

일상의 가치를 더하는 개발자

사용자를 위한 서비스를 고민하는 개발자입니다.

두 번의 서비스 운영을 통해 니즈 반영과 가치 전환을 경험했습니다.

새로움에 두려움이 없는 개발자입니다.

인력부족을 해결하기 위해 일주일만에 React를 학습하고 서비스를 운영했습니다.

소통하는 개발자입니다.

팀장을 맡아 스프린트를 진행하며 기획, 개발, 운영, 마케팅까지 프로젝트 전과정을 경험했습니다.

정구연

+82 10-2433-1103

lottlerudolph@gmail.com

[GitHub](#)

[Linked In](#)

프로젝트 경험

뉴스낵(Newsnack) - 솟폼 뉴스 플랫폼

2024.04 ~ 2024.12. (팀장 / 백엔드 개발)

사용 기술 스택

- | | | | |
|-------------------|-----------|----------|--------------|
| - Java 17 | - Python3 | - MySQL | - AWS |
| - Spring MVC | - FastAPI | - Redis | - Git Action |
| - Spring Security | - MoviePy | - Docker | - Nginx |

소개

'뉴스낵'은 낮은 문해력으로 뉴스 소비에 어려움을 겪는 바쁜 현대인을 위한 기능을 제공하는 앱입니다.

기획의 출발점은 “바쁜 현대인들이 뉴스를 좀 더 편하게 접할 수 있는 방법이 없을까?” 하는 문제의식이었습니다. 직접 인터뷰와 설문을 통해 사용자 니즈를 수집했고, 실제로 '긴 텍스트가 부담스럽다', '핵심만 빠르게 알고 싶다'는 반응이 많았습니다.

그래서 ‘텍스트 → 영상’으로 콘텐츠를 바꾸는 자동화 파이프라인을 기획했고, GPT로 요약, CLOVA TTS로 음성 변환, Stable Diffusion으로 이미지 생성까지 한 번에 처리할 수 있도록 백엔드 구조를 설계했습니다. 비용 문제 해결을 위해 AWS Lambda + Step Functions + Spot 인스턴스를 활용해 GPU 운영 비리를 약 70% 절감했습니다.

기획부터 운영까지 8개월간 팀장과 백엔드 역할을 맡으며, 사용자 피드백을 받아 지속적으로 개선해 나갔고, **운영 한 달 간 MAU 500, DAU 200을 달성했습니다.**

담당했던 기능 개발 및 개선

솝폼 변환 파이프라인 구축

뉴스 콘텐츠를 1분 이내의 솟폼 영상으로 자동 생성하기 위한 AI 기반 파이프라인을 구축했습니다.

초기에는 뉴스 수집부터 요약문 작성, 음성 합성, 배경 이미지 제작, 영상 렌더링까지 모든 과정을 수작업으로 진행했으며, 1건당 평균 15~20분이 소요되어 확장성이 제한되는 문제가 있었습니다. 이를 해결하기 위해 AI 모델 연계를 통한 자동화 파이프라인을 설계하고 구축했습니다.



1단계에서는 필요한 리소스를 자동으로 생성합니다.

- GPT API를 활용한 뉴스 요약문 자동 생성
- CLOVA Voice를 활용한 TTS 기반 음성 생성
- Stable Diffusion을 활용한 배경 이미지 생성
- Montreal Forced Aligner(MFA)를 이용한 자막 타임스탬프 추출

2단계에서는 위 리소스들을 조합해 영상으로 변환합니다.

이를 위해 Python 기반 영상 편집 라이브러리인 MoviePy를 활용했습니다. MoviePy는 타임라인 기반 자막 삽입, 배경 이미지와 오디오 싱크, 자막 애니메이션 적용 등을 손쉽게 처리할 수 있어, 빠른 개발과 유연한 편집이 가능했습니다.

