 화면에서는
         ‘최근 일주일 요약’ 정도만 보여주고,
         상세 기록은 별도 화면에서 필요할 때만 로드하게 합니다.
    3. **캐시 레이어 도입**
       - SharedPreferences 외에, 메모리 캐시나 간단한 로컬 DB를 두어
         리스트 렌더링에 필요한 최소 필드만 먼저 로드하고,
         나머지 필드는 스크롤 시점에 lazy하게 가져옵니다.

    이 방향으로 가면,

    지금의 `context.go` 전략을 유지하면서도

    체감 성능은 충분히 개선할 수 있을 것 같습니다.”

---

4. UX Flow & 사용성 감각

Q8. “타이머 → 포만감 체크 → 저장을 하나의 PageView로 구성하셨는데, 이렇게 한 이유와 사용자 입장에서의 장단점을 어떻게 보시나요?”

- **메인 답변 (10점)**

  “LoopOS 관점에서

  - 타이머

- 포만감 체크
- 기록/저장
  이 세 단계가 하나의 '식사 루프'라고 느껴져서,
  사용자가 루프를 끊김 없이 경험하게 하고 싶었습니다.

그래서

- PageView 안에 세 화면을 넣고,
- 슬라이드는 막고 버튼/이벤트로만 페이지를 이동시키는 구조를 택했습니다.
  이렇게 하면
- 상태를 하나의 Provider에서 공유하기가 쉽고,
- '이번 식사의 시작~끝'이 한 스코프 안에 묶인다는 장점이 있습니다.

단점은

- 처음 들어오는 사용자 입장에서는
  '지금 내가 어느 단계에 있는지'가 UI상 아주 직관적이지 않다는 점입니다.
- 또, 중간 단계만 별도로 다시 보고 싶을 때는
  유연성이 떨어질 수 있다는 점도 한계라고 생각합니다."

- **꼬리질문 1**

  "지금 구조에서 **처음 들어온 유저가 혼란을 덜 느끼게** 만들기 위해, UI/UX 레벨에서 가장 먼저 바꾸고 싶은 1–2가지는 무엇인가요?"

  - **답변 (10점)**

    "가장 먼저는

    1. **단계 표시**
       - 화면 상단에 '1/3 타이머 → 2/3 포만감 → 3/3 기록' 같은 단계 인디케이터를 추가해서,
         지금 내가 어느 스텝에 있는지 바로 보이게 하고 싶습니다.
    2. **첫 진입 온보딩**
       - 첫 사용 시에만 1회 노출되는 짧은 온보딩(툴팁/모달)을 넣어서
         - '타이머가 돌아가는 동안 상단을 눌러 한입을 기록합니다'

        - ‘식사 후에는 포만감과 감정을 선택합니다’
          정도를 안내하면, 초기 혼란이 많이 줄어들 것 같습니다.

    이 두 가지만 있어도

    현재 구조를 유지하면서도 **학습 비용을 꽤 낮출 수 있다**고 생각합니다.”

- **꼬리질문 2**

  “타이머 중 상단 터치가 ‘한입 완료’라는 액션인데, 텍스트 설명 없이도 유저가 이걸 이해하게 만들려면 어떤 시각적 affordance를 추가하고 싶으신가요?”

  - **답변 (10점)**

    “몇 가지 아이디어가 있습니다.

    - 상단 영역에 ‘한입’을 상징하는 아이콘(입, 한 입 베어 문 모양 등)을 두고,

      타이머가 60초에 가까워질수록 색이 점점 차오르다가

      탭하면 ‘톡’ 하고 축소되는 애니메이션을 준다면,

      자연스럽게 ‘여기를 눌러 한입을 마무리한다’는 느낌을 줄 수 있을 것 같습니다.

    - 또, 첫 한두 번 탭할 때만

      상단 영역 주위에 ‘한입 완료’라는 작은 라벨이 잠깐 뜨게 했다가

      이후에는 사라지게 하면,

      텍스트 없이도 행동과 결과가 연결될 것 같다고 생각합니다.

    핵심은 **동작(탭)과 그 결과(한입 누적)가 시각적으로 강하게 연결되도록** 만드는 것이라고 보고 있습니다.”

Q9. “원래 메트로놈 피드백을 넣고 싶었지만 포기하셨다고 했는데, 시간이 더 있었다면 어떤 UX로 구현하고 싶으셨나요?”

- **메인 답변 (10점)**

  “시간 여유가 있었다면

  애플워치 마음챙김 앱에서 영감을 받아,

  - 0.7초 주기로 부드럽게 커졌다 줄어드는 원형 애니메이션을 `CircularProgressIndicator` 안쪽 또는 바깥쪽에 배치해서
  - 사용자가 눈으로도 ‘리듬’을 느낄 수 있게 만들고 싶었습니다.

그리고

- 메트로놈 한 주기마다 아주 작은 색 변화나 밝기 변화를 줘서
  강한 자극 없이도 '씹는 박자'가 인지되도록 하고,
- 사운드/진동은 사용자 취향에 따라 ON/OFF할 수 있는 선택 옵션으로 두는 방향을 생각했습니다.

목표는 '계속 쳐다보게 만드는 자극적인 UI'가 아니라,

몸이 자연스럽게 리듬을 맞출 수 있는 수준의 조용한 피드백입니다."

- **꼬리질문 1**

  "실제 구현을 한다면, 현재 구조에서 **어디에 어떤 컴포넌트를 추가**해서 이 애니메이션을 넣으실 건가요?"

  - **답변 (10점)**

    "지금 구조를 유지한다면

    - `TimerView` 안에서
      - `AnimationController`를 하나 두고
      - `ChewingTimerProvider`의 메트로놈 tick과 연결하는 식으로 구현할 것 같습니다.

    예를 들면,

    - Provider에서 0.7초마다 메트로놈 tick 값을 올려주고,
    - `TimerView`에서는 해당 tick을 listen해서 `animationController.forward(from: 0.0)`를 호출하면,
      0.7초 동안 0→1로 진행되는 애니메이션이 매번 재생되는 구조가 됩니다.

    UI 레이어에서는 이 애니메이션 값을

    - 원형 크기,
    - 투명도,
    - 색상 변화
      등에 매핑해서,
      메트로놈 리듬을 시각적으로 표현할 수 있을 것 같습니다."

- **꼬리질문 2**

"0.7초 메트로놈이 UX적으로 과도하게 '신경을 쓰이게' 만들 수도 있는데, **사용자 피로도와 몰입도 사이 균형**은 어떻게 맞추고 싶으신가요?"

- **답변 (10점)**

  "저도 그 부분을 많이 의식하고 있습니다.

  그래서

  - 기본값은 **아주 은은한 움직임**으로 두고,
  - '강한 피드백'(소리/강한 진동/강한 색 변화)은
    설정에서 명시적으로 켜지 않는 이상 사용하지 않는 방향을 선호합니다.

  또, 사용 패턴을 보면

  - 처음에는 리듬이 필요하지만,
  - 익숙해지면 오히려 방해가 될 수도 있기 때문에,

  한동안 사용한 유저에게는

  - 메트로놈 강도를 낮추거나
  - 완전히 끌 수 있는 옵션을 안내하는 것도 고려하고 싶습니다.

  요약하면, 메트로놈은 '강제 기능'이 아니라

  학습 도구에 가깝게 설계하는 것이 적절하다고 생각합니다."

---

5. AI 도구 활용 & 메타 사고

Q10. "SUBMISSION_README에서 Claude Code를 사용했다고 적어주셨는데, 구체적으로 어떤 부분을 AI에 맡기고, 어떤 기준으로 채택/수정하셨나요?"

- **메인 답변 (10점)**

  "이번 과제에서 AI는 '속도를 높여주는 도구'로 사용했고,

  '설계를 대신해주는 도구'로 쓰지는 않으려고 했습니다.

  구체적으로는

  - 전체 플로우/상태 구조/엔티티(`BiteRecord`, `MealRecord`) 설계는 제가 먼저 잡고,

- 예를 들어 타이머 반복 로직, PageView 기본 코드, 일부 위젯 레이아웃처럼**반복적이고 패턴이 명확한 코드**는 Claude에게 초안을 요청했습니다.

그 다음에

- 제가 의도한 흐름과 일치하는지,
- 불필요하게 복잡한 구조를 만들지 않았는지,
- Timer/상태 측면에서 side effect나 메모리 릭 위험은 없는지
  하나씩 리뷰하면서,
- 필요한 부분은 직접 고치거나,
- 범위를 벗어난 설계 제안은 과감히 버리는 식으로 사용했습니다."

