

FRONTEND DEVELOPER PORTFOLIO

강민재

프론트엔드 개발자

010-5519-7425

towndev@naver.com

<https://minjaekang.dev/>

소개

대학교를 휴학하고 창업에 도전하며 서비스를 직접 구현하기 위해 개발을 시작했습니다. 이후 오토바이 정비·라이딩 플랫폼 라이더타운과 협업 여행 플랫폼 저스트트립을 기획하고, React·Next.js·React Native 기반의 웹과 앱으로 개발해 출시·운영했습니다.

프론트엔드부터 백엔드, 인프라까지 직접 구축해 본 풀스택 경험을 바탕으로 시스템 전체의 데이터 흐름을 폭넓게 이해하며, 깊이 있는 소통과 집요한 실행력으로 조직과 서비스의 성장에 기여하겠습니다.

기술 역량

- Frontend: React, Next.js, TypeScript, Redux, React-query, Tailwind CSS
- Mobile: React Native, Expo, React Native Webview
- Backend: Java (Spring Boot), Node.js (Express), RESTful API 설계
- Database & Cache: MySQL, PostgreSQL, Redis, Firestore
- Cloud & DevOps: AWS, Docker, GCP, Firebase, Vercel, Git-flow 기반 협업

목차

1. 저스트트립 - AI 기반 크로스플랫폼 협업 여행 일정/기록 서비스

- CASE 01** 낙관적 삭제와 서버 복제 지연으로 되살아나는 항목
- CASE 02** 여행 상세 화면 First Load JS 890kB → 452kB
- CASE 03** Document AI로 바우처·영수증 자동입력
- CASE 04** 비동기 경쟁 상태: 취소할까, 무시할까, 공유할까
- CASE 05** 비행기 모드에서도 열리는 오프라인 모드

2. 라이더타운 V2 - 라이딩 기록 서비스 피벗 & 서버리스 전환

- CASE 06** 화면이 꺼져도 끊기지 않는 주행 기록

3. 라이더타운 V1 - 오토바이 정비 플랫폼 창업·라이딩 기록 및 그룹 음성 통화

- CASE 07** 웹뷰 하이브리드 앱의 느린 첫 화면과 네이티브 연동

저스트트립

AI 기반 크로스플랫폼 협업 여행 일정/기록 서비스

친구·가족과 함께 여행을 계획하고 기록하는 협업 여행 서비스입니다. 일정·항공·숙소·바우처·지출·체크리스트·사진을 한곳에서 관리하고, Next.js 웹과 Expo 앱으로 함께 출시했습니다.

기간	2026.03 - 2026.10 (8개월)
팀	백엔드 개발자 1명과 2인 협업
역할	서비스 기획 · 프론트엔드 개발
담당	서비스 기획, 도메인·API 명세 작성, 디자인 시스템, 웹·앱 프론트엔드 전담
기술	Next.js, React Native (Expo), TypeScript, TanStack Query, Redux Toolkit, Axios, SQLite, Azure Container App, MCP

[Web ↗](#) [App Store ↗](#) [Google Play ↗](#)

이 프로젝트의 문제 해결 사례

- **CASE 01** 낙관적 삭제와 서버 복제 지연으로 되살아나는 항목
- **CASE 02** 여행 상세 화면 First Load JS 890kB → 452kB
- **CASE 03** Document AI로 바우처·영수증 자동입력
- **CASE 04** 비동기 경쟁 상태: 취소할까, 무시할까, 공유할까
- **CASE 05** 비행기 모드에서도 열리는 오프라인 모드

낙관적 삭제와 서버 복제 지연으로 되살아나는 항목

삭제 상태를 대기·전송 중·최근 커밋 3단계로 나눠 지운 항목이 다시 나타나는 문제를 없애고, 연속 삭제 시 재조회를 1회로 줄였습니다.

1회

연속 삭제 시 재조회

삭제할 때마다 → 마지막에 한 번

7개

도메인 공통 적용

웹·앱 동일 로직

0

백엔드 변경

클라이언트에서 해결

문제 동행자가 함께 편집하는 여행에서 항목을 지우면 바로 다음 목록 응답에 지운 항목이 다시 나타났습니다. 사진 여러 장처럼 연속으로 지우면 삭제할 때마다 목록 전체를 다시 불러왔습니다.

원인 삭제 요청이 성공한 직후의 조회 응답에 서버 복제 지연으로 옛 데이터가 섞여 왔고, 삭제마다 캐시를 무효화하고 있었습니다.

고민 삭제 UX를 어떻게 할까

기각 확인 다이얼로그 사진 수집 장을 지울 때마다 확인을 눌러야 해서 피로했습니다.