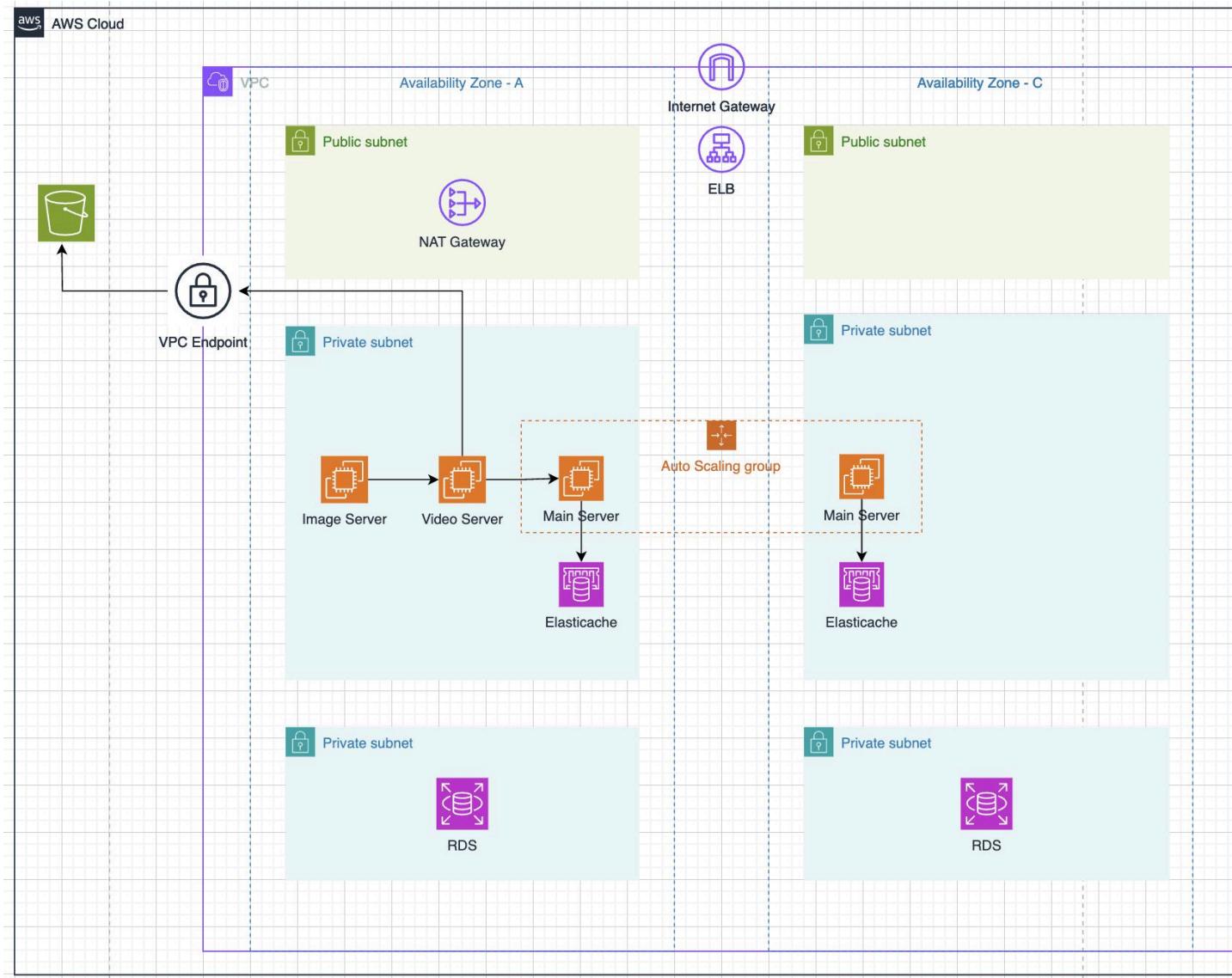
초기에는 Java 진영의 Xuggler, JCodec, FFmpeg JNI wrapper 등의 라이브러리도 고려했지만, 낮은 추상화 수준과 자막 삽입 및 편집에 대한 복잡성, 커뮤니티 자료 부족 등으로 인해 빠른 프로토타이핑에 어려움이 있었습니다. 반면 MoviePy는 Python 환경에서 다양한 미디어 포맷을 직관적으로 처리할 수 있었고, 다른 AI 처리 파이프라인들과의 연동도 용이했습니다.