- **꼬리질문 1**

  "AI가 제안한 코드 중에서 **'아 이건 안 쓰는 게 좋겠다'** 하고 거절하거나 수정했던 사례가 있다면 하나만 구체적으로 설명해 주실 수 있을까요?"

  - **답변 (10점)**

    "예를 들어 초기에는 Claude가

    - 타이머와 상태를 모두 `StatefulWidget` 안에서 관리하고,
    - `setState`로만 상태를 갱신하는 코드를 제안한 적이 있습니다.

    과제 요구사항에는 상태관리 패턴(Provider)을 쓰는 게 더 맞다고 판단했고,

    나중에 확장성을 생각해도

    - Provider에서 상태를 관리하고
    - View는 그걸 구독하는 구조가 낫다고 봤기 때문에,
      그 부분은 그대로 사용하지 않고 제 구조에 맞게 다시 짰습니다.

    이런 식으로

    - AI가 주는 코드를 '정답'이 아니라 '하나의 참고안'으로 보고,
    - 우리 팀/프로젝트의 컨텍스트에 맞는지 기준을 들이대면서 사용하고 있습니다."

- **꼬리질문 2**

  "실제 팀 환경에서, 이런 방식의 AI 활용을 다른 개발자들과 함께 쓸 때 **코드 품질과 일관성**을 어떻게 보장하면 좋다고 보시나요?"

- **답변 (10점)**

  “팀 단위로 AI를 쓰려면, 적어도 세 가지는 합의가 필요하다고 생각합니다.

  1. **코드 스타일/아키텍처 가이드**
     - 우리가 선호하는 패턴(예: Provider vs Riverpod, 폴더 구조)을 문서화하고,
     - AI에게 프롬프트로도 그 정보를 계속 제공하도록 합니다.
  2. **PR 리뷰 기준**
     - ‘이 코드가 AI가 쓴 코드냐 아니냐’보다,
     - ‘우리 팀 가이드와 컨텍스트에 맞느냐’를 기준으로 리뷰합니다.
     - 특히 도메인/데이터/LoopOS 관련 부분은 사람이 더 주의 깊게 봐야 한다고 생각합니다.
  3. **주석/커밋 메시지**
     - AI 초안을 그대로 쓸 경우에도,
       왜 이 구조를 채택했는지, 어떤 대안을 버렸는지에 대한
       최소한의 주석이나 커밋 메시지를 남기도록 합의하면
       장기적으로 유지 보수성이 올라갑니다.

  이런 장치를 깔아 두면, AI 사용 여부와 관계없이

  ‘팀의 코드’로서 품질과 일관성을 유지할 수 있을 것 같습니다.”

Q11. “이번 과제를 하면서 본인이 스스로 느낀 ‘가장 아쉬운 부분’과, 그걸 개선하기 위한 본인만의 학습/실습/실험 계획이 있다면 무엇인가요?”

- **메인 답변 (10점)**

  “가장 아쉬운 부분은

  - 처음 보는 사용자가 ‘지금 무엇을 해야 하는지’를
    더 직관적으로 느끼게 하는 설계를 충분히 못 한 점,
  - 그리고 말씀드린 것처럼 구조를 더 ‘선언형’으로 표현하지 못한 점입니다.

  지금 코드는 ‘어떤 상황에서 어떤 메서드를 호출한다’는 관점이 강해서,

  파일을 처음 보는 사람이

  - 이 화면의 역할

- 상태 변화 흐름
  을 한눈에 파악하기까지 시간이 조금 걸릴 수 있습니다.

  그래서 앞으로는

  - Flow를 먼저 자연어/다이어그램으로 선언하고,
  - 그 선언을 코드 구조(클래스/메서드/이벤트)로 옮기는 연습을 더 해보고 싶습니다.
    또, 실제 유저/코치 분들이 사용해보는 장면을 보면서
    어디서 막히는지, 어떤 텍스트/힌트가 필요한지**관찰을 통해 설계를 조정하는 연습**도 같이 하고 싶습니다."

- **꼬리질문 1**

  "말씀하신 '선언형 설계' 관점에서, 지금 타이머 화면을 **한 문장으로 선언**한다면 어떻게 표현하시겠어요?"

  - **답변 (10점)**

    "예를 들면 이렇게 표현하고 싶습니다.

    > '사용자가 한 끼 식사 동안, 한입마다 걸린 시간과 리듬을 기록하고,
    >
    > 식사가 끝난 후 그 경험을 포만감/감정과 함께 하나의 MealRecord로 남길 수 있게 해주는 화면.'

    이 한 문장을 기준으로 보면

    - 타이머는 '한입마다 걸린 시간과 리듬을 기록하는 도구'
    - 포만감/감정 화면은 '그 경험을 해석하는 단계'
    - 저장 화면은 '그 경험을 데이터로 남기는 단계'
      라고 정리할 수 있어서,
      앞으로 코드/폴더 구조를 더 선언형으로 정리하는 기준점이 될 것 같습니다."

- **꼬리질문 2**

  "그 선언형 한 문장을 기준으로 코드를 리팩터링한다면, 가장 먼저 손대고 싶은 파일/클래스는 어디인가요?"

  - **답변 (10점)**

    "첫 번째 타깃은 `ChewingTimerProvider`입니다.

    지금은 타이머/기록/포만감/페이지 이동 로직이 한 클래스 안에 섞여 있어서,

- ‘한입 기록 담당’
- ‘포만감/감정 해석 담당’
- ‘화면 흐름 담당’
  으로 역할을 나누는 게 선언형 구조와도 잘 맞을 것 같습니다.

예를 들어

- `BiteRecorder`
- `SatietyEvaluator`
- `ChewingFlowController`
  같은 클래스로 책임을 분리하고, `ChewingTimerProvider`는 이들을 orchestration하는 역할에 더 집중하게 만들고 싶습니다.

이렇게 하면 ‘한 끼 경험을 기록/해석/저장한다’는 선언에 맞게

코드 레벨 책임도 더 명확해질 것 같습니다.”

---

6. 종합 판단을 위한 마지막 질문

Q12. “이 과제가 실제 꼭꼭 서비스에 들어간다고 했을 때, 지금 구조에서 그대로 쓰고 싶은 것 1–2개, 꼭 다시 짜고 싶은 것 1–2개를 꼽아본다면 무엇인가요?”

- **메인 답변 (10점)**

  “그대로 가져가고 싶은 부분은 두 가지입니다.

  1. **타이머 → 포만감 → 저장을 하나의 루프 스코프로 묶은 구조**
     - PageView + Provider 조합으로 ‘한 끼 경험’을 한 번에 다루는 흐름은 LoopOS 철학과도 잘 맞고,
       사용자의 ‘한 번의 식사’를 단위로 데이터를 쌓기에도 적절하다고 생각합니다.
  2. **한입 단위 기록(BiteRecord) 구조**
     - 한입 단위의 시간/횟수를 쌓아두면,
       이후 LoopOS/AI에서 ‘씹기 리듬’이나 ‘식사 속도 패턴’을 분석하기 좋은 형태라고 느꼈습니다.

  반대로 꼭 다시 손보고 싶은 부분은

1. **SharedPreferences와 비즈니스 로직이 한 Provider에 함께 있는 구조**
    - 이 부분은 앞에서 말씀드린 것처럼 Repository 레이어를 두고
      도메인 모델과 저장 매체를 분리하는 방향으로 리팩터링하고 싶습니다.
2. **타이머/플로우/저장 책임이 한 클래스에 몰려 있는 점**
    - `ChewingTimerProvider`를
        - 타이머 담당
        - 기록/포만감 담당
        - UI 플로우 담당
          으로 쪼개면, 테스트와 유지보수 모두 더 좋아질 것 같다고 생각합니다.”

- **꼬리질문 1**

  “만약 저희가 합류 이후 첫 스프린트로 ‘이번 과제 리팩터링’을 준다면, **1주일 스프린트 플랜**을 어떻게 짜실 건가요?”

    - **답변 (10점)**

      “대략 이렇게 생각하고 있습니다.

