

사용자 경험을 위해서라면  
**처음부터** 시작할 수 있는  
프론트엔드 개발자 입니다.

## About

**Name** 박철현

**Birth** 2000.10.16

**Phone** 010-3180-7113

**Email** play3step@naver.com

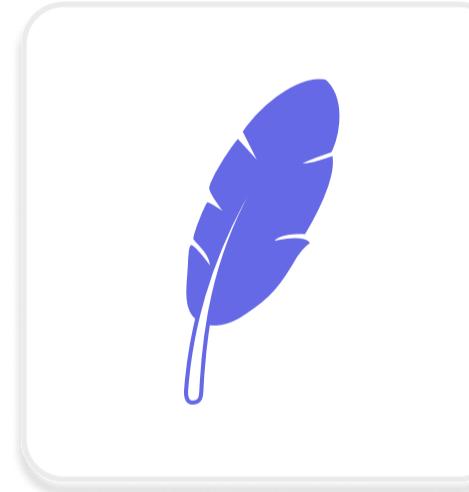
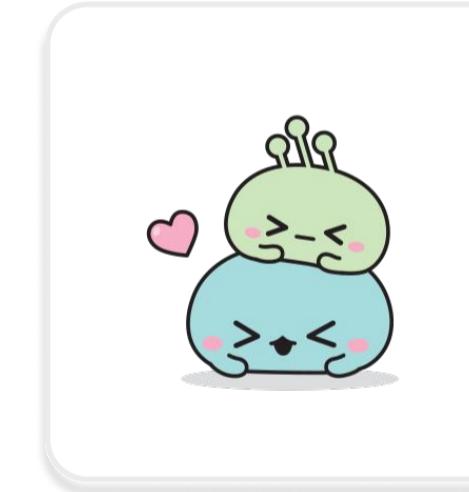
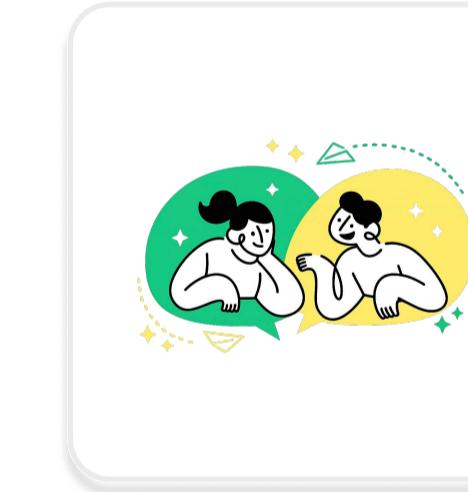
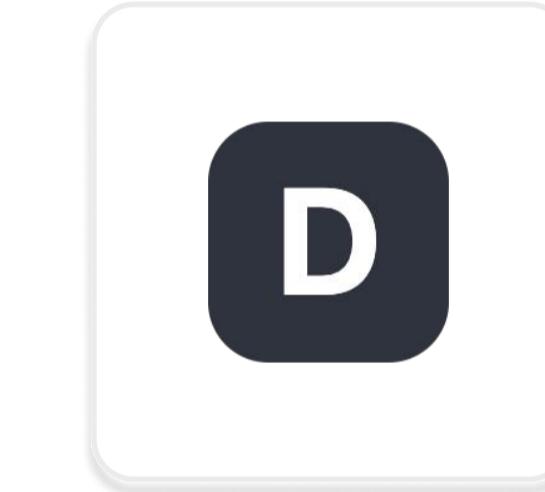
**Github** <https://github.com/play3step>

## Experience

- 2023.09 ~ 2024.01 UMC 5기 (University MakeUs Challenge)
- 2024.01 ~ 2024.02 Helper Robotics (인턴) Frontend Developer
- 2024.04 ~ 2024.10 2024 한이음 ICT멘토링 참여
- 2019.03 ~ 2025.02 한국공학대학교 (소프트웨어학과)
- 2025.02 ~ 2025.04 2025년 제1회 정기 기사 (정보처리기사)
- 2025.05 ~ 2025.10 프로그래머스 데브코스 (부트캠프)

# Project

---

2023.10 ~ 2024.10	2024.04 ~ 2024.10	2025.03 ~ 2025.06	2025.09 ~ 2025.10	2025.12 ~ 진행중
				
<b>MoreView</b>	<b>PlanDing</b>	<b>MapleLink</b>	<b>ZoopZoop</b>	<b>Deemo</b>
2D/3D 요소를 조합한 인터랙티브 프레젠테이션 툴	개인과 그룹 일정을 통합 관리 하는 실시간 일정 플랫폼	메이플스토리 길드를 관리하고 탐색할 수 있는 통합 플랫폼	수집부터 공유까지, 생 각을 이어주는 플랫폼	Mac OS 컨셉의 개인 포트폴리오 사이트



**프로젝트 기간** 2024.04 ~ 2024.10

**프로젝트 소개** PlanDing은 개인 및 팀 단위의 일정 관리를 지원하는 협업 플랫폼입니다. 사용자는 개인 일정뿐만 아니라 팀 프로젝트 일정도 한곳에서 관리할 수 있으며, 실시간 공유 기능을 통해 팀원 간의 효율적인 커뮤니케이션과 일정 조율이 가능합니다. 또한 각 사용자는 자신만의 개인 플래너 기능을 활용해 보다 세밀게 계획을 세울 수 있습니다.

**인원** 4명(FrontEnd 1, BackEnd 1, Android 1 Designer 1)

**기술스택** React, Redux, WebSocket, MUI, TailwindCSS, dayjs, EventSourcePolyfill

**URL** GitHub : <https://github.com/2024-Hanium-PlanDing/PlanDing>

시연 영상 : <https://youtu.be/a1mq1rUVoYg>



## 개인 일정 관리 도구의 한계

기존의 캘린더나 To-Do 앱들은 개인의 일정과 관리를 잘 지원하지만, 팀 프로젝트나 협업 상황에서는 제한적이다.

- Google Calendar: 개인 일정에 특화, 팀 협업 기능 부족
- Todoist: 개인 할 일 관리 중심, 실시간 협업 어려움

## 팀 협업 도구의 복잡성

Slack, Notion 같은 협업 도구들은 기능이 복잡하고, 단순한 일정과 할 일을 위해서는 과도한 설정과 학습 비용이 필요하다.

- Slack: 커뮤니케이션 중심, 일정 관리 기능 미흡
- Notion: 다양한 기능이지만 설정이 복잡하고 진입 장벽 높음

## 핵심 아이디어

개인 일정과 팀 협업을 하나의 플랫폼에서  
자연스럽게 연결하는 서비스가 부족

## PlanDing 목표

- 개인 일정과 그룹 일정의 통합적 관리
- 실시간 커뮤니케이션과 일정 관리의 결합
- 직관적이면서도 협업에 최적화된 UI/UX



## 시각화된 일정 관리

주간·일간 타임테이블을 통해 하루 일정을 한눈에 확인할 수 있습니다.

- 제목, 내용, 시간, 날짜 단위로 일정을 작성하고, 팀원들과 실시간으로 공유할 수 있습니다.
- 팀원 초대 및 즐겨찾기 가능합니다.