프로젝트 경험

MoviePy 사용을 위해 별도의 FastAPI 기반 영상 처리 서버를 구축해 Python 환경을 분리 관리했으며, Spring 서버에서 생성 요청이 들어오면 FastAPI 서버로 요청을 전달하여 비동기적으로 영상 렌더링이 수행되도록 구성했습니다. 렌더링이 완료된 영상은 S3에 저장되며, 최종적으로 사용자 피드에 노출됩니다.

이 파이프라인을 통해 기존 대비 작업 시간을 약 90% 이상 단축했으며, 하루 수백 건의 뉴스 영상을 자동으로 생성할 수 있는 수준으로 확장할 수 있었습니다.

GPU instance, NAT 운영비용 최적화



뉴스숏폼 자동 생성 파이프라인은 이미지 생성, 음성 합성, 영상 렌더링 등 GPU 자원이 집중적으로 요구되는 구조였기 때문에, 운영 비용을 최소화하면서 안정적인 인프라를 구성하는 것이 핵심 과제였습니다. 이를 해결하기 위해 AWS Step Functions와 Lambda를 활용한 GPU Auto-Provisioning 아키텍처를 구축했습니다. AWS EventBridge가 StepFunctions를 호출하면, Lambda가 GPU 서버를 자동으로 기동하고, 영상을 변환이 끝난 후에 인스턴스를 자동 종료하는 구조로 설계하여 상시 기동 대비 약 70%의 비용을 절감할 수 있었습니다.

또한, NAT Gateway를 통한 S3 트래픽 비용을 최소화하기 위해 VPC Endpoint를 도입했습니다. 이를 통해 퍼블릭 NAT를 거치지 않고 프라이빗 서브넷에서 S3에 직접 접근할 수 있어 트래픽 비용을 절감하는 동시에 보안성도 강화할 수 있었습니다.

추가로, 두 개의 Availability Zone에 걸친 Auto Scaling Group 구성과 ELB 기반 로드밸런싱을 통해 서비스의 고가용성과 확장성을 확보하고, 필요한 순간에만 자원을 사용하는 구조를 실현했습니다.

Redis 기반 뉴스 랭킹보드 구현

서비스 초기에는 인기 뉴스 정렬을 위해 매번 DB에 조회수, 좋아요, 댓글 수를 기준으로 정렬하는 쿼리를 수행했습니다. 하지만 사용자 수가 증가하고 트래픽이 몰리는 시점에서 응답 지연이 빈번해졌고, DB 부하도 증가했습니다.

```
/* 뉴스 인기순 정렬 가중치 */
public static final Long VIEW_WEIGHT = 1L; 2 usages
public static final Long COMMENT_WEIGHT = 4L; 4 usages
public static final Long REACTION_WEIGHT = 3L; 4 usages
public static final Long SHARE_WEIGHT = 6L; 2 usages
public static final Long DATE_WEIGHT = -1000L; 1 usage
```

이를 해결하기 위해 Redis Sorted Set을 활용한 랭킹보드를 구축했습니다. 각 뉴스 콘텐츠마다 사용자 반응(조회, 좋아요, 공유 수 등)을 기반으로 가중치를 부여한 점수를 실시간으로 반영하고, 카테고리별로 별도의 Sorted Set으로 관리했습니다. 클라이언트가 요청할 때마다 실시간으로 랭킹 순위와 뉴스를 빠르게 조회할 수 있도록 했고, 비동기 Task로 주기적인 정렬 결과를 DB에 동기화했습니다.

도입 이후 평균 인기 뉴스 응답 속도는 기존 1.2초에서 약 0.4초로 단축됐고, 사용자 체류 시간과 클릭률도 유의미하게 증가했습니다.

댓글 알림 기능의 비동기 처리로 API 응답 속도 개선

댓글 기능을 구현하던 중, 대댓글 작성 시 API 응답 속도가 비정상적으로 느려지는 현상을 발견했습니다. 로그와 APM 분석을 통해 원인을 추적해본 결과, 대댓글 작성 후 FCM(Firebase Cloud Messaging)을 통한 알림 전송 로직에서 병목이 발생하고 있다는 점을 확인했습니다.

알림 전송은 평균 수 초 이상 소요될 때도 있었고, 이는 사용자 경험에 직접적인 영향을 미쳤습니다. 또한, 댓글 생성과 알림 발송은 본질적으로 독립적인 기능임에도 하나의 흐름으로 강하게 결합되어 있어 유지보수성과 확장성 측면에서도 부적절하다고 판단했습니다.