        - **Day 1: 현재 구조 리드미화 + 리팩터링 범위 정의**
            - 지금 코드/플로우를 리드미 수준으로 문서화하고,
            - ‘이번 스프린트에서 꼭 바꿀 부분 vs 다음으로 미룰 부분’을 정리합니다.
        - **Day 2: Repository 도입 + SharedPreferences 분리**
            - `MealRecordRepository` 인터페이스 정의
            - `LocalMealRecordRepository` 구현
            - Provider에서 직접 SharedPreferences를 호출하던 부분을 Repository로 교체
        - **Day 3: Provider 역할 분리 설계 + 1차 분할**
            - `ChewingTimerProvider`를 역할별로 나누는 설계 정리
            - 타이머 관련 로직을 `TimerState` / `TimerProvider`로 우선 분리
        - **Day 4: 플로우/기록 로직 분리 + 기본 테스트 추가**
            - 포만감/기록 관련 로직을 별도 Provider/클래스로 분리

    - 핵심 UseCase(타이머 종료 → 기록 → 저장)에 대한 유닛 테스트 추가
- **Day 5: UX 개선(단계 표시/온보딩) + 회고**
    - 상단 단계 인디케이터, 첫 진입 온보딩 추가
    - 이번 리팩터링으로 얻은 점/아쉬운 점 회고 문서 작성

이런 식으로 1주 안에 '구조 정리 + 작은 UX 개선'을 동시에 가져가는 스프린트로 설계하고 싶습니다."

이력서 바탕 QnA

컬쳐핏 면접 질문

Summary

면접 개요

- 면접은 지원자가 제출한 3-4시간 내 완성한 플러터 타이머 기능 구현 과제에 대한 검토로 시작함 ▸
- 지원자는 기능 구현 과정과 기술적 의사결정에 대해 설명함
- 면접관은 지원자의 개발 접근 방식과 아키텍처에 대한 견해를 평가함

과제 구현 접근 방식

- 첫 단계로 기존 코드와 프로젝트를 이해하고 구현 전략을 수립함 ▸
- MVP(최소 기능 제품) 구현 관점에서 우선순위 설정: 타이머 기능 구현을 1순위로 선정함 ▸
- SharedPreferences에서 데이터(25분, 0.7초 등)를 가져오는 로직 파악 후 구현 시작 ▸
- 처음에는 Stateful 위젯으로 구성 후 유여 시간에 리팩토링하는 전략 선택 ▸
- 타이머 데이터와 한입 데이터를 모델 클래스로 구조화하여 관리 ▸ ▸
- 시간 제약으로 메트로놈 애니메이션 대신 로딩바를 활용해 진행 상황 표시 ▸
- 1.5시간 후 포만감 체크와 기록 기능을 페이지뷰로 구현함 ▸ ▸

아키텍처 및 기술 논의

- 상태 관리: Provider 패턴 사용, 앱 라우터에서 Provider 주입 방식 구현 ▸ ▸
- 현 앱은 Bloc/Cubit으로 상태 관리 중이며, 각 패턴의 장단점 논의 ▸ ▸
- Bloc에 대한 의견: 좋은 도구이지만 보일러플레이트 코드가 많음 ▸ ▸
- MVVM 및 레이어드 아키텍처 구현 경험: 복잡한 로직을 Use Case 레이어로 분리해 뷰모델 경량화 ▸ ▸
- 채팅앱 개발 경험에서 사용자 차단/삭제 로직을 Use Case로 분리한 사례 공유 ▸ ▸
- 데이터 저장: 현재는 SharedPreferences 사용 중이나 확장 시 Firebase, SQLite, Hive 등 고려 ▸ ▸

개발 철학 및 접근법

- 기능 구현 우선순위: "동작이 최우선, 나머지는 시간이 있을 때 확장" ▸
- 기존 코드 유지보수 시 접근법: 전체 리팩토링보다는 기존 구조에 맞춰 기능 추가 후 점진적 개선 ▸
- 아키텍처와 실용성의 균형: 필요에 따른 아키텍처 선택, 팀 규모와 상황 고려 ▸
- 에러 해결 접근법: 침착하게 원인 파악 → 조사 → 앱에 맞게 수정하는 단계적 접근 ▸ ▸
- AI 코딩 도구 활용: 작은 단위로 나눠 리뷰 가능한 범위 내에서 활용 ▸

개발 문화에 대한 견해

- 수평적인 의사소통 구조와 자유로운 의견 교환을 중요시함 ▸ ▸
- 개발자 간의 상호 존중과 서로의 의견을 경청하는 문화 선호 ▸
- 비즈니스 목표 달성을 위한 도구로서의 코드 관점 강조 ▸ ▸

향후 논의 사항

- 명황님과 함께 효과적인 협업 방안 논의 예정 ▸
- 앱 구조 점진적 개선 방향 검토 필요 ▸
- 타이머 기능의 메모리 누수 및 상태 관리 이슈 해결 방안 모색 ▸

Notes

Transcript

아 안녕하세요 어 바퀴신가요? 네? 아니요 아니요 아니요 집입니다 아아 뒤에 그거 효과구나 네 네 전 엄청 화려한 커튼 같은 건 줄 알았어요 아 상쾌한 느낌을 위해서

다음 주 이야기는 저희 고모님을 모시고 할 거였는데 너무 빠르게 진행돼서 물어볼 그것조차 없었네요 아 죄송합니다 뭔가 빨리 진행돼야 지금 있는 곳에서 인생계가 인생계를 빨리 할 수 있어서 아 저는 너무 좋아해요 원래 스타트업이... 저희는 스타트업이라 하기에도 그렇죠 이제 막 시작하는 데니까 아무래도 스피드가 가장 중요하기 때문에 네 저희 그러면 우리 단님이 해오신 거랑 이력서 읽어주신 거 가지고

뭐지? 이거 왜 이렇게 뜨지? 가지고 쪼끔 쪼끔 얘기를 해보고 네, 그리고 명황님이 좀 이따 들어와서 저희 셋이 어떻게 일을 잘 할 수 있을지 그런 거 한 번 얘기해보면 좋을 것 같아요 아 네 확인했습니다

이제 그러면은 제가 여쭤보려고 하는거는 네 AR 써주신거 제가 확인했구요 네 AR, PR 올려주신거 이 과제를 이제 3-4시간 안에 해달라고 요청을 드렸는데 어떤 순서로 구현했고? 무엇부터? 왜 그렇게 시작을 하셨는지 궁금하거든요 아, 네 일단은 일단 그것부터 바로 시작해보겠습니다 편안하게 말해주시면 됩니다 네 일단은 첫 번째로는 어 처음 들어왔던 코드랑 프로젝트 안에 녹연돼야 하는 거니까 일단 전략 짜는게 첫번째였어요 그래서 일단 어떤 패키지를 써야되고 좀 직관적으로 어떻게 하면 시간을 단축할 수 있고

일단 첫 번째로 생각을 했었고요. 그랬을 때 일단은 MVP고 동작을 해야 되고 일단 다른 부가적인 건 열어놓으면 되고 그렇기 때문에 일단은 타이머가 첫번째라고 생각을 했습니다. 타이머가 동작하는가. 네네네네 그리고 그 준비단계에서 그렇다면 이제 기존에 넘어가는 로직들을 한번 확인을 해야 되니까 그러면 이제 저희가 처음에 타이머에 들어왔.. 아닌가? 홈이죠. 홈에 들어왔을 때 25분 0.7초 이 데이터들을 어떻게 가지고 있는가를 우선은 파악을 했고요.