The screenshot shows a weekly calendar for August 2024. The sidebar on the left allows users to create events by inputting a title and description, setting a date and time, and choosing a color. The main area displays a grid where each row represents a day from Monday to Sunday. The 31st of August is highlighted with a yellow background, indicating it is a scheduled event. The interface is clean and modern, with a light blue header and a white background.

## 그룹 관리 및 초대 시스템

그룹 생성, 멤버 초대, 권한 관리 등 협업 중심의 그룹 관리 기능을 제공합니다.

- 그룹 이름과 이미지로 직관적인 그룹 식별이 가능합니다.
- 초대받은 그룹을 사이드바에서 손쉽게 이동할 수 있으며,
- 알림을 통해 해당 그룹의 스케줄 및 일정도 즉시 확인할 수 있습니다.

The screenshot shows the group management section of the PlanDing app. On the left, there is a sidebar with a '나의 일정' (My Schedule) section and a '팀 일정' (Team Schedule) section. The '팀 일정' section lists a group named 'PlanDing ict 연토링'. Below the list is a button with a plus sign to add more groups. The overall design is consistent with the rest of the app, featuring a light blue header and a white background.

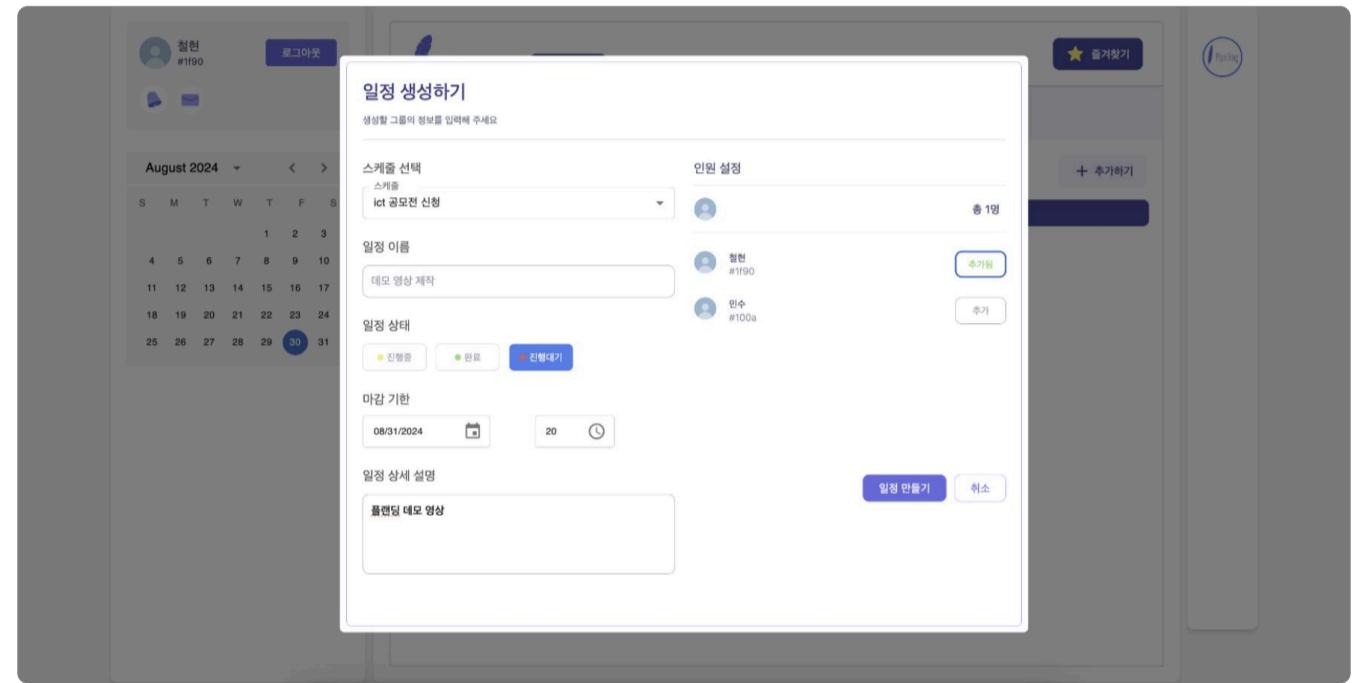


# PlanDing | 핵심 기능

## 플래너(Planner) 관리

그룹 또는 개인 일정에 새로 할 일(To-Do)을 추가하여 프로젝트를 체계적으로 관리할 수 있습니다.

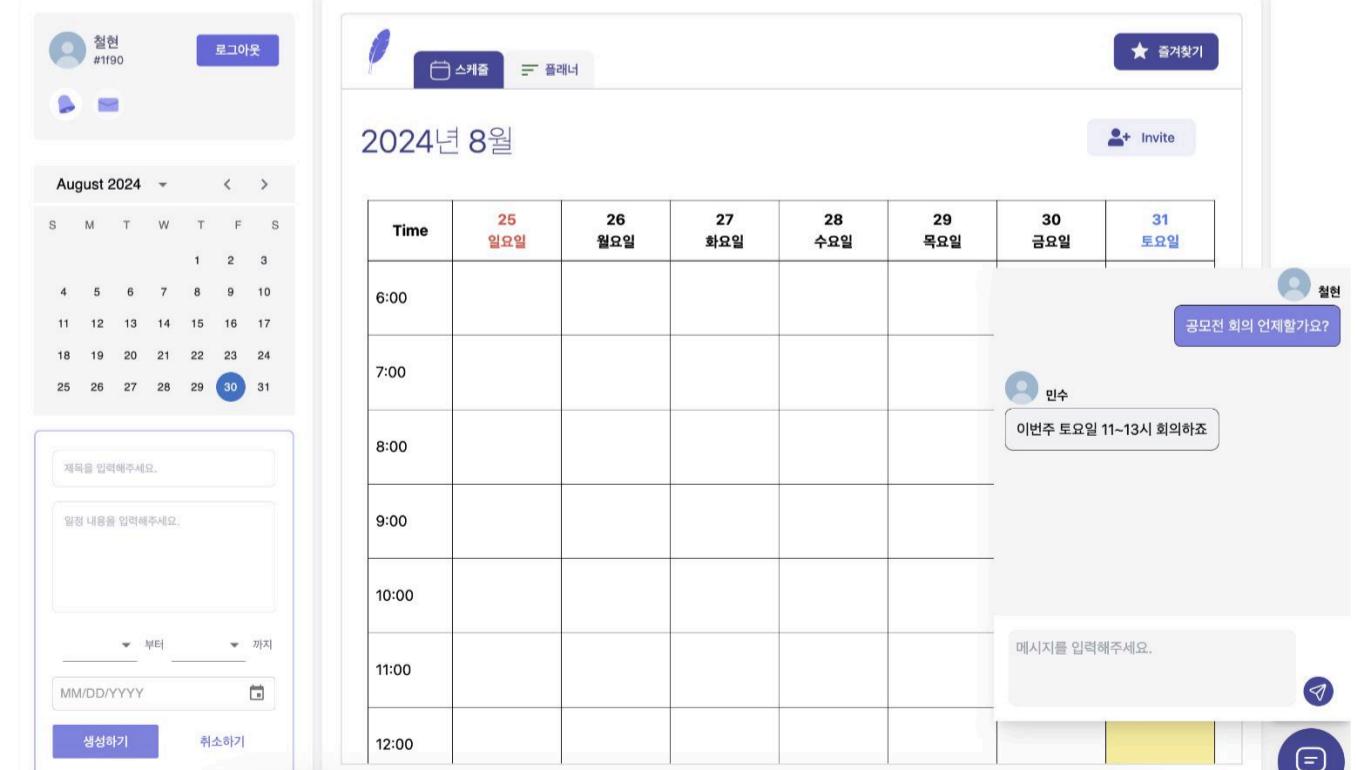
- 일정별로 상태(진행 중/완료)를 설정할 수 있습니다.
- 일정은 캘린더와 연동되어 프로젝트별 진행 현황을 한눈에 파악할 수 있습니다.