이를 개선하기 위해 알림 전송 로직을 별도의 서비스로 분리하고, Spring의 @Async 어노테이션을 활용하여 비동기 실행되도록 리팩토링했습니다. 댓글 API는 댓글 데이터를 저장한 후 즉시 응답을 반환하고, 이후 알림은 백그라운드에서 처리되도록 구조를 변경했습니다.

그 결과, 댓글 작성 API의 응답 시간이 평균 2초에서 100ms 이하로 감소하며 사용자 경험이 크게 개선되었고, 기능 간 의존성 분리로 인해 시스템의 구조적 안정성과 확장 가능성도 확보할 수 있었습니다.

프로젝트 경험

팝핀(Poppin) - 팝업스토어 정보 공유 플랫폼

2023.12 ~ 현재 (팀원 / 백엔드 개발)

사용 기술 스택

- Java 17
- MySQL
- AWS
- Spring MVC
- Redis
- Git Action
- Docker
- Spring Security
- FCM
- Nginx

소개

'Poppin'은 서울 곳곳에서 열리는 팝업스토어 정보를 수집해, 사용자가 한눈에 확인할 수 있도록 만든 위치 기반 통합 정보 서비스입니다.

2023년부터 팝업스토어 열풍이 불고 있지만, 사용자들은 정보를 직접 찾아다녀야 했습니다. 종료된 행사, 잘못된 위치 정보 등으로 혼란을 겪는 일이 잦았고, '정보가 흩어져 불편하다면, 직접 정리해서 보여주자'는 문제의식에서 출발해 개발을 시작하게 되었습니다.

저는 이 프로젝트에서 백엔드 개발과 인프라 운영을 담당했습니다. Spring Boot로 서버를 구축하고, AWS 환경에서 서비스 배포 및 운영을 진행했습니다. 특히 트래픽 급증으로 인해 응답 속도 지연 문제가 발생했을 때, Redis 캐싱과 Auto Scaling을 적용해 평균 응답 속도를 약 50% 개선하며 안정적인 서비스를 제공할 수 있었습니다.

실제 사용자 데이터를 기반으로 운영하며 문제를 개선해 나간 경험은, 사용자 중심의 실전 개발 역량을 키울 수 있는 소중한 계기가 되었습니다.

담당했던 기능 개발 및 개선

공통 응답 DTO 및 Exception Handler 구성

프로젝트 초기에는 API 응답 형식이 컨트롤러마다 상이하고, 예외 발생 시 에러 메시지 구조가 제각각이라 프론트엔드와의 협업에 불편이 발생했습니다. 특히 사용자 요청 종류가 많아질수록 응답 포맷의 불일치와 예외 처리 체계의 부재는 유지보수 리스크로 이어질 수밖에 없었습니다.

이러한 문제를 해결하기 위해 ResponseDto<T>를 정의하고, 모든 API 응답을 success, data, error로 통일된 형태로 감싸는 구조를 도입했습니다. 성공 응답은 ok(), 실패 응답은 fail() 정적 메서드로 구분해 반환하며, 프론트엔드에서는 일정한 형식의 데이터를 예측 가능하게 받아 처리할 수 있게 되어 개발 효율성과 협업 편의성이 크게 향상되었습니다.

예외 처리 측면에서는 @RestControllerAdvice와 @ExceptionHandler를 활용한 전역 예외 처리 핸들러를 구현해, 공통적으로 발생할 수 있는 요청 파라미터 누락, 잘못된 타입, 인증 오류, 리소스 미존재 등의 상황을 ErrorCode enum 기반으로 세분화해 관리했습니다. 개발자가 직접 정의한 CommonException과 시스템 예외를 구분 처리함으로써 사용자에게 불필요한 기술 정보를 감추고, 개발자에게 상세한 로그를 제공할 수 있도록 했습니다. 이로써 서비스 안정성과 유지보수 편의성을 동시에 확보할 수 있었습니다.

```
public enum ErrorCode {  
    //Bad Request Error  
    INVALID_PARAMETER( code: "40000", HttpStatus.BAD_REQUEST, message: "유효하지 않는 파라미터입니다." ),  
    MISSING_REQUEST_PARAMETER( code: "40001", HttpStatus.BAD_REQUEST, message: "필수 파라미터가 누락되었습니다." ),  
    INVALID_ROLE( code: "40002", HttpStatus.BAD_REQUEST, message: "유효하지 않은 권한입니다." ),  
    INVALID_PROVIDER( code: "40003", HttpStatus.BAD_REQUEST, message: "유효하지 않은 제공자입니다." ),  
    INVALID_HEADER( code: "40004", HttpStatus.BAD_REQUEST, message: "유효하지 않은 헤더값입니다." ),  
}
```