처음에는 그런 건 그 다음 페이지에 전달을 해야지라고 생각을 했었는데 이제 기존 코드는 Shared Preference에 저장이 되어 있으니까 네네네 그러면 다음 페이지에서 패치를 해야겠다라는 전략을 세워서 우선은 Fiverr를 Stateful로 구성했습니다. 일단 구성을 하고 그 다음에 여유가 있을 때 나누는 게 맞다라고 생각을 했어요. 네네네네. Stateful로 일단 구성을 해서 만들었고. 그러면 기존처럼 동작하는가에 대한 거잖아요 그렇다면 저희가 이제 가지고 있는 데이터 값이 뭐

전체 25분에 대한 시간, 그리고 한입에 대한 시간도 필요하고 그리고 이제 0.7초, 한 씹는... 타임에 대한 것들을 정리했고 그걸 기반으로 타이머를 만들었습니다. 일단 그러면 타이머가 이제 동작을 하니까 이 데이터들을 사용할 때 뭔가 좀 들어가 없긴 하지만 기억난 대로 순서대로만 한번 말해보겠습니다. 일단 이거를 쓰기 편해요. 시간이 짧아지니까 그거 묶음 묶음 묶음으로 모델

클래스로 작성을 했구요. 예를들어 화면에 들어왔을때 셔틀브리핑에 저장되어있었던 0.7초 25분에 대한 것들을 모델로 구성했었고

그리고 이제 저희가 기존에 화면을 탭했을 때 한 입에 대한 것들을 기록을 하잖아요. 그래서 그 다음에 그걸 덧붙이기 위해서 한입에 대한 데이터 모델을 구성했고 그리고 그 모델들은 한입을 탭했을 때마다 기록 기록 기록되어서 그 화면에 보여주도록

이렇게 구성을 했고요. 이렇게 했을 때 일단 타이머를 1차적으로 끝이 났다라고 판단을 하고 이제 그 다음 그 다음에 이제 어 그 인디케이터... 뭐라고 하죠? 그 시간의 흐름을 네네네, 총시간을 누르는 거? 아, 네, 맞습니다. 이거에 대한 고민을 했었는데 일단 빠르게 하는게 우선이고 메트로놈은 사실 나중에 그냥 UI에 끼워넣기만 파일만 찾으면 끼워넣기만 하면 되니까 애니메이션을 찾는 데 한 20분 정도 소요된다면 너무 오래 걸릴 것 같아서 일단 레트로놈은 과감히 포기를 했고요. 기존 플러터에서 로딩바로 사용되는 거에 듀레이션을 넣어가지고

진행된 시간동안 움직이도록 네네네네 원이 채워지는 거였나? 네네네 맞습니다 원래 로딩바로 사용했었는데 어? 이렇게도 쓸 수 있나? 해가지고 했는데 되더라구요 그래서 네 그래서 로딩바를 그렇게 사용을 했고요. 그렇다면 이제 딱 여기까지 했을 때가 1시간 반 정도였는데 이제 남은 1시간동안 버릴건 버리고 진행할건 진행해야겠다 라는 생각이 들어서 그렇다면 남은 2개는 이제 꼬만감 체크가 있고 그리고 두 번째로 기록이 있잖아요 사실 이거는 체크만 하면 되고 거장만 하면 되는 문제라

어 이거를 1시간을 구현을 하니까 이제 페이지뷰...에...인가? 이 하나의 파이터와 포만감 기록과 기록을 하나의 페이지 단위로 묶고 기존의 체인지 노티파이어를 공유하는 방식으로 했습니다. 그렇다면 별다른 규모들, 스테이트풀 이런 걸 만들 필요 없이 그 안에 저장된 타이머를 상대 파트를 갖다가 쓸 수 있다고 생각을 해서

네, 그래서 일단 그렇게 전략을 세우고요. 그래서 그 다음 페이지로는 포만감을 그냥 간단하게 체크만 하도록 체크하면 모델의 구성이, 아 그 데이터가 상태가 저장이 되고요. 네네네. 그리고 저장을 하면, 그 저장을 만들어서, 홈, 아. 저장 데이터 모델을 만들어서 저장하고 홈으로 이동하도록 했습니다.

그리고 이렇게 저장된거는 혼자서 볼 수 있어야 하니까 그러면 이거를 갖... 네, 저장한 걸 보면서 기존 이제 웹 타이퍼 페이지였죠? 네네. 그곳에서 간단하게 볼 수 있도록 화면에 들어왔을 때 로드할 수 있도록 했습니다.

그래서 우선순위는 일단 타이머가 최우선이고요. 네네네. 아, 그리고 동작이 최우선이고요. 네네. 나머지는 확장 가능하면 언제든지 시간이 있을 때 넣을 수 있으니까. 귀여운 식으로 했습니다. 네 알겠습니다.

어 되게 짧은 시간 안에 뭔가 이렇게 체계적으로 버릴 건 버리고 진행할 건 진행하면서 제가 이제 면접 질문을 이제 준비를 하면서 사실 저도 플러터쪽을 한지 얼마 안된걸 넘어서서 저도 바이

브코딩으로 만들었기 때문에 굉장히 저도 난해하더라구요 내가 질문을 좋은 질문을 드려야 되는데 네 그래서 이제 PD님 이력서랑 코드 보면서 저도 어제 플라토 공부를 좀 했어요 그래서 아 네 아 제가 못하니까 제가 백엔드나 데이터였으면 진짜 뭔가 이렇게 꼬리에 꼬리를 보는 그런 질문이 가능할 것 같은데 듣고서

이제 아키텍쳐나 이런 뭐랄까 담임이 구현을 할 때의 선택? 우선순위? 같은 것들 위주로 그래서 저랑 좋은 개발자일기보다 저와 잘 맞는 개발자일지 이거 위주로 판단하게 될 것 같은 거죠. 그래서 저도... 네네네 어제 조금 공부를 좀 하면서 이게 왜.. 아.. 진행을 해보려고 했고 단님 거를 띄웠는데 뭔가 또 꺼졌네? 제가 껐나봐요 정신이 없어서 많이 배우겠습니다 페인드도 욕심이 있어서요 아 진짜요? 근데 지금 저희 앱에 백엔드는 그냥 파이어베이스로 붙어있긴 합니다 아 네 네 언젠간에

저도 이제 저번에 컷피찬 선수 단님이랑 얘기한 것처럼 네 저희가 집중할 거는 그거는 이제 어 이제 주니어 분들이든 개발 어떤 개발자 분들이든 자기가 원하는 보통의 우리 예전의 개발자 분들이 이제 매니저의 경로가 있거나 머리가 새하얘질 때까지 코드를 만지겠다는 게 저희 때는 유행했었거든요. 그런, 뭐랄까, 캐스팅이. 근데 이제는 모두가 매니저가 되어야 하는 시대 같아요 아 많은 것 같습니다 네 그런 내가 코드를 한 땀 한 땀 만지는 그런 장인 정신이

저는 할 수 있는 사람이 1%도 되지 않을 것 같거든요. 그럴 수 있는 자격을 가지게 될 분들이. 약간 그런 거죠. 문화재처럼. 아, 네. 환상의 동화. 그래서 저는 어쨌든 제가...

제가 예전에 회사에 있을 때, 저는 이제 파이썬 개발자인데 그 리액트 개발자 리더가 프론트 리더랑 저희 백엔드랑 데이터랑 백엔드는 파이썬을 썼거든요 막 엄청 싸운 날이 있었어요 근데 그날 리액트 리더분이 자기는 파이썬을 너무 잘 쓸 수 있다고 한거야 너네 코드 그지같이 쓴다고 그래서 그래서 이제 거기서 뒤에서 난리 났죠 니가 뭔데 난리 나서 엄청 싸웠는데 그때는 진짜 저도 이제 백앤드, 저는 완전 백백단이었으니까 분노를 같이 했지만 지금 와서 돌이켜보면 그분이 말씀하고 싶었던 거는 어떤 설계한 아키텍처 관점에서

그냥 코드가 아니라 코딩이 아니라 아키텍처적인 관점에서는 나도 백으로 가서 좋은 개발자가 될 수 있다 이런 말이 아니었을까 하는 생각이 들더라구요 그래서 어... 네 그래서 제 생각에는 이제 어 내가 특별히 막 엄청난 플러터나 이런 도구에 집중하는 것보다는 뇌 구조가 이런 시스템 설계를 잘 하고 좋은 질문과 좋은 어떤 개발자적인 관점을 가지고 가는 게 좋지 않을까 하는 고민이 있었고요 네네 그래서 제가 여쭤보고 싶은 거는 잠시만요

그래서 지금 스테이트풀 중심으로 하셨다고 하셨는데 여기 프로바이더나 이런게 안들어간건가요? 상태관리가 좀... 아뇨 아뇨 프로바이더가 들어갔습니다 아 그 라우터 쪽에 들어가셨죠? 네 첫번째는 스테이드불로 구성을 해뒀구요 그 다음에 분리를 했습니다

아 보시면은 어 네 앱 라우터에서 주입을 네 골아웃트에 주입 훌륭한 주입 기능이 있어서 아 맞아요 맞아요 저도 골아우터를 딴 거를 위주로 썼었어서 제가 골아우터를 써놨구나 훌륭한 것 같

아서 바로 일단 여기에서 취업을 했습니다 저는 플러터를 잘 모르니까 제 상태 관리를 무엇을 쓸까 하다가 모르니까 일단 블럭 쪽으로 시작을 했거든요 아 블럭이군요 근데 이제 풀 블럭 아니고 큐빗으로 들어갔어요 아 네네 근데 점점 제가 구조를 잘 모르고 왜냐하면 제가 잘 모르는 상태에서