## 채팅 및 커뮤니케이션

그룹 내 전용 채팅 기능으로 즉각적인 소통이 가능합니다.

- 협업 중 발생하는 변경 사항을 실시간으로 팀원에게 공유합니다.





## 문제 상황

- WebSocket은 하나의 연결로 모든 메시지를 처리하기 때문에, 채팅·일정·할일 등 기능별 메시지를 프론트에서 직접 분기·관리 해야 했음
- 기능이 늘어날수록 메시지 구조가 복잡해지고, 유지보수가 힘듬

## 해결 방안

- STOMP 프로토콜 기반 WebSocket으로 구조 전환
- 기능별로 독립된 구독 채널 구조 설계

## 결과

- 실시간 채팅, 일정, 할일 데이터 동기화 안정화
- 다중 채널 기반 구조로 확장성 강화 및 코드 가독성 개선

## 아쉬운 점

- 새로고침 시 일시적 재연결 문제 존재
- 불필요한 재연결 최소화 목표

```
const useWebSocket = (token, code, WEBSOCKET_URL) => {
  useEffect(() => {
    const client = new Client({
      brokerURL: WEBSOCKET_URL,
      connectHeaders: { Authorization: `Bearer ${token}` },
      reconnectDelay: 5000,
      onConnect: () => {
        // 일정, 플래너, 채팅을 각각 구독
        client.subscribe(`sub/schedule/${code}`, handleSchedule)
        client.subscribe(`sub/planner/${code}`, handlePlanner)
        client.subscribe(`sub/chat/${code}`, handleChat)
      }
    })
    client.activate()
    return () => client.deactivate()
  }, [token, code])
}
```

```
const handleSchedule = (message) => {
  const data = JSON.parse(message.body).data
  switch (data.action) {
    case 'CREATE':
      dispatch(addGroupSchedule(data))
      break
    case 'DELETE':
      dispatch(removeGroupSchedule(data.scheduleCommonResponse.id))
      break
  }
}
```



## 문제 상황

1. Moment.js: 288.1 KB (전체 번들의 55%)
2. Tree-shaking 불가능
3. Mutable API로 인한 사이드 이펙트 위험

```
● ○ ●  
const date = moment()  
date.add(1, 'day') // 원본이 변경됨
```

```
● ○ ●  
import { AdapterMoment } from '@mui/x-date-pickers/AdapterMoment'  
<LocalizationProvider dateAdapter={AdapterMoment}>  
  <DatePicker />  
</LocalizationProvider>
```

## 해결 방안

Day.js로 교체

- 크기: 2KB (Moment.js 대비 99% 감소)
- Moment.js와 거의 동일한 API
- MUI 공식 Adapter 제공

## 결과

- 288KB ⇒ 2KB 초기 로딩 속도 개선

```
● ○ ●  
import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs'  
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider'  
import { DatePicker } from '@mui/x-date-pickers/DatePicker'  
import dayjs from 'dayjs'  
  
const DatePickerCalendar = ({ value, onChange }) => {  
  const handleDateChange = date => {  
    onChange(date ? dayjs(date).format('YYYY-MM-DD') : '')  
  }  
  
  return (  
    <LocalizationProvider dateAdapter={AdapterDayjs}>
```

## 배운 점

- 라이브러리 선택이 번들 크기와 성능에 큰 영향을 준다는 것을 체감했다.
- 라이브러리 비교 및 선택에 대해 고민해 볼 수 있었다.



# MapleLink

| 프로젝트 소개

**프로젝트 기간** 2025.03 ~ 2025.06

**프로젝트 소개** MapleLink는 메이플스토리 유저들이 자신의 길드와 캐릭터를 한곳에서 관리하고 탐색 할 수 있도록 만든 통합 플랫폼입니다. Nexon Open API를 기반으로 길드와 캐릭터의 최신 데이터를 실시간으로 조회할 수 있으며, 다중 길드 검색, 멤버 정보 탐색 등 다양한 길드 관리 기능을 제공합니다. 또한, 게임 내 공식 이벤트 일정과 사용자의 개인 일정을 함께 캘린더를 통해 관리할 수 있습니다.

**인원** 2명(FrontEnd 1, BackEnd 1)

**기술스택** React, TailwindCSS, Tanstack Query, Zustand, dayjs

**URL** 배포 링크 : <https://maplelink.co.kr/>



# MapleLink

| 주제 선정 배경

The screenshot shows the homepage of the 'maplesstory partners for Developers' website. At the top, there's a navigation bar with links for 'Home', 'MaplesStory Partner?', 'Partner Application/FAQ', and 'Open API'. Below the navigation, a section titled 'Meet the MapleStory Partners!' lists several services:

- 환산 주스탯** (2454 likes): A service for calculating MapleStory character stats. Description: '환산 주스탯 서비스는 메이플스토리 게임의 "주스탯" 혹은 "전투력" 개념에는 반영되어 있지 않은 거의 모든 인게임 스페크 관련 수치를 일정 기준에 따라 계산하여, 해당 유저의...'. Buttons: '캐릭터 조회', '전투력 지표', '시뮬레이터', '최적화 도구', '명예의 전당', '스팩업 힌트'.
- 메애기** (1250 likes): A service for searching MapleStory character information. Description: '메이플 정보 검색은 메애기! 캐릭터의 장비, 코디, 스킬 등 다양한 정보와 지난 날짜들의 캐릭터 정보, 코디들을 확인해 보세요.''. Buttons: '캐릭터 조회', '코디 정보', '닉네임 생성기', '주화 시세', '직업 분석', '통계 정보'.
- 메이플지지** (922 likes): A service for providing MapleStory character information. Description: '2018년 오픈한 메이플지지는 오랜 기간 서비스 운영을 통해 누적된 다양한 통계 정보와 캐릭터 정보, 코디들을 확인해 보세요.''. Buttons: '캐릭터 조회', '통계 정보', '시뮬레이터', '코디 분석', '프로필 카드 제작'.
- 메이플로드** (871 likes): A service for sharing MapleStory character information. Description: '메이플로드 사이트는 많은 능력자분들께서 제작한 각각의 사이트를 한 곳에 모아 쉽게 확인할 수 있게 제작 되었고, 닉네임 검색을 통해 다양하게 확인할 수 있는 자체 제작 서...'. Buttons: '캐릭터 조회', '시뮬레이터', '링크모음', '제작자'.
- 츄츄지지** (490 likes): A service for sharing MapleStory character information. Description: '츄츄지지는 메이플스토리 유저들에게 전에 없던 새롭고 재밌는 플레이 경험을 제공하는 플랫폼입니다. 인게임에서 경험해보지 못한 새로운 재미...'. Buttons: '캐릭터 조회', '직업 분석', '강화 이력', '길드 조회', '캐릭터 목록', '시뮬레이터'.
- 메짱** (356 likes): A service for sharing MapleStory character information. Description: '메짱은 기존의 사이트들과는 차별화된 인게임 디자인을 채택하여 유저분들에게 친숙한 UI를 제공합니다.''. Buttons: '캐릭터 조회', '캐릭터 캡처', '길드 조회', '썬데이메이플', '팬덤 직업 뽑기'.
- 메이플히스토리** (351 likes): A service for sharing MapleStory character information. Description: '내 캐릭터는 그동안 얼마나 성장했을까? 메이플 히스토리에서 캐릭터/길드의 실시간 정보와 지난 기록들을 한눈에 알아보세요.''. Buttons: '캐릭터 조회', '길드 조회', '랭킹 정보', '강화 이력', '챗봇'.
- 메이플 서포트** (298 likes): A service for sharing MapleStory character information. Description: '나는 그동안 큐브에 얼마나 많은 네슨캐쉬를 썼을까? 그동안 얼마나 많은 파워엘릭서 교환권을 샀을까?...'. Buttons: '큐브학습', '큐브사용량'.