Full-Text Search 기반 검색 기능 성능 개선

초기에는 LIKE 연산자를 이용한 단순 문자열 검색 방식으로 구현되어 있었지만, 데이터가 1천 건 이상 누적되면서 검색 성능 저하가 발생했습니다. 특히 다중 키워드 검색에서는 쿼리 실행 시간이 급격히 길어졌고, 이는 사용자 경험 저하로 이어졌습니다.

이러한 문제를 해결하기 위해 MySQL의 Full-Text Search 기능을 도입했습니다. MATCH(title, description) 구문과 AGAINST(? IN BOOLEAN MODE) 조건을 활용하여 제목과 설명 컬럼에 대해 자연어 기반 검색을 구현했고, 해당 컬럼에 Full-Text 인덱스를 사전 구축하여 검색 성능을 크게 개선할 수 있었습니다.

기존보다 정교한 키워드 매칭이 가능해졌을 뿐 아니라, 평균 응답 속도는 약 300ms 이상 단축되어 빠르고 유연한 검색 환경을 제공하게 되었습니다. 또한 쿼리 결과 정렬 및 필터링 로직도 함께 개선해 사용자 만족도를 높일 수 있었습니다.

Enum 기반 정적 데이터 관리

초기에는 팝업 운영상태나 카테고리 같은 정적 데이터를 문자열(String)로 직접 저장하고 비교하는 방식으로 구현했습니다. 그러나 이 방식은 오타나 잘못된 값 저장 가능성을 열어두고 있어, 코드와 데이터 간의 불일치 문제가 자주 발생하고, 조건문 비교 시 하드코딩된 문자열로 인해 가독성과 유지보수성이 저하되는 문제가 있었습니다.

이러한 문제를 해결하기 위해 운영상태(운영 중, 종료, 예정 등)를 나타내는 데이터를 Enum 클래스로 정의하고, DB 저장 시에도 enum의 value만 저장하도록 구조를 변경했습니다. 예를 들어 EOperationStatus enum을 정의하고, 각 상태 값을 의미 있는 이름과 함께 관리함으로써 비즈니스 로직 내에서 더 명확한 조건 처리가 가능해졌습니다.

프로젝트 경험

```
public enum EOperationStatus {  
    NOTYET("NOTYET"),  
    OPERATING("OPERATING"),  
    TERMINATED("TERMINATED"),  
    EXECUTING("EXECUTING");  
  
    private final String status;  
}
```

이를 통해 개발자 간의 공통된 이해 기반이 형성되었고, 잘못된 값 저장이나 비교 오류를 컴파일 타임에 방지할 수 있었으며, 추후 상태가 추가되거나 변경되더라도 enum 클래스만 수정하면 되어 확장성과 유지보수성 또한 크게 향상되었습니다.

VPC 및 Git Action 기반 CI/CD 구성

서비스 운영 안정성과 트래픽 대응력을 높이기 위해 AWS 기반 인프라를 구성했습니다. 기본적으로 퍼블릭/프라이빗 서브넷을 분리한 VPC를 설계하여 보안성과 관리 효율성을 확보했고, 서비스 트래픽을 효과적으로 분산 처리하기 위해 Application Load Balancer(ALB)를 도입했습니다. ALB는 헬스체크를 통해 비정상 인스턴스를 자동 감지하고 트래픽을 우회시켜 고가용성 확보에 기여했습니다.

또한 Auto Scaling Group(ASG)을 적용하여 시간대별 사용자 트래픽 변화에 유연하게 대응할 수 있도록 구성했습니다. 피크 타임에는 인스턴스를 자동 확장하고, 비활성 시간에는 축소하여 비용 효율성과 확장성을 동시에 확보할 수 있었습니다.