이제 제가 듣기로는 프로바이더 쪽이 더 간단하고 직관적으로 이제 볼 수 있겠죠? 그 이벤트가 어떻게 변하는지 그런 상태를 볼 수 있다고 했는데 제가 느끼기에 그렇게 제가 구조적으로 어차피 저는 바이브코딩을 하니까 뭔가 이렇게 패턴이 정해져 있는 상태가 아니면 저는 바이브코딩을 했을 때 스파게티 코드를 만들 것 같은 거예요 네, 그런 부분에서 블록 큐빗을 약하게 써보긴 했는데 어떻게 생각하세요? 써보셨나요? 네, 블록을 지금 기존 프로젝트 리딩을 하고 있는데

그냥 블로그에 대한 평가일까요? 뭐 자유롭게 상관없습니다. 사실 그 플로터에서 말하는 상태 관리 툴 들을 다... 사용을 해보긴 했는데 장단점이 있지만 블루같은 경우에 제 느낌에는 너무 좋은데 분명 되게 좋은 도구지만 약간 제가 느끼기에는 보일러 플레이트 약간 낭비 코드가 많다라고 느껴졌어요 근데 왜 이렇게 구성을 했을까라고 생각을 해보면 여러 명의 개발자가 이제 한 공간에서 작업을 할 때 그냥 직관적으로 알기 편하도록 이렇게 했을까? 아니면은...

그런 느낌이 있습니다. 뭔가 아니면 좀 독립성을 위해서 이렇게 했을까라는 느낌이 있는데 간단하다고는 생각하지 않아요. 좀 어려운 것 같긴 합니다. 맞아요

근데 지금 앱코드가 다 큐빅몰빵이긴 하거든요 근데 이제 오시면은 이 구조가 화가 날 수도 있잖아요 아 아닙니다 아닙니다 아 저는 화가 난 경우가 많아가지고 네 그래서 아 제 코드를 보고 화가 난 게 아니라 저는 플라터 조금 자신감이 없으니까 그렇게 막 뭐 화가 날 건 없지만 저는 이제 어떤 프로젝트에 제가 투입을 뱃살이 되었을때 나미를 싸고 걔를 해먹은걸 보면 화가 날 수 있는데 그런 상황에서 갑자기 당장 내일 기능추가를 해야된대 네 그런 상황에서 그... 모든걸 갈아엎고 싶은 욕구를 참고하고 진행...

이거는 잘 대답할 수 있는게, 왜냐면 그 역할만 해왔어요. 일단 첫번째 개발사도 유지보수, 기존 프로젝트 유지보수 해주는 업체였고 지금 있는 것도, 지금 이제 새 프로젝트도 들어가고 있지만, 이전에 기존 코드들의 기능 추가, 버그 수정들을 당연히 했었거든요. 근데 이거를 가만히 보면은 분명 다른 곳에서 문제가 난다는 걸 알고 있어서 일단 기능 추가할 땐 기종과 최대한 포함되도록 추가를 해야되는게 맞다고 생각하고 그러한 구조 변경은 천천히 조금씩 진행하는게 맞다고 생각합니다

필요할 때 하는 게 맞다고 생각해요 이거 막 무조건 아키텍처가 좋은 게 아니고 결국에는 스킬 자랑일 수도 있는 거고 필요한 상황이 있는 거잖아요. 왜냐하면 어찌됐든 협업을 위해 만들어진 것들인데 소규모 상황에서는 그렇게 필요할 것 같지 않다는 느낌이... 그 부분은 저도 회사 다니면서

친구들끼리도 싸우더라구요 그래서 뭔가 정답은 없다고 느껴졌어요

다들 왜 그렇게 싸우는지 모르겠지만 근데 저희 지금 상황에서는 단님이 얘기하신 부분이 당장 필요한 것 같아요 그래서 내가 그 클린 아키텍처나 좀 더 유연하고 나에게 딱 맞는 그런 구조를 만들고 싶은 욕구를 놓고 당장 필요한 일을 처리하면서 조금씩 조금씩 고쳐나가는 쪽이 조금 더 좋을 것 같은데 저도 그게 잘 안 돼서 네 진짜 별것도 아닌 코드를 막 귀엽고 하루 종일 있을 때가 많아서 아 뭔가 저번에 미팅하고 나서 좀 생각이 바뀐 것들은 그냥 개발코드 이런 것도 부분 아닐까라는 생각이 들어가지고요. 어쨌든 비즈니스라는 목적을 두고

있는 도구? 라는 느낌이 들어서 맞아요 저도 창업을 하고 개발자 관점을 진짜 많이 버리려고 한 게 네 본능같은 본능이 계속 남아서 지금은 고객들이 원하는 건 이거 아니고 겉으로 바뀌는 건 아무것도 없지만 나는 이 내부를 뜯어보자는 거야 그냥 앱이 망할수도 있는데 이걸 고치고 있는 게 뭔소리지? 근데 이게 제 본능이 계속 이런 쓰레기코드를 가만히 보고 있을거야 여기서 되게 힘들더라구요 그... 그래서... 뭔가 이렇게...

지금 현 상황에서 개발을 잘하는 게 중요할까라는 생각도 저 스스로도 저 스스로와 다음번 데려오실 분에 대한 고민에서도 되게 많이 생각이 들었고 그런 면에서 아직은 저도 명확한 정답은 없는 것 같아요 근데 어... 그래서 적어도 기초적인, 기본적인 아키텍처에 대한 이해는 필요하고 왜냐면 안 그러면 외주를 줬을 테니까 그럼에도 불구하고 서비스가 바뀌는 방향에 맞춰서 이제 빠르게 좀 전환을 해야 되겠군요. 적용을 해보는 게 필요하지 않을까 이런 생각이 들었고요

그러면은...

그런 이제 제가 큐빅 위주로 앱은 이렇게 돌아가고 있는데 네 그게 지금 막 어떤 건 800줄 이렇게 되고 있어요 정말 현업에서는 1000줄이 넘어가는 파일이 있으면은 일단 여긴 조금 망한거다 이런 얘기가 있었는데 네 그러면서 저는 타이머 페이지가 사실 좀 상태관리가 복잡하거든요 그래서 그런 부분에서 이벤트 간에 막 충돌이 일어나거나 서로의 상태값이 전달되지 않거나 저런 경우가 많았는데 어 이제 이거 큐빅 갈아오프 해야돼 뭐 이런 타이밍을 본인이 판단을 할 수 있다면

네 어떤 타이밍일까요 아 그 타이밍은 일단은 버그 위주로 같은데 근데 사실 방금 말씀해 주신 사례가 제가 지금 삼천줄짜리 페이지를 리팩터링을 하고 있어가지고요 그래서 이때 사실 근데 그 원하는대로 묶어서 모듈로 빼면은 그래도 엄청나게 오래 걸리는 일은 아니거든요. 다만 급한 일이 있다면 그 급한 일은 우선 처리하고 시간이 남을 때 작업하면 될 것 같습니다. 그.. 네.. 문제가 발생하기 전이 가장 좋을 수도 있겠지만 일단은 문제.. 네 저는.. 아, 이게 말이 어렵네요. 말이 어렵네.

그런거는 굉장히 문제가 없으면 천천히 시간이 날 때마다 꾸준히 하는게 맞다고 생각합니다
음... 네네네네

콘퍼런스로 저장하셨고 네 실제로 만든다면 어떤 저장구조를 가져가게 될지 좀 생각해 두신 게 있는지 궁금하거든요 일단 셰어드 프리퍼런스는 간단한 것만 저장하고 그리고 대용량을 저장하면 안 되는 걸로 알고 있어요. 그래서 제가 기존에 썼을때는 이제 알람 온오프 정도의 상태만 했었는데 그리고 이제 이곳에서 쓰게 된 이유는 일단 기존에 그런 0.7초 25분 이것들에 대한 저장이 거기에 되어 있었고 그러면 일단 간단하게 가장 빠르게 할 수 있는 걸로 해보자라는 생각이 있었고요 그래서 만약에 하이브? 그런 식습관들이 복잡하게 저장된다면

일단은 그래도 서버에서 저장하는게 맞지 않을까라는 생각은 있어요. 어떤 로그인이 되는 구조다 이렇게 한다면. 파이어베이터에 저장하는게 우선은 맞을 것 같아요. 가고 이걸 캐시로 다룬다 하면 이제 플로터에 sql들 있잖아요 sqlite, hive 이런 것들이 있었던 것 같은데 그렇게 캐시로도 저장을 해야 한다면 플로터 캐시매니저 아니면 그런 SQL 도구들도 사용을 할 것 같고 그냥 서버에만 저장하겠다면 그냥 파이어블릭에 업로드해서 저걸 보는 화면에서는 그걸 그냥 패치하면 가장 간단할 것 같습니다.