## 서비스 분석

넥슨 API 공개 이후 다양한 서비스들이 출시되었지만, 대부분 캐릭터 조회, 길드 정보, 확률 정보 등 정보 조회 중심

## 게임 유저의 불편함

- 엑셀 수기 관리: 길드원 최소 50명 이상의 정보를 여전히 엑셀로 수동 관리
- 본/부캐 파악 어려움: 누가 누구의 부캐인지 매번 물어봐야 함
- 반복적인 수작업: 매주 출석체크, 활동 기록 등을 손으로 작성
- 인원 변동 추적 곤란: 가입/탈퇴 기록이 남지 않아 히스토리 파악 불가



# MapleLink

핵심 기능

The screenshot displays two main sections of the MapleLink website:

**Top Section (Character Information):**

- Character Profile:** Shows a character named "단뱅" (DanBang) with level 286, job DeamonArcher, and a guild "길드: 뉴에서". It includes stats like HP: 500000, MP: 2191, INT: 2155, and LUK: 2155.
- Skills:** Lists skills like "어빌리티" (Ability) and "스탯" (Stat).
- Abilities:** Lists abilities like "스킬 사용 시 20% 확률로 재사용 대기..." (Skill usage has a 20% chance to cooldown immediately after use).
- States:** Lists states like "상태 이상에 걸린 대상 공격 시 데미지 8..." (When affected by status, attack damage is increased by 8%).
- Bosses:** Lists bosses like "보스 몬스터 공격 시 데미지 10% 증가" (Boss monster attack damage increases by 10%).

**Bottom Section (Guild Search):**

- Guild Search Form:** A search bar where "루나" is typed, with a placeholder "길드 이름을 입력하세요" (Enter guild name). Buttons include "추가" (Add) and "검색하기" (Search).
- Search Results:** Shows a list of guilds: "루나" and "소월".

## 1. 캐릭터 & 길드 검색

- 기본 정보 (레벨, 직업, 길드, 월드)
- 스탯 정보 (전투력, 공격력, 방어력 등 40+ 항목)
- 어빌리티 (3개 프리셋)
- 하이퍼 스탯 (3개 프리셋)
- 장비 아이템 (3개 프리셋, 상세 옵션 포함)

## 2. 길드 검색 시스템

- 길드 기본 정보 (마스터, 부마스터, 길드원 수, 레벨)
- 길드원 목록 및 본캐/부캐 자동 구분
- 최대 4개 길드 동시 비교



# MapleLink

핵심 기능

The screenshot shows the 'Guild Management' section of the MapleLink interface. It displays a list of characters under the 'Main' category, including: 단뱅 (Lv.286), 노침 (Lv.290), 비글 (Lv.290), 맵백 (Lv.290), 몽맞춰 (Lv.260), and 비글제논 (Lv.285). A dropdown menu is open over the 'Main' category, showing options like '+ Guild Creation', 'Character Selection', 'Main/Sub Status', and 'Compare'. The top navigation bar includes links for 'Character Information', 'Guild Management', 'Calendar', and 'Guild Announcements'.

Two detailed character profiles are shown side-by-side. The left profile is for '단뱅' (Lv.286), a Demon Archer, with tabs for 'Basic Information' and 'Sub Character'. The right profile is for '소울' (Lv.270), a Monk, with tabs for 'Basic Information' and 'Sub Character'. Both profiles show character portraits, names, levels, and guild information.

### 3. 길드 관리 시스템

- Room 생성: 그룹명 지정 후 메인 길드 등록
- 서브 길드 추가: 동맹 길드나 여러 캐릭터 길드 관리
- 길드원 변동 감지: DB와 게임 내 데이터 비교하여 신규/탈퇴 자동 탐지
- 본캐/부캐 구분: 넥슨 API로 메인 캐릭터 자동 판별
- 멤버 상세 정보: 각 길드원의 직업, 레벨, 인기도 등 조회



## 문제 상황

초기에는 캐싱 없이 매번 API를 직접 호출하는 방식으로 개발함

- 로딩 시간 과다: 페이지 이동마다 매번 API 호출로 3~5초 대기
- Nexon API Rate Limit: 분당 호출 횟수 제한으로 잣은 429 에러 발생
- 사용자 불편: 같은 캐릭터 정보를 볼 때마다 로딩
- 불필요한 네트워크 비용: 변경되지 않은 데이터도 반복 요청

## 해결 방안

- TanStackQuery 사용
- 캐릭터 스탯/인벤토리: 5분 (변경 빈도 낮음)
- 길드 멤버 목록: 10분 (가입/탈퇴)
- 검색 결과: 10분 + refetchOnWindowFocus: false (정적 데이터)

## 결과

- 불필요한 API 호출 70% 감소
- Nexon API Rate Limit 회피
- 사용자는 로딩 없이 즉시 데이터 확인 가능

```
const guildsQuery = useQuery({
  queryKey: ['guildsInfo', guildList, server],
  queryFn: () => searchGuildWithoutLogin(guildList, server),
  enabled: enabled && guildList.length > 0 && !server,
  retry: false,
  staleTime: 1000 * 60 * 10,
  refetchOnWindowFocus: false
})
```



## 문제 상황

길드 멤버 정보를 갱신할 때, 단순히 API 재호출을 전체 데이터에 적용하면 불필요한 네트워크 비용이 발생.