기각 즉시 삭제 후 실행 취소 시 다시 생성 새 ID가 발급돼 일정·지출·바우처와의 연결이 끊어졌습니다.

채택 화면에서 먼저 지우고 4초 뒤 서버로 보내는 지연 커밋 실행 취소가 자연스럽게 ID가 유지됩니다.

되살아나는 항목을 어떻게 막을까

기각 커밋 후 재조회를 몇 초 늦추기 지연 시간을 추측해야 하고, 복제가 더 늦으면 같은 문제가 재발합니다.

보류 서버에 버전 필드나 soft delete 요청 백엔드 일정에 묶여 당장 적용할 수 없었습니다.

채택 삭제한 ID를 기억하고 모든 목록 응답에서 걸러내기 클라이언트만으로 바로 적용할 수 있었습니다.

연속 삭제와 탭 종료는 어떻게 처리할까

기각 debounce로 재조회 묶기 커밋이 4초씩 엇갈리면 중간에 재조회가 생겼습니다.

기각 sendBeacon으로 남은 삭제 전송 POST만 가능하고 Authorization 헤더를 붙일 수 없습니다.

채택 남은 작업이 0개일 때만 재조회 + fetch keepalive 재조회는 1회로 줄고, 탭을 닫아도 삭제가 유실되지 않습니다.

해결 삭제 상태를 대기(실행 취소 가능)·전송 중·최근 커밋(20초)의 3단계로 관리했습니다. 재조회는 세 시점 어디에나 끼어들 수 있어서, 대기 큐만 두면 전송 중에 도착한 응답에 항목이 되살아났기 때문입니다. pagehide·beforeunload 시 남은 삭제는 fetch keepalive로 일괄 전송했습니다.

성과 삭제한 항목이 되살아나는 문제가 사라졌고, 연속 삭제 시 재조회가 1회로 줄었습니다. 일정·숙소·항공·바우처·지출·체크리스트·사진 7개 도메인에 웹·앱 공통으로 적용했습니다.

기술 TypeScript, TanStack Query, fetch keepalive, Next.js, React Native (Expo)

여행 상세 화면 First Load JS 890kB → 452kB

사용자 행동이 있어야 보이는 다이얼로그와 탭만 골라 분할하고, 첫 클릭 지연은 유희 시간 프리페치로 보완해 초기 JS를 49% 줄였습니다.

49%

First Load JS 감소

890kB → 452kB

18곳

다이얼로그·탭 분할

next/dynamic + LazyMount

문제 여행 상세 화면에 다이얼로그 11개와 탭 콘텐츠가 한 번에 묶여 First Load JS가 890kB였고, 지도를 움직일 때마다 일정 목록까지 다시 렌더링됐습니다.

고민 무엇을 나누고, 어떻게 마운트할까

기각 next/dynamic만 적용 닫혀 있어도 렌더 트리에 있으면 청크를 내려받았습니다.

기각 열 때 마운트, 닫을 때 언마운트 닫힘 애니메이션이 끊기고 입력하던 내용이 사라졌습니다.

채택 처음 열릴 때 마운트하고 이후 유지하는 LazyMount 청크는 필요할 때만 받고 상태와 애니메이션은 유지됩니다.

분할로 생기는 첫 클릭 지연을 어떻게 줄일까

기각 hover 시점 프리페치 모바일에는 hover가 없습니다.

채택 첫 로딩 후 requestIdleCallback 프리페치 미지원 브라우저(Safari)는 2초 지연 setTimeout으로 대체했습니다.

memo를 어디에 걸까

기각 모든 컴포넌트에 memo 비교 비용만 늘고 효과는 불분명했습니다.

채택 지도 조작 시 다시 그려지던 일정 목록에만 memo + props 메모이즈 매 렌더마다 새로 생기는 콜백·배열을 고정해야 memo가 동작합니다.

해결 다이얼로그와 탭 콘텐츠 18곳을 next/dynamic과 LazyMount로 분할하고, 일정 목록에 React.memo와 콜백·배열·객체 메모이즈를 적용했습니다. 낱자 유틸이 dayjs 전체를 간접적으로 끌어오던 의존성도 끊었습니다.

성과 /travel/[id] First Load JS를 890kB에서 452kB로 49% 줄였고, 지도 조작 시 일정 목록 재렌더링을 없앴습니다.

기술 Next.js (next/dynamic), React.memo, requestIdleCallback, TypeScript

Document AI로 바우처·영수증 자동입력

AI 분석 결과가 틀릴 수 있다는 전제로 업로드부터 폼 입력까지의 흐름을 설계해, 항공·숙소 입력을 파일 업로드 한 번으로 줄였습니다.