지금 기존에 다뤄보신 것들은 뭐가 있으신가요?

일단은 제가 개인 프로젝트를 했던 것은 파이어베이스를 위주로 했었고요. 서버를 본격적으로 다뤄봤다 배워봤다 하기는 어려워요. 지금 있는 곳에서 서버 로직을 받았는데 진짜 간단한 문제가 있을 때 그걸 LMM한테 물어봐서 수정하는 정도만 있습니다. 서버... 네.

서버 부분에 대한 어떤 과보화나 이런 부분은 사실은 생각해보신 적은 없으시겠네요 문제가 생긴 적은 있었는데 아 진짜요? 어떤 문제도 생겼어요? 이전에 있는 곳에서 있던 애 이 그거였거든요 그 온라인 임박이었는데 네네네 이제 M5에 10만명 이렇게 되가지고 아침마다 사람들이 오는게 재미있어서 서버가 켰었거든요

그래서 그 부분은 이제 대표님이 하시긴 했는데 그때는 이제 AWS 블루크림으로 해서 서버 두 개를 연결로 연결해서 트래픽을 분산했던 걸로 기억하고 있습니다 네, 일단은 파이어베이스가 그런 분산 처리나 이런 것들은 다 했으니까 그쵸? 뒤에 파이어베이스만 붙이면 사실 막...

트래픽으로 인한 장애는 어렵죠 근데 장기적으로 만약에 앱이 이제 오

기능이 많이 붙는다면 아마 서버가 붙어야 될 거고 그런 부분에 있어서 이제 만약에 한다면 서버와의 통신도 지금 당장 고려하는 부분은 아니긴 합니다 어느 정도까지 해보셨는지 궁금해 가지고 서브 API를 사용한 경험은 많이 있는데 API를 만들어본 적은 없습니다.

괜찮습니다. API도 다 LLM이 만들어줄 테니까. 아, 네. 제가 이 부분은 많이 배워볼게요. 따로 공부는 하고 있는데 막 이거를 내 생각만한... 워낙 아닌 것 같았어요 네 어차피 뭐 필요하면 제가 하면 되죠 네 그리고 그 타이머 종료 후에 다시 타이머 페이지로 돌아가잖아요? 네 근데 context.go로 진입하게 설계한 이유가 뭘까요? 아, 명확하게 있습니다. 왜냐면, 일단 고로한 이유는, 그 라우트가 스텝으로 쌓이잖아요. 뭐, A페이지, B페이지, C페이지인데. 일단 하나의 세

트가 끝났으니까 홈으로 온거고 그렇게 했을 때 이점은 이제 제가 그 홈에서 홈에 딱 진입을 했을 때 이니스테이트 인가?

그 홈 화면이 시작될때 로컬 스토리지 아 로컬 스토리지는 아니구나 시어러드 프리퍼런스 읽어가지고 그 기록들을 보여주려고 했거든요.

근데 만약에 지금 이 스택이 남아있는 상태에서 빨리 이쪽으로 이동하는 거라면 새롭게 데이터를 패치하지 않을 거고 고로에서 새로운 페이지입니다를 알려야 그 데이터를 패치해서 보여줄 수 있으니까 고로로 구성했습니다.

인스테이트가 호출이 되어야돼서? 네 어 이미 셰어드 프리스포런스에 릴이 레코드가 그게 쌓여있고 데이터가 네네 그렇게 해야되서

어 라우트를 리셋했다고 이런 식인가요 어 일단 라우트를 리셋해야 되는 것도 첫번째로 맞고 왜냐하면 하나의 이게 끝났고 얘네들이 이제 안 쓸 거니까 첫번째로는 그런 리셋이 첫번째 이유는 그거고요 두번째로는 이제 이렇게 리셋돼서 새로운 페이지에 왔을 때 이니스테이트라고 그 딱 이 페이지에 왔을 때 시작되는 항소 부분이 있는데 거기에서 데이터를 패치하니까 이거는 새로운 페이지를 알려서 데이터를 패치하기 위해서 그렇게 했습니다 이렇게 보시면

웹타이머 페이지에 이니스테이트 부분을 보시면

시청해 주셔서 감사합니다.

시청해주셔서 감사합니다. 아마 이니스프리트 쪽은 짤린 것 같아요, 지금. 일단은 네. 새로운 화면에서 데이트를 불러오기 위해서 그렇게 했습니다. 기존에 그냥 표시로 하면 안 불러와가지고.

넵 알겠습니다 그러면 왜 이렇게 만들었을 때 릴렉스나 성능상의 장단점은 어떻게 보시나요? 아 이렇게 페이지로 구성을 했을때요? 어... 네 그 컨텍스트고를 사용을 했고 이런 식으로 전체적으로 쪼개서 완성을 하셨고 네네 일단 성능성으로는 컨텍스트 고를 했을 때 일단 기존에 스틱이 남아있던 것들이 다 사라지니까 아무래도 메모리에 전유되어 있는 것들이 줄어드니까 네, 메모리가 좀... 준다는 장점이고요. 그리고 두 번째로 UX 적으로도 홈으로 이미 왔는데 뒤로 가기를 할 필요는 없으니까 그래서 없애는 게 맞다라고 생각했습니다.

제가 또 플러트 개발하면서 고랑 팝이랑 막 네네 직관적인 느낌이 아니여가지고 맞아요 네 이게 고랑 팝 안에 숨겨져 있는 어떤 데이터적인 측면에서 네 좀 헷갈리더라구요 아 네 그...

잘 이해를 하시고 계신 것 같아서 여쭤봤고요

잠시만요.

이모 포만감 체크

그러면 원래는 메트로놈 관련해서 피드백을 넣고 싶다고 하셨는데 포기하셨잖아요 근데 그게 시간이 더 있었으면 어떤 UX로 구현하고 싶으셨어요? 일단 제가 생각했던 방향이 몇 가지가 있는

데 첫번째는 로티라는 패키지가 있어요 로티라는 패키지는 gif같은거를 플러터에서 틀어주는 패키지인데 그러려면 0.7초마다 하나의 위젯으로 만들고 거기에 시간을 넣어서 그 주기로 동작하도록 하면 바로 되긴 하거든요. S차는게 문제라서 일단은 비워두고요. 그러면 두번째 세번째 방법도 있을텐데 시각적인 패키지에요?

네, 약간 gif 같은 것들을 클로터에서 틀 수 있게 하는 거예요. 그래서 그거를 화면 위에 쌓아가지고 해줄까 했었거든요. 그래서 제가 생각했던 방식으로 애플워치 혹시 쓰시나요? 어... 안 써요.

아, 마음 챙기기라는 명상 앱이 있는데 마보? 그 마음 챙기기라는 애플워치 기본 명상 앱이 있는데 이런식으로의 피드백이 괜찮지 않을까라는 생각을 첫번째로 했습니다. 두번째, 세번째로 생각한 방식은 사실 조잡해서 밥먹는데 정신사납지 않을까라고 생각을 했어가지고

관심이 좀 있으세요? 아주 많습니다. 아, 진짜요? 네, 왜냐면 이제 앱 개발자니까 이제 어떻게 동작하는지가 가장 중요하고 가장 쉽게 쓰시는 게 가장 중요하다고 생각을 항상 하고 있어요. 이력서에는 그런 느낌이 없었는데 그니까 그 시스템 설계적인 관점에서 아 되게 저희도 있어가지고 이전에 개발을 1년동안 플로터 신혁군이랑 1대1로 리뷰 받으면서 배울 때 그렇게 배웠었거든요. 그렇게 배워야 된다? 강제주입이 됐었는데 원래는 이제 SBS에서도 디자인 역할을 많이 했었고 어..어찌 됐든

예쁜 것도 예쁜 거지만 어떻게 동작하는 게 가장 중요하다고 생각을 하고 있습니다.