- 전체 재호출의 비효율: 한 멤버의 50명의 정보를 조회하면 5~10초 소요
- 사용자 경험 저하: 사용자는 멤버 목록을 완전히 다시 기다려야 함

## 해결 방안

두 가지 접근 방식

### 1. 낙관적 업데이트 (Optimistic Update)

- UI를 먼저 변경하고 나중에 서버와 동기화
- 실시간 반응성이 중요한 곳에 적합 (좋아요, 댓글 등)
- 하지만 실시간성이 중요하지 않은 길드 시스템에는 과도한 복잡도

### 2. 부분 캐시 갱신 (선택한 방법)

- onMutate에서 setQueryData를 사용해 변경된 멤버만 업데이트
- API 호출 최소화하면서도 데이터 일관성 유지
- 사용자는 즉시 변경사항 확인 가능



```
queryClient.setQueryData(['guildsInfo', guildList, server], (oldData) =>
{ return oldData.map(guild => ({
  ...guild,
  guildMember: guild.guildMember.map(member => {
    const match = response.find(res => res.memberName === member.name)
    return { ...member, ...match } // 즉시 병합
  })
}))})
```

## 결과

- API 호출 횟수 대폭 감소 - UI 변경 시 즉시 반영, 검증만 서버 통신
- 실시간 업데이트 느낌 - 사용자는 즉각적인 반응 체험 (5~10초 → 0초)
- 데이터 일관성 유지 - onError로 실패 시 자동 룰백
- 네트워크 최적화 - 변경된 데이터만 처리



**프로젝트 기간** 2025.09 ~ 2025.11

**프로젝트 소개** 웹 서핑 중 마음에 드는 뉴스나 블로그를 발견할 때마다 매번 메모장이나 카카오톡에 저장해야 하는 불편함을 해결하고자 시작한 프로젝트입니다. 사용자는 크롬 확장 프로그램을 통해 클릭 한 번으로 콘텐츠를 수집하고, AI 요약 기능으로 핵심 내용을 자동 정리할 수 있습니다. 이렇게 저장된 정보는 플랫폼 내에서 주제별로 분류·정리되며, 팀 단위로 공유하고 브레인스토밍할 수 있는 협업형 지식 관리 공간으로 확장됩니다.

**인원** 8명(FrontEnd 3, BackEnd 5)

**기술스택** Next.js, Liveblocks, XYFlow(React Flow), TailwindCSS, Tanstack Query, Zustand, Shadcn/UI

**URL** GitHub : <https://github.com/Team-ZoopZoop>

개인 리펙토링 : <https://github.com/play3step/FE-ZoopZoop-Refactor>

시연 영상 : <https://youtu.be/8ofrx60MaL4>



# ZOOOPZOOOP

주제 선정 배경

## 1. 자료 수집의 비효율성

→ 블로그·뉴스 등 관심 콘텐츠를 매번 복사해 노션이나 카톡에 따로 저장해야 함

## 2. 제목, URL만 저장되는 북마크

→ 링크만 남아 내용이 기억나지 않거나, 미리보기조차 제공되지 않음

## 3. 정보 파악의 어려움

→ 어떤 이유로 저장했고, 왜 중요했는지 한눈에 파악하기 어려움

## 4. 공유 및 협업의 단절

→ 팀원과 유용한 자료를 쉽게 공유하거나 함께 정리하기 힘듦



# ZOOOPZOOOP

자료 수집·정리부터  
팀 공유·브레인스토밍까지 연결되는  
지식 관리 협업 플랫폼

북마크 검색

- [취준] 중소기업 최종 합격 후기
- 우리 팀의 성장을 이끈 협업 문화와 프로세스 개선 후기
- 프론트엔드 개발자 - 원종석 프로필

### • 북마크 활용

## 자료

Redux

- React 입문 - react-redux 리덕스 사용법
- [Redux] Redux 사용법 기초 useState처럼 사용하기
- [React] 리액트에서 Redux 써보기
- React 상태관리 라이브러리 Redux의 설명과 사용법

Redux

act": "^17.0.2",  
act-dom": "^17.0.2",  
act-redux": "^8.0.0",  
eduxjs/toolkit": "^1.8.1"

리액트

풀더 구조

- React의 풀더구조

### • 노션 활용



ZOOOPZOOOP

| 핵심 기능

## 1. 개인 아카이브

- 웹에서 발견한 자료를 한 번의 클릭으로 개인 아카이브에 저장
- 크롬 확장프로그램 또는 URL 입력만으로 손쉽게 추가 가능
- AI가 자동으로 요약, 분류, 태그 추출하여 체계적으로 정리
- 폴더별로 주제나 프로젝트 단위의 맞춤형 아카이브 구성 가능



ZOOPZOOP

| 핵심 기능

## 2. 공유 아카이브 & 대시보드

- 팀원들과 공유 가능한 협업형 아카이브 공간 제공
- 개인 아카이브의 자료를 손쉽게 공유하고 팀별 스페이스로 관리
- 대시보드에서 아카이브 간의 연관 관계 시각화 및 아이디어 브레인스토밍
- 실시간 협업 및 의견 교환을 통한 지식 확장 가능

The image displays two screenshots of the ZoopZoop platform interface. The left screenshot shows the 'Space' section, where users can upload files to shared spaces. It includes a search bar, a file selection area, and a preview of selected files. The right screenshot shows the 'Dashboard', which features a timeline of news items and a network graph showing connections between different spaces.



ZOOPZOOP

| 핵심 기능

### 3. AI 추천

- AI가 아카이브 내 태그와 관심사를 분석해 맞춤형 뉴스 및 자료 추천
- 폴더 단위로 관련 콘텐츠를 자동 추천하여 지속적인 정보 확장
- 팀 스페이스에서는 공통 관심사 기반 뉴스 피드 제공

The screenshot shows the ZoopZoop application interface. On the left is a sidebar with a user profile icon, 'Zoop's Today' button, and a 'News' tab which is currently selected. Other tabs include 'Archive', 'Space', 'Usage Methods', and 'Settings'. The main content area is titled 'AI Recommended News' and contains a message: '저장한 폴더를 분석하여 관심사에 맞는 뉴스를 추천합니다.' Below this is a 'Category Filter' section with 'ai' selected and other options like '기본 폴더', '넥슨', and '삼성'. There are three news cards displayed:

- ORACLE + xAI** (2025.06.20. 12:16)  
일론 머스크가 세운 xAI, OCI 선다...  
일론 머스크 대표이자 최고경영자(CEO)가 세운 인공지능(AI) 기업 xAI가 오라클의 AI 인프라 활용을 통해... OCI 베이메탈 GPU 인스턴스는 생성형...
- ORACLE + xAI** (2025.06.20. 09:31)  
일론 머스크, 오라클과 AI 인프라 통...  
일론 머스크 대표이자 최고경영자(CEO)가 세운 인공지능(AI) 스타트업 xAI가 오라클과의 협업을 발표했다... OCI 베이메탈 GPU 인스턴스는 생...
- [인터뷰] 마이크 히츠와 오라클 부사...** (2024.09.12. 01:00)  
업계에서 인공지능(AI)을 활용한 로우코드-노코드 개발 플랫폼이 주목받고 있다. 복잡한 코딩 필요 없이... 이런 트렌드 속 오라클은 자체 로우코드...

A large banner at the bottom of the screen displays the text '42MA 42MA'.