배포 자동화는 GitHub Actions를 통해 구현했지만, API 서버가 프라이빗 서브넷에 존재해 직접 접근이 불가능한 환경이었습니다. 이를 해결하기 위해 퍼블릭 서브넷의 Bastion Host를 통해 SSH로 내부 서버에 접속한 뒤, 해당 서버에서 프라이빗 네트워크 내 API 서버에 접근해 배포를 수행하도록 배포 흐름을 구성했습니다. 이를 통해 보안을 유지하면서도 CI/CD 자동화를 실현할 수 있었습니다.

프로젝트 경험

호텔 및 리조트 관리 시스템 - 호텔, 부대시설, 교통편 관리

2023.09 ~ 2023.11. (팀장 / 프론트엔드 개발)

사용 기술 스택

- React
- Typescript
- Vite

소개

객실, 예약, 고객 정보를 효율적으로 통합 관리할 수 있는 웹 기반 전산 시스템을 목표로 기획된 팀 프로젝트로, 호텔 및 리조트 운영을 통합 지원하기 위해 개발되었습니다.

초기 설계 단계에서 유스케이스 다이어그램, 액티비티 다이어그램, 시퀀스 다이어그램 등 다양한 UML 기반 설계 문서를 작성하며, 요구사항 기반의 시스템 흐름을 시각적으로 이해하고 이를 UI 설계에 반영했습니다.

담당했던 기능 개발 및 개선

UML 설계 및 요소 별 상태관리

시스템의 동작 흐름을 명확하게 파악하고 구현 리스크를 줄이기 위해 UML 기반의 설계를 선행했습니다. 필요한 요구사항의 유스케이스를 정의하고, 각 유스케이스의 흐름을 UML로 도식화했습니다.

예를 들어, 시퀀스 다이어그램에서는 관리자가 '객실 추가 메뉴'를 선택하면 웹 클라이언트가 해당 UI를 렌더링하고, 이후 사용자가 정보를 입력하고 '추가' 버튼을 누르면, /api/v1/rooms로 POST 요청이 발생하는 구조를 명확하게 정의했습니다. 이 덕분에 API 연동 지점, 즉 네트워크 요청이 발생하는 시점과 사용자 인터랙션(입력/확인)의 순서를 사전에 파악할 수 있었고, 이를 기준으로 컴포넌트 간 책임을 나눌 수 있었습니다.