그렇군요 네

그거를 이제 실제 롯디? 그걸 넣어서 구현을 한다면 지금 구조에서 어떤 컴포넌트를 추가해가지고 애니메이션을 넣을 것인지 궁금하거든요. 그렇죠. 그렇다면 롯데패키지를 사용한 스피트 리스 위치에서 하나 만들 것 같고요. 그러면 그 파라미터 알규모트 안에다가 이제 저희 타이머 프로바이더 안에 있는 0.7에 대한 타이머를 넣을 것 같고 그 주기마다 돌아가도록 할 것 같습니다. 리셋이 되는 시점은 한 번 씹었..인가? 한 입이 끝났을 때 다시 재시작되도록 그렇게 할 것 같아요 네 알겠습니다

어 그러면은 그...

어떤 부분을 ai한테 맡기고 어떤 기준으로 그래 그건 쓰자 이렇게 하신 건지 그 의사결정 과정 아 그거는 약간 명확한 기준이 있습니다. 아무래도 이 생산성을 최대한 활용을 해야 하니까 제가 구성해 놓은 그림이고, 그 방향으로만 동작해야 하는 구현을 맡겼고요. 리뷰를 했을 때, 그 방향이 아니면... 취소했습니다. 아니면은 그거에 어떤 사이트 이펙트가 있을 수 있으면 취소했고요. 내가 생각한 범위 설명한 범위의 이상을 수정하려 하는 경우에도 취소했습니다 그래서 딱 그 부분만 수정되고 그 항소만 만들 수 있도록 그리고 이게 정상적으로 동작하는지를 봤습니다

아 그럼 이 질문 뒷질문은 제가 궁금한게 있어서 네 근데 아까 전 그 로티 붙이는 부분에 대해서 한입이 끝날때마다 리셋한다고 하셨죠? 네 근데 저희는 30번 40번 기록하시는 분들도 있는데

그때 막 타이머가 그 프로바이더나 이런거 상대방 내부에서 계속 리셋이 되거나 할 때 혹시 생길 수 있는 그런 문제를 한번 생각해 보신다면 어떤 게 있으실 것 같고 그러면 그걸 어떻게 설계를 해서 해결을 하실 것 같은지 일단 리셋을 한다면 한 입이 끝났을 때 다시 그려지도록 할 것 같거든요.

그렇다면 다른 거랑 이제 엮여있지 않으니까 문제가 없을 거라 생각합니다 어찌 됐든 그 메트로놈의 주기에 맞춰서 움직이는 그릇 값만 넣어둘거고 한입이 끝나는 끝나다면 버튼을 누를거고 그러면 이제 set state나 notify listener 그 다음 주기로 그냥 시작만 그냥 네 한번 다시 시작하는 그런 느낌으로 갈 것 같아서 그렇다면 네네 아 왜냐면 저도 이제 타이머 쪽을

버그가 꽤 있는데 여기서 이제 뭔가 이렇게 메모리 누수가 일어나구나 아 네 데이터 값이 제대로 전달이 되지 않거나 이 스테이트가 제대로 작동을 하는 데에서 조금 물론 당연히 코드상의 문제이겠지만 문제가 발생하거나 뭔가 같은 결국 리셋하는 동작들을 어디서 문제가 발생했는지 모르니까 막 앞뒤로 넣고 이런 부분이 있거든요. 그래서 그런 상황에 대해서 뭔가 경험이 있으신지 궁금해가지고. 그 상황을 발견을 해서 이 코드에서 수정이 되어있거든요 어... 지금 과제예요? 네네 이제 확인해야 될 부분은 프로바이더

트윈 타이머 프로바이더 어 일단은 이제 그런 타이머는 구독의 느낌이 강하고 취소하지 않으면 계속 진행되는 형태잖아요 네네네네 아마 말씀하신 문제는 이제 모달로 띄우고 저장을 했을 때 그때 타이머가 출동을 안 해서인 것 같거든요 그래서 저는 이제 페이지로 구성을 했고 다음 페이지 하프가 있을 거거든요 좀 아래쪽에 넥스트 페이지 봤는데

이제 퍼스트 타이머 함수를 만들어놔서 퍼스트 타이머 함수는 그 타이머들을 다 멈추고 다음 페이지로 넘어갈 수 있도록 그래서 페이지뷰를 한 것도 있어요 이버의 한 주기를 끝내고 그다음으로 넘어간다

라는 느낌으로 사용할 때만 사용하도록 구성을 했습니다.

음 네 알겠습니다 이게 뭔가 이 부함수가 굉장히 네 원래 이렇게 쓰는 건가 싶네요 이거는 사실 필요 없는 한 수에요 이거는 없애도 문제가 없구요 그냥 넥스트페이지만 왜냐 보통은 이런 페이지퓨에서 상속값을 이렇게 넣어두지 않으니깐

갑자기 눈에 띄어서 여쭤봤습니다. 아, 아닙니다. 네. 어, 그러면은... 우와, 왜 이렇게 시간이 빨리 가지? 어우, 그러게요. 이력서 본 부분에서 제가 아까 궁금해 했던 게 AI를 저희가 바이브코딩을 쓸 때 클로즈코드를 쓰시죠? 아 네 클럽코드를 쓰고 있습니다 그냥 궁금한건데 모델 뭐 쓰세요? 저는 소네스 4.0을 씁니다 아 진짜요? 오퍼스가 좋은데 오프닝이 너무 많이 받아가지고요 맥스플랜 쓰신다고 안했어요? 맥스플랜 쓰지만 아끼고 싶은 마음이 항상 있습니다

이게 제가 막 잘 아는 건 아니지만 사실 이제 제가 알기로는 한 번을 처리할 수 있는 콤플렉스 양이라고 들었던 것 같은데 저는 작은 부분을 요청한다면 쏘넷을 충분히 하지 않을까라는 생각이 있어서 일단 다른 곡 비 느낌으로 하고 있습니다 적긴 한데 좀 큰 작업 저는 맡길 때 기다려야 되

잖아요 작업을 하는 과정을 계속 감시하고 있는 게 아니면 네 그럴 때 보통은 어떤 식으로 작업을 하세요? 저는 작은 부분을 일단 맡기고 있어요 뭔가 큰 부분을... 이거는 근데 아직 숙련도가 부족해서 일수도 있겠지만 콤보를 했을 때 이걸 내가 감당할 수 있나? 이런 생각이 가끔 들어서

네 아마 그래서 오래 걸린 것 같습니다. 오퍼스를 해서 크게 하는 것도 괜찮은 것 같아요. 이게 부모니가.

네 그 저도 정답이 없는 것 같은데 네 다른 분들은 어떻게 하시나 궁금

아 네 저 제 케이스에는 한 번도 다이브 코딩이라는 게 현업에서는 없었으니까 네네네 이 시간 동안 제가 원래는 뭐 이렇게 코딩을 할 때 집중하던 그 맥락이 많이 헤쳐지는 느낌이어가지고 저 스스로도 이 시간을 어떻게 보내는 게 가장 생산성이 있을지 이 AI의 문맥을 따라가는 게 베스트일 것 같긴 한데 그럼에도 불구하고 수동적으로 제가 따라가다보니까 집중력이 흐려지더라구요 아 그쵸 예 네 그래서 그런 부분에 대해서 다른 분들은 어떻게 하는지 궁금해가지고

네 저 제 주변 개발자들한테도 물어보면 아예 안 쓴다부터 네 필요할 때 그냥 내가 코딩 다 하고 그냥 해석하는 용어로만 쓴다 뭐.. 아니면 막혀놓고 딴 일 한다고 하면서 별 답변이 다 있어가지고 네 쟈님은 어떤 식으로 하시는지 궁금해서.. 저는 좀 작은 단위로 하는 것 같습니다. 한 번에 리뷰가 가능할 정도? 네 네 알겠습니다 어 이력서 주신거에서 네

MVVM 유즈케이스 레이어드 아키텍처 이런거 써주셨는데 어떻게 적용하셨고 각 레이어별로 어떻게 책임을 나누셨는지 아 이제 mvm 은 뭐 이제 앱에서 많이 사용을 하는 그런 방식이죠. 레이어를 나누는 그런 방식인데 어찌됐든 폴린은 객체지향과 클린하키텍처에서 온 거고 필링 아키텍처의 원칙을 그대로 따라해서 우리 플러터 앱에서 이렇게 쓰면 좋습니다라고 공식 문서에서 사실 MVM 패턴을 그렇게 소개를 하고 있어요.