## 문제 상황

- 사용자 A가 노드를 선택하거나 드래그하면 사용자 B의 화면도 같이 선택됨
- 한 사용자가 노드를 조작하는 동안 다른 사용자는 아무것도 할 수 없음
- 네트워크 사용량, 및 사용자 리렌더링 60fps ⇒ 15fps로 저하

## 원인 분석

- 선택 상태가 공유 Storage에 저장
  - 드래그 중 과도한 Storage 업데이트  
⇒ 드래그 1초당 = position 업데이트 60회(60fps)
- Liveblocks 서버 부하 + 네트워크 트래픽 증가 = UI 블로킹 및 버벅임

## 해결 방안

데이터를 성격에 따라 3가지 계층으로 분리

Local State(개인)	Presence(실시간, 임시)	Storage(영구)
• 각 사용자 독립	• 빠른 업데이트	• 영구 저장
• 공유 불필요	• 연결 끊기면 사라짐	• 최종 노드 위치
• 선택 상태	• 드래그 중 위치, 커서	

```
const handleNodesChange: OnNodesChange = changes => {
  // 1. 선택 → 로컬만
  const select = changes.filter(c => c.type === 'select')
  if (select.length) {
    setSelectedIds
  }

  // 2. 드래그 중 → Presence
  const dragging = changes.filter(
    c => c.type === 'position' && c.dragging
  )
  dragging.forEach(change => {
    updateDraggingNode(change.id, change.position)
    setMyDraggingPosition({ nodeId: change.id, position: change.position })
  })

  // 3. 드래그 끝 → Storage (1회만)
  const finished = changes.filter(
    c => c.type === 'position' && !c.dragging
  )
  if (finished.length) {
    onNodesChange(finished)
    updateDraggingNode(null)
    setMyDraggingPosition(null)
  }
}
```

## 결과

- 10초간 Storage 업데이트 600회 ⇒ 10회
- devtool(프레임 렌더링) 20fps ⇒ 60fps
- 동시 작업 가능인원 증가



## 문제 상황

1. SSR 후 CSR 전환 시 불필요한 재요청으로 화면 깜빡임
2. Query key 하드코딩으로 인한 캐시 미스(invalidateQuerys 등)
3. Next.js 캐싱과 TanStack Query 캐싱 충돌로 데이터 불일치

## 1. Hydration Boundary를 통한 SSR/CSR 캐시 동기화

### 접근 방안

SSR에서 미리 쿼리를 Prefetch하고,  
HydrationBoundary를 통해 해당 캐시를 CSR로 전달해 데이터 일관성 유지

### 결과

- CSR 재요청 제거 → 화면 깜빡임 해소
- SSR 데이터와 CSR 데이터 완전 동기화
- 초기 렌더링 속도 향상 및 캐시 일관성 확보

```
● ● ●  
export default async function Page() {  
  const data = await fetchData()  
  return <Component initialData={data} /> // 첫 마운트만 사용  
}
```



```
● ● ●  
export default async function Page() {  
  const queryClient = new QueryClient()  
  
  await queryClient.prefetchQuery({  
    queryKey: QUERY_KEYS.USER.me(),  
    queryFn: fetchUserProfile  
  })  
  
  return (  
    <HydrationBoundary state={dehydrate(queryClient)}>  
      <Component />  
    </HydrationBoundary>  
  )  
}
```



## 2. Query Key 통합 관리 & 캐싱 충돌 해결

### 문제 상황

- Query Key가 하드코딩되어 invalidate 시 일관되지 않음
- 캐시 무효화 시 어느 쪽이 우선인지 명확하지 않아 데이터 불일치 발생
- Next.js의 Server Cache와 TanStack의 Client Cache가 따로 존재

### 접근 방안

- Query Key 중앙 관리 시스템 구축
- 이중 캐싱 레이어 전략 수립
  - Server Cache (Next.js) → revalidateTag()
  - Client Cache (TanStack) → invalidateQueries()

```
export const QUERY_KEYS = {
  USER: {
    me: () => ['user', 'me']
    byName: (name: string) => ['user', name]
  },
  SPACE: {
    all: () => ['spaces'],
    pagination: (page: number, state?: SpaceStatus)
      => ['spaces', page, state]
  },
  // 8개 도메인으로 체계화
}
```

```
// 캐시 무효화 + 즉시 재렌더링
onSuccess: async () => {
  await queryClient.invalidateQueries({ queryKey: QUERY_KEYS.SPACE.all() })
  await revalidateSpaceList()
  closeModal()
  router.refresh()
}
```

- router.refresh()

```
const { mutate } = useMutation({
  mutationFn: createSpace,
  onSuccess: async () => {
    await queryClient.invalidateQueries({ queryKey: QUERY_KEYS.SPACE.all() })
    await revalidateSpaceList() // Server 캐시도 무효화
  }
})
```

- revalidateTag()

D

Deemo

| 프로젝트 소개

**프로젝트 기간** 2025.12 ~ 진행중

**프로젝트 소개** 단순 나열식 포트폴리오를 벗어나, 인터랙티브한 사용자 경험을 제공하는 Mac OS 컨셉의 개인 사이트입니다.

**인원** 개인 프로젝트

**기술스택** Next.js, TailwindCss, Zustand, Framer Motion, Gemini API

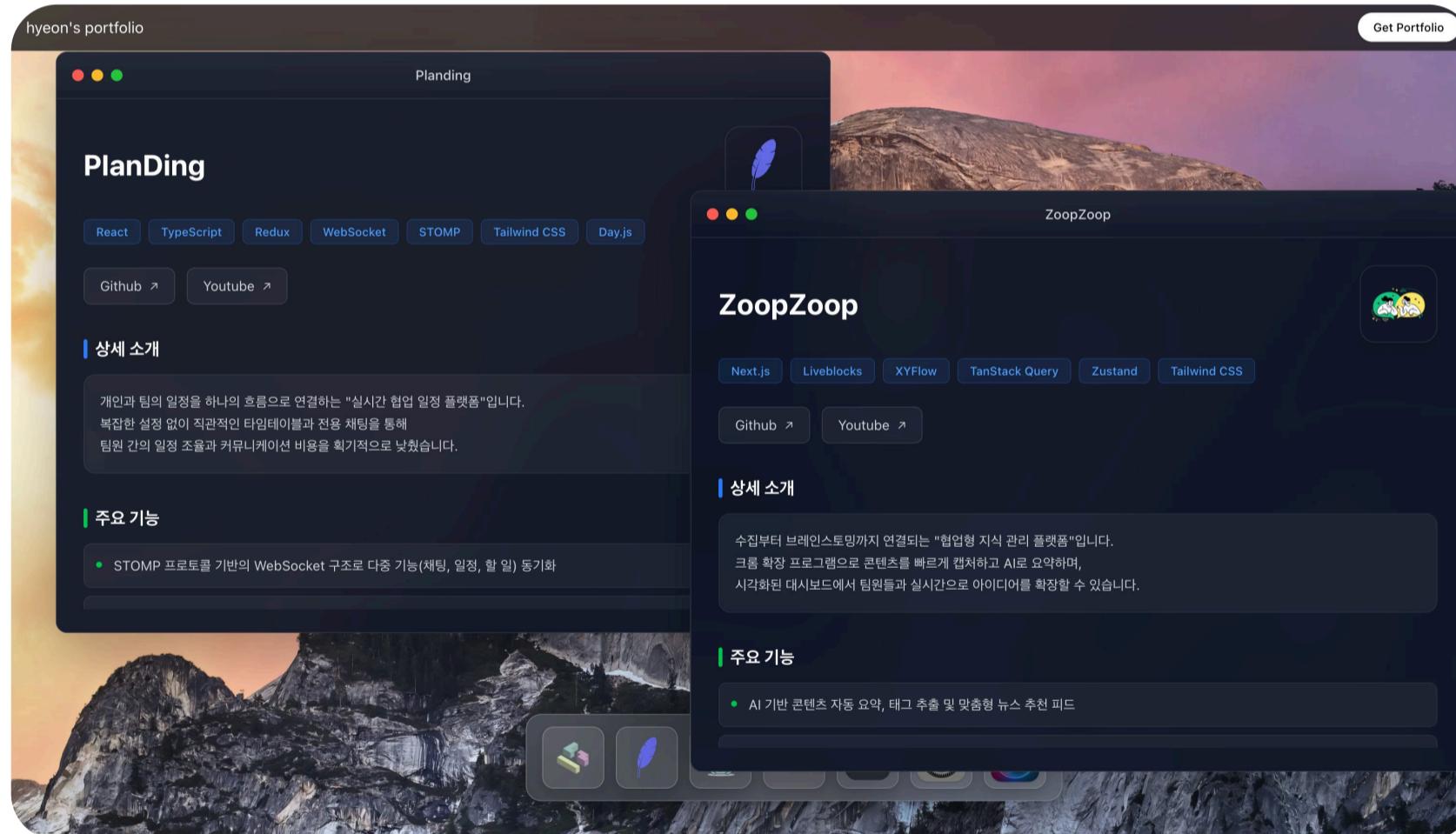
**URL** GitHub : [https://github.com/play3step/portfolio\\_web\\_site](https://github.com/play3step/portfolio_web_site)