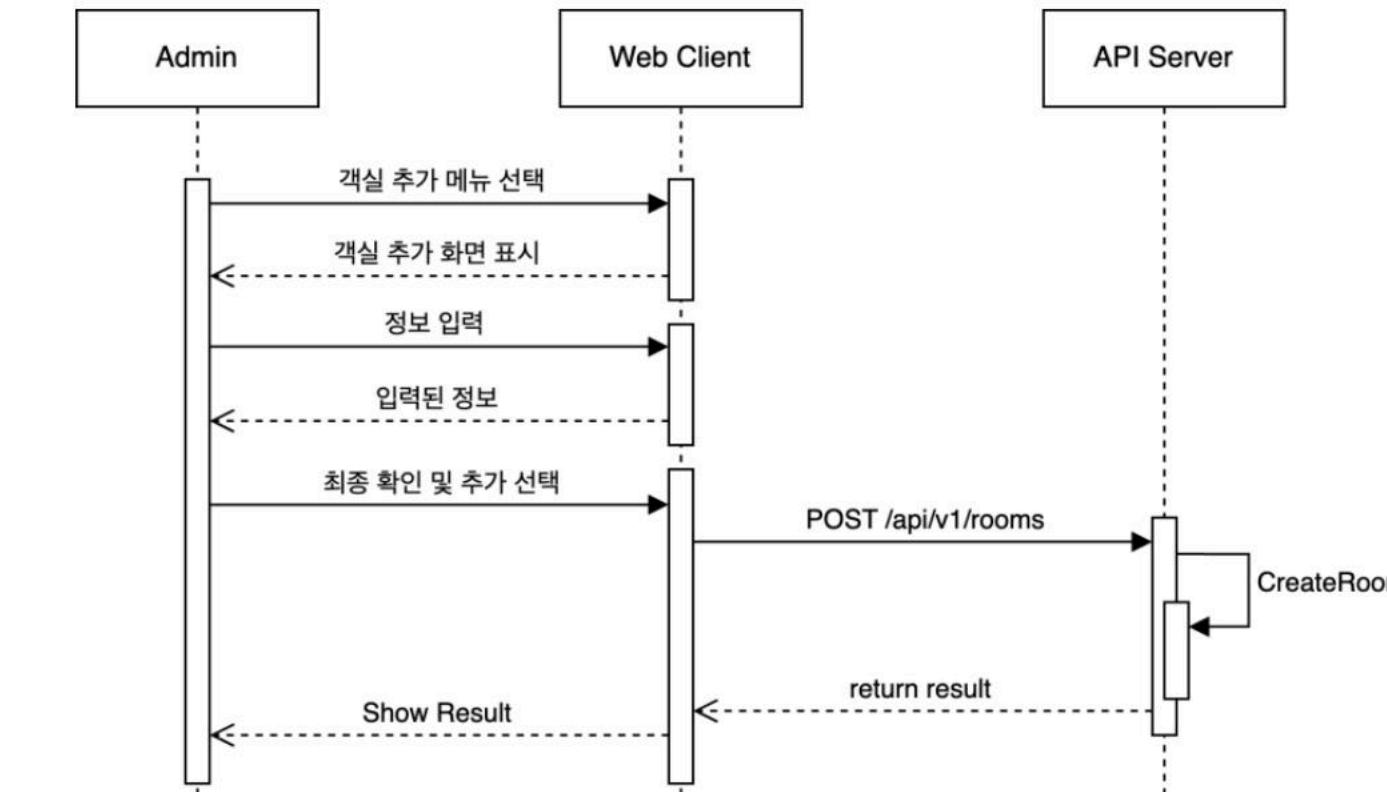
3.1. UseCase Specification

- ① (관리자용) 객실 관리
- (관리자용) 객실 정보 추가

시스템 제목	호텔 및 리조트 예약 관리 시스템
유스케이스 이름	객실 추가
액터	관리자, 인증 시스템
시작 조건	관리자는 시스템에 등록된 상태여야 한다. 내부 시스템과 인증 시스템과의 연결이 정상적이어야 한다.
기본 흐름	<ol style="list-style-type: none">관리자는 시스템에 등록된 관리자 계정으로 로그인한다.내부 시스템은 인증 시스템 측에 로그인 정보 인증을 요청한다.인증 시스템 측은 로그인 정보를 확인하여 일반 사용자인지 관리자인지 확인 후 인증한다.올바르게 인증되어 관리자 계정으로 로그인 된 후 내부 시스템은 홈페이지에서 관리자만 확인할 수 있는 관리자 메뉴를 표시한다.관리자는 관리자 메뉴에서 객실 관리를 선택한다.내부 시스템은 객실 관리 화면을 표시한다.관리자는 객실 메뉴에서 객실 추가를 선택한다.내부 시스템은 날짜 정보를 표시한다.관리자는 객실을 추가할 날짜를 선택한다.내부 시스템은 객실 추가 화면을 표시한다.관리자는 추가할 객실의 정보를 입력한다.내부 시스템은 관리자가 입력한 정보를 표시한다.관리자는 입력한 정보를 확인 후 완료를 선택한다.내부 시스템은 객실을 예약 가능 목록에 추가 후 완료 되었다는 표시를 한다.관리자는 객실 추가가 완료된 것을 확인한다.
대안 흐름	3a 1. 관리자 인증에 실패한 경우 단계 1로 돌아간다. 12a 1. 관리자가 중복되는 내용을 입력했다면 안내 메시지를 표시하고 단계 7로 돌아간다.
종료 조건	내부 시스템이 최종적으로 입력된 정보를 관리자에게 표시하고 관리자가 정상적으로 객실이 추가 되었음을 확인한다.

4.1 시퀀스 다이어그램

- ① (관리자용) 객실 관리
- (관리자용) 객실 정보 추가



실제 개발 단계에서는 React를 기반으로 화면을 구성하고 useState를 통해 객실 크기, 구성, 서비스 항목 등 다양한 입력 상태를 개별적으로 관리했습니다.