그래서 이제 일단은 저희가 이 여기서 볼 수 있는 페이지 자체는 뷰로 두는거고 화면은 그냥 화면이죠 화면이고 뷰 모델은 그냥 로직들, 화면에 대한 로직과 상태만 보관하고 있는 거고 이제 유즈케이스, 거기에서 나와있는 건 유즈케이스라고 나와있는데 사실 이제 뷰모델이 되게 커질 때가 많잖아요 예를 들면 뷰모델이 500줄, 1,000줄, 2,000줄이 되는 경우가 되게 많은데 일단 플로터에서 권장하는건 중모델은 커지면 안된다거든요

복잡한 로직을 규모 대비해서 하는 것을 지향한다라고 되어 있고 왜냐하면 그러면 승리가 되게 어려워지고 버그가 많이 생기니까 그러면 그냥 순수한 로직들은 유즈케이스라는 다른 레이어로 분리한다 라는 그런 것들이 있어서 그러면 테스트할 때도 그 로직만 테스트하면 되는 거고 그리고 뷰 모델은 그 로직을 이용만 하면 되는 거니까 관리할 수 있는 코드에 관리가 좀 용이하도록 그렇게 저는 분리했습니다. 제가 했던거는 소셜앱, 그때 적용했던거는 소셜앱이였거든요. 채팅앱인데, 애플에 출시할때 채팅앱은

사용자 차단도 해야되고 삭제도 해야되고 하잖아요 네네네 그러면 이제 채팅방이나 친구를 불러올 때마다 이 사람이 차단인지 아닌지 그리고 그러면 그 파이어베이스 스토어 안에 차단 목록 안

에서 차단인지 아닌지 뭐 삭제인지 이런 것들을 다 판단해야 되고 그게 뷰 모델에 그대로 들어가서 엄청 꼬였었거든요 그래서 그러면은 이슈 케이스를 하나 만드는 거죠 예를 들어 어 내 차단 리스트를 가져오고 내 현재 침구리스트를 가져와가지고 여기서 필터링만 해주고

그리고 화면 안에서는 필프링이 된 것만 보여주도록 구성했습니다.

거기에 대해서 그런 뮤직케이스마다 유니테스트도 작성하고 해보신 경험도 있으신 거예요? 유니테스트는 지금 있는 곳에서는 작성을 하고 있고요. 그때 당시에는 작성을 못했어요 그때는 약간 뭔가 이거를 그때는 테스트에 대한 욕구가 잘 안 났었는데 현재는 남매 프로젝트 유닛 테스트를 만들어 주고 있습니다.

알겠습니다 그 유즈케이스 조차 막 비대지거나 여러 페이지에서 로직을 공유를 하면서 중복이 생기거나 이렇게 꼬이는 상황이 되는데 그러면 어떤 기준으로 책임을 나누실 건지 궁금하거든요. 아 그쵸 일단 뮤즈케이스의 경우에는 일단 간단한 작업들은 무조건 배제를 해야한다고 나와 있어요 이제 뭐 CRUD라고 하죠 시나리오에서 복잡한 로직만 뉴스케이스로 저는 묶고 있거든요. 아! 사례가 하나 있습니다. 하나 생각나는 사례가 있는데

제가 지금 하고 있는 프로젝트에서는 관측소에 대한 분류를 뮤직케이스로 뽑았어요.

이 사람이 좌표를 통해서 에셋을 읽고 어디 관측소인지 딱 그 정도만 강렬하게 뮤직비디오를 짜놓고 다른 화면에서는 그냥 그 로직을 갖다 쓰는 거죠. 다른 레이어를 있으니까 사이에 전달해서 사용하고 좀 분리되도록 이렇게 했습니다.

23분이고 30분에 묘왕님이 들어온다 해가지고 네

아 이 부분 여쭤볼게요 그러면은 그 하이마더 앱에서 아 네 fcm 설정 에러 뭐 카카오 공유 api 문제 이런거 해결했다고 네. 실제로 이런 문제를 발견했을 때 어떤 순서로 아 저는 이게 비교가 맞을지 모르겠는데 약간 보드게임 할 때 느낌으로 일단 첫번째는 목표를 정해야 갈 길이 정해지잖아요 네네네 에러난 부분과 원인을 우선 찾으려고 하는 편인 것 같아요 예를 들면 그 때 카카오 초대를 보냈는데 카카오 초대가 전송이 아이오스 에서는 안되었던 거죠 그리고 안드로이드에서는 뭐 뭐 뭐 국가적인 이런거 있었지만 그러면 이게 왜 전송이 안되어지를 파악을 해야되니까 그때는 이제

호스트맨으로 테스트를 해서

포스트맨 진짜 오랜만에 듣는데? 포스트맨으로 계속 엘에서 딸깍딸깍 하고 있으면 너무 오래 걸리니까 포스트맨으로 테스트를 했을 때 어 그때 전송이 안됐던 이유는 첫번째로 iOS 세팅이 안됐어서 iOS 세팅을 했었고 네네네 그럼에도 전송된 로직이 제대로 가지않은 상황이 있어서 이 문제는 그러면 그 카카오 API 내부 설정이 있겠다라는 생각이 들어서 콘솔 들어가서 기입하는 라인이 있거든요. 거기 부분에 iOS는 배제가 되어 있더라고요. 그래서 그 부분 추가를 했습니다.

약간 정황이지만, 목표 설정인 것 같아요 그러면 저희가 이미 배포를... 앱은 해서 장애가 나는 상황을 보잖아요 네네 그러면 보통 이제 그 스트레스 관리를 어떻게 하시고 이제 딱 이 문제를 발견했을 때 30분 액션 플랜을 어떻게 세우시는지 액션 플랜이요? 뭐 이렇게 굳이 표현하자면 그런데 그 딱 확인하고 지금 이렇게 해야겠어 이 단계가 어떻게 흘러가시는지 어 최대한 침착한 성격 같지는 않은데 침착하려고 하는 편이거든요 그래요 네 일단은 어 처음 그 출발점이 잘못되면은 다른 걸 수정할 수도 있게 된다고 생각을 해서 일단 원인 먼저 찾으려고 노력합니다

첫번째는 원인을 찾고, 원인이 찾는 방법은 인터넷이 많잖아요. 원인을 찾고, 조사하고, 앱에 맞게 수정. 이 사고 구조로 가는 것 같습니다.

질문을 드리고 싶은데 본인이 생각하는 어떤 좋은 팀? 개발 문화는 어떤 모습인가요?

있습니다. 진짜요? 네. 좋아하는 분이 있어요. 멋진 분이라고 생각하는 시니어분이 계신데. 어 유명하신 분이세요?

유명할지도 모르겠어요. 서버 개발자 분이신데 누구야 누구야? 아니 막 그 정도로 그런 분은 아니고 완전 고인, 시니어 서버 개발자분이신데 그 분이랑 미팅을 진행하다가 중간에 이제 딸이 일곱살이셔서 같이 밥을 먹게 되는데 저를 소개해 줄 때 아빠 친구야 이렇게 소개해 주시더라구요

제가 만들고 싶은 조직문화는 그런 문화인 것 같습니다.

수평적인? 네, 아니 그러니까 너무 막 그러면 안되겠죠 근데 서로 의견 나눌 때 제한은 없는 정도가 좋은 것 같아요

네. 그렇지 않은 조직이 있나? 제가 잘... 제가 일한 데에서는... 네. 뭔가 이렇게...

기술을 잘 몰라서 까인 적은 많지만 대화에 있어서 뭔가 이렇게 상급자 이런 게 없었거든요 저는 그래서 위드분이랑 대화할 때도 뭔가 이렇게 하달 받는다는 느낌보다는 내가 해야할 프로젝트에 대해서 서로 설명하고 공유한다는? 항상 그런 느낌이었어가지고 그렇지 않은 조직이 있는지 잘 모르겠네요 SBS는 군대 문화였어가지고요. 아 그렇죠. 개발 조직이랑 다르죠. 네. 그래서 그런 수평적인 개발 조직 문화. 그런 좋은 팀 분위기를 항상 만들고 싶다는 생각이 있었어요

알겠습니다. 잠시 쉬거나 하시지 않으셔도 될까요? 저는 괜찮습니다. 아니면 중간에 잠깐 끊어가도 돼요. 저는 상관없습니다. 그럼 저 1분만 화장실을 좀 갔다 올게요 아 네 그럼 저도 화장실 좀 다녀오겠습니다