배포 링크 : <https://www.deemo.dev/>



Deemo

| 핵심 기능



- 멀티 윈도우 탐색

여러 프로젝트 창을 동시에 띄워 놓고 내용을 비교하거나 자유롭게 탐색할 수 있습니다.

- 자유로운 윈도우 배치

실제 OS처럼 각 창을 원하는 위치로 드래그하여 나만의 작업 공간을 만들 듯 포트폴리오를 감상할 수 있습니다.

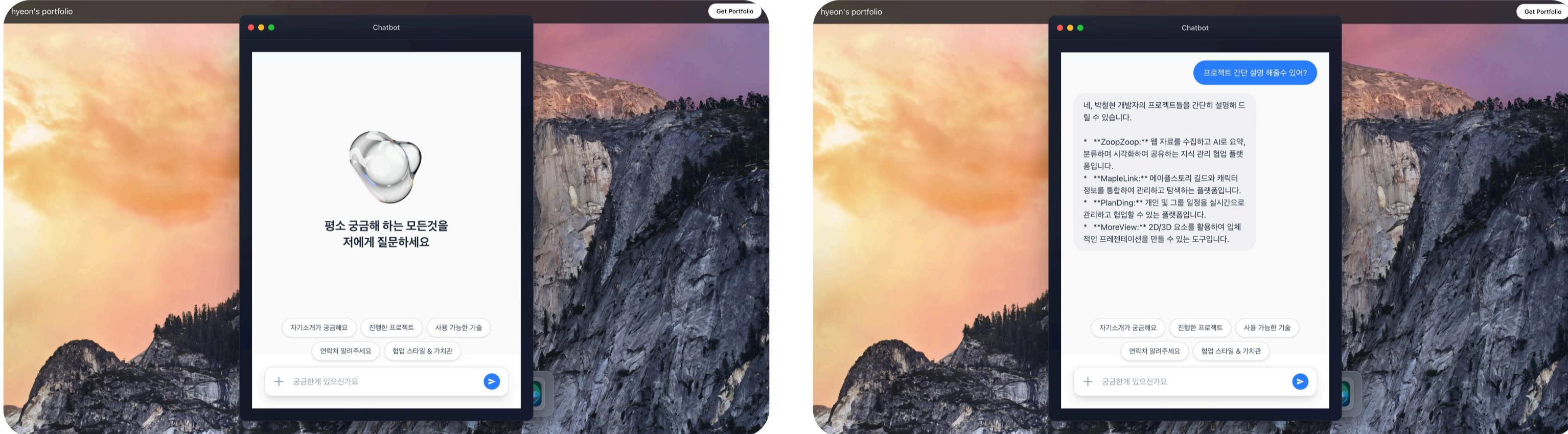
- 통합 프로젝트 도슨트

각 프로젝트의 핵심 소개와 기술 스택을 별도의 페이지 이동 없이 한 화면에서 즉시 확인할 수 있습니다.

**D**

Deemo

핵심 기능

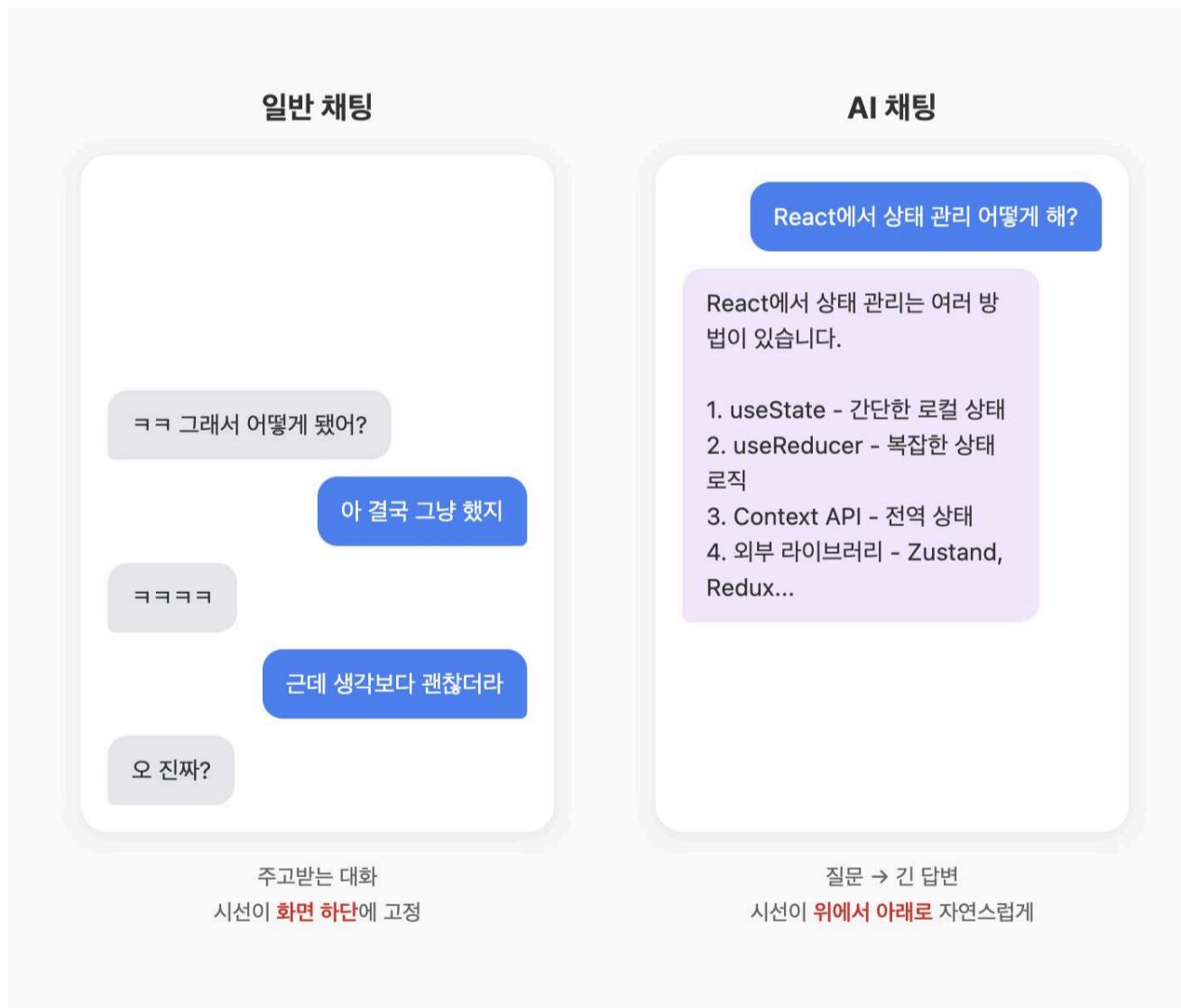


### 선택 (Select): 원클릭 퀵 답변 시스템

- 사용자가 고민하지 않고도 핵심 정보를 찾을 수 있도록 '자기소개', '진행 프로젝트', '사용 기술' 등을 하단 버튼으로 배치했습니다.
- 버튼 클릭 한 번으로 미리 정의된 최적의 답변을 즉시 얻을 수 있어 탐색의 피로도를 낮춥니다.