또한 TypeScript를 도입하여 각 입력 항목의 타입을 명확히 선언하고, 객실 구성, 서비스 선택, 침대 정보 등 복합적인 사용자 입력을 타입 안전하게 처리할 수 있도록 구조화했습니다. 이를 통해 실수로 인한 런타임 오류를 줄이고, 데이터 흐름의 일관성을 유지할 수 있었습니다.

프로젝트 경험

하투시그널 - 교내 축제 주점 소개 및 인원 매칭 서비스

2023.09 ~ 2023.10. (팀원 / 프론트엔드 개발)

사용 기술 스택

- React
- Typescript
- Vite

소개

'하투시그널'은 교내 축제에서 낯선 사람들과 자연스럽게 어울릴 수 있도록 돋는 인원 매칭 서비스입니다.

축제 기간, "같이 갈 사람이 없다"는 말이 자주 들리는 걸 보고, 자연스럽게 연결되는 서비스가 있다면 더 많은 학생들이 축제를 즐길 수 있지 않을까 하는 생각에서 프로젝트가 시작되었습니다.

저는 이 프로젝트에서 매칭 리스트 확인, 매칭 보내기, 수락 및 거절 화면을 구현했습니다. 당시에는 프론트엔드 경험이 전무했고, 백엔드로 활동하던 교내 개발 커뮤니티에서 처음으로 참여한 협업 프로젝트였습니다. 생소한 환경이었지만 React 공식 문서를 참고하고, 동료들에게 모르는 부분을 적극적으로 질문하며 기능을 하나씩 구현해 나갔습니다.

이 과정을 통해 운영 이틀 간 500명의 사용자를 유입시켰으며, 협업의 중요성과 새로운 기술을 빠르게 익히는 방법을 배울 수 있었습니다. 익숙하지 않은 환경에서도 끊임없이 배우며 팀에 기여한 이 경험은, 앞으로 어떤 프로젝트에서도 유연하게 적용하고 함께 완성해낼 수 있다는 자신감을 심어주었습니다.

담당했던 기능 개발 및 개선

컴포넌트 구조 분리 및 UI 라이브러리 연계

styled-components를 활용해 UI를 설계함으로써 구조적 명확성과 유지보수성을 확보했습니다. 예를 들어, 상단 네비게이션 바와 사이드바는 각각 Nav, SideBar 컴포넌트로 분리해 독립적으로 동작하도록 설계했고, 사용자 매칭 정보를 시각화하는 MeetingGroup 컴포넌트는 재사용성을 고려해 구성했습니다.

스타일링은 styled-components로 진행하여 각 컴포넌트와 스타일을 한 파일 내에서 관리함으로써 스타일의 지역화와 동적 스타일 적용이 가능하도록 했습니다.

이러한 컴포넌트 분리 및 스타일 설계 방식은 팀원들과의 역할 분담을 수월하게 만들었고, 각자 맡은 기능을 독립적으로 구현하고 유지보수할 수 있는 기반이 되었습니다.

상태 기반 사용자 인터페이스 제어

매칭 요청/응답 흐름에 따라 실시간으로 UI가 변화해야 했기 때문에, 상태 관리를 통해 사용자 경험(UX)을 유연하게 제어하는 구조가 필요했습니다.

예를 들어:

- 사용자가 매칭 요청을 이미 보냈는지에 따라, 다시 요청을 보내는 버튼이 비활성화되거나 다른 UI로 전환되도록 했고,
- 상대방의 요청을 거절했는지 여부에 따라 수락/거절 버튼을 숨기거나 고정된 안내 메시지를 노출시켰습니다.
- 또한, 해당 사용자가 팀의 리더인지 여부에 따라 매칭 요청을 보낼 수 있는 권한 자체를 제한하는 로직도 적용했습니다.

이러한 조건 분기들은 모두 상태 값으로 관리되었고, 사용자 행동과 연결되어 실시간으로 반응하는 UI를 만들어냈습니다.

이를 통해 사용자는 자신의 역할과 현재 상황에 맞는 버튼과 안내만 볼 수 있게 되었고, 자연스러운 UX 흐름이 가능해졌습니다.