### 대화 (Chat): Gemini 기반 실시간 질의응답

- Gemini API를 통해 사용자의 자연어 질문에 실시간으로 답변하는 지능형 인터페이스를 구현했습니다.
- 방문자가 채팅창에 궁금한 점을 직접 입력하면, AI가 맥락을 분석해 관련 정보를 정확하게 안내합니다.



## 1. 문제 상황 (Problem)

본질적 차이: 일반 채팅(Bottom-Up)과 달리 AI 답변(Top-Down)은 위에서 아래로 읽는 흐름을 가짐.

가독성 저해: 기존 하단 고정 스크롤 방식 사용 시, 스트리밍 답변이 길어지면 질문(맥락)이 상단으로 밀려나 사용자가 답변 시작점을 놓치는 문제 발생.

## 2. 해결 (Solution)

상단 고정 로직: 사용자 질문을 화면 상단에 고정하고 답변만 아래로 흐르게 설계하여 시선 흐름을 일치시킴.

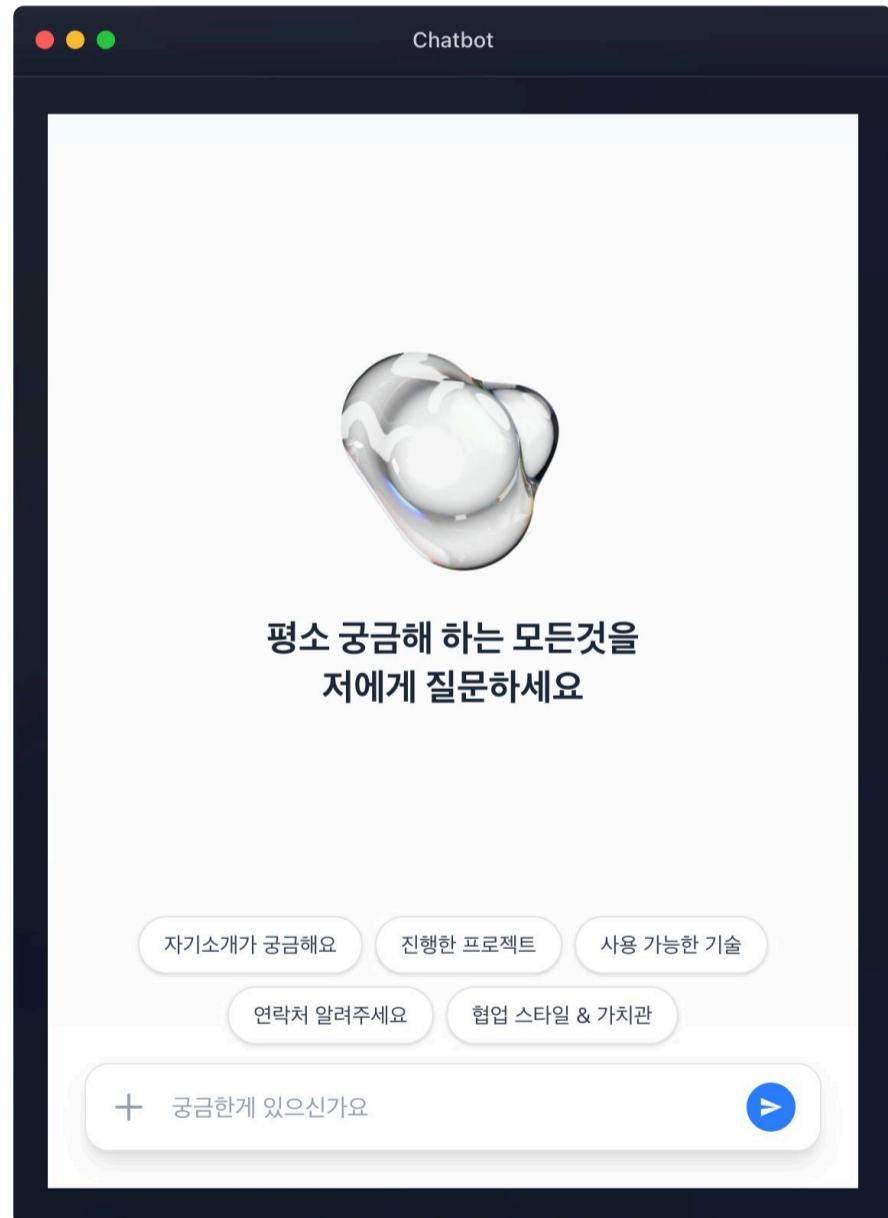
여백 기법: 콘텐츠가 짧아 스크롤이 생기지 않는 경우에도 하단에 가상의 공간을 계산·삽입하여 항상 상단 고정 UX를 유지함.

실시간 레이아웃 추적: ResizeObserver를 활용해 스트리밍 중인 답변의 높이 변화를 실시간으로 감지하고 스크롤 위치를 정밀하게 보정함.

## 3. 성과 (Result)

맥락 유지: 답변이 길어져도 질문이 항상 상단에 노출되어 대화의 맥락을 놓치지 않음.

자연스러운 UX: 사용자가 읽기 시작하는 시점과 AI 답변의 시작점을 일치시켜 정보 습득 효율을 극대화함.



"제한된 자원을 보호하는 3단계 API 방어 및 최적화 체계"

### 1. 문제 상황 (Problem)

리소스 한계: Gemini API 무료 티어의 일일 요청 제한(Quota)으로 인한 서비스 중단 우려.  
부적절한 요청: 포트폴리오와 무관한 질문이나 매크로 공격으로 인한 API 자원 고갈 위험.

### 2. 해결 방안 (Solution)

- Step 1. 정적 가드레일: API 호출 전 로직에서 부적절한 키워드를 1차 검사하여 불필요한 호출 비용 0원 차단.
- Step 2. 멀티 모델 폴백: 특정 모델 한도 초과 시 Flash → Flash-lite 순으로 자동 전환하여 서비스 연속성 확보.
- Step 3. Upstash Rate Limit: Redis를 활용해 IP당 호출 횟수(4회/1분)를 제한하여 리소스 독점 및 봇 공격 방지.

### 3. 성과 (Result)

가용성 극대화: 비정상 요청을 사전에 차단하여 실제 방문자를 위한 API 할당량을 상시 확보.  
안정적 운영: 별도의 인프라 비용 지출 없이 무료 티어만으로 중단 없는 챗봇 서비스 운영 성공.