95%

바우처 파싱 정확도

80%

다구간 등록 시간 단축

90% ↑

가계부 입력 공수 절감

문제 항공 10개, 숙소 6개, 지출 10개 항목을 일일이 수기로 입력해야 했고, 영어나 현지 언어로 된 PDF 바우처와 영수증을 보며 입력하다 오타와 입력 이탈이 잦았습니다.

고민 AI 결과를 얼마나 믿을까

기각 분석 결과를 그대로 저장 OCR 결과는 틀릴 수 있어 잘못된 일정이 그대로 등록됩니다.

채택 값이 있는 항목만 폼에 채우고 사용자가 확인 후 저장 입력 공수는 줄이면서 최종 판단은 사용자가 합니다.

결과가 여러 개일 때 어떻게 보여줄까

기각 항상 같은 폼 사용 경우 항공편이나 여러 숙소처럼 결과가 많으면 확인하기 어렵습니다.

채택 결과 개수에 따라 단일 폼 / 일괄 등록 모드 자동 전환 실패한 파일만 다시 분석할 수 있습니다.

추출한 장소를 어떻게 매칭할까

기각 이름만으로 구글 장소 검색 다른 도시의 같은 이름 숙소가 잡혔습니다.

채택 "이름 + 주소 → 주소" 순으로 매칭 동명 숙소 오인을 막습니다.

해결 백엔드 개발자와 Azure AI Document Intelligence 추출 스키마를 함께 설계하고, 업로드부터 폼 입력까지를 맡았습니다. 업로드 전에 파일 개수·크기를 검증하고 아이폰 HEIC는 JPEG로 변환했으며, 앨범 다중 선택·PDF·카메라 촬영 세 가지 입력을 지원했습니다.

성과 항공·숙소 입력은 파일 업로드 한 번, 지출은 업로드 후 3개 항목 선택으로 줄였습니다.

기술 Azure AI Document Intelligence (백엔드 협업), Next.js, React Native (Expo), libheif-js, TypeScript

비동기 경쟁 상태: 취소할까, 무시할까, 공유할까

자동완성 결과 역전, 중복 토큰 갱신, 미디어 권한 중복 발급, "참가자가 아님" 깜빡임을 상황에 맞는 방식으로 각각 해결했습니다.

1회

동시 401에도 토큰 갱신

Axios 인터셉터 대기 큐

1회

사진 수백 장 권한 발급

진행 중인 Promise 공유

문제 늦게 도착한 이전 검색 결과가 최신 결과를 덮어썼고, 동시에 401을 받은 요청 수만큼 토큰 갱신이 일어났습니다. 사진 수백 장이 동시에 미디어 권한을 요청했고, 새로 만들거나 참가한 여행에 들어가면 "참가자가 아님" 화면이 잠깐 떴습니다.

고민 장소 자동완성

기각 debounce만 적용 응답 순서를 보장하지 못하고, 캐시 결과는 즉시 돌아와 순서가 더 쉽게 뒤집혔습니다.

기각 AbortController로 취소 Google Maps SDK가 요청 취소를 지원하지 않습니다.

채택 요청 순번으로 오래된 응답 무시 취소 없이도 최신 결과만 반영됩니다.

토큰 갱신과 미디어 권한

기각 요청마다 각자 갱신·발급 동시 요청 수만큼 갱신과 발급이 반복됐습니다.

채택 갱신은 1회, 나머지는 큐 대기 / 진행 중인 Promise 공유 만료 2분 전 선제 재발급으로 로딩 중 403도 막았습니다.

"참가자가 아님" 깜빡임

기각 단건 조회 API 추가 백엔드 작업과 요청 수가 늘어납니다.

기각 생성 직후 캐시에 직접 삽입 다른 기기에서 참가한 경우를 막지 못합니다.

채택 "캐시에 없음"과 "권한 없음"을 구분하고 1회 재조회 후 판단 ref로 무한 재조회도 막았습니다.

해결 자동완성은 요청 순번, 토큰 갱신은 Axios 인터셉터의 대기 큐, 미디어 권한은 Promise 공유와 선제 재발급, 여행 진입은 1회 재조회 후 판단하는 방식으로 정리했습니다.

기술 TypeScript, Axios Interceptor, TanStack Query, Google Maps JavaScript API

비행기 모드에서도 열리는 오프라인 모드

조회 데이터와 바우처 원본 파일을 나눠 저장하고 연결 상태에 따라 동작을 바꿔, 한 번 연 여행은 네트워크 없이도 볼 수 있게 했습니다.

문제 여행 앱 특성상 기내나 해외 로밍 음영 지역처럼 네트워크가 없는 상황에서 일정과 바우처를 확인해야 하는 경우가 많았지만, 기존 구조는 네트워크가 끊기면 빈 화면이 나타났습니다.

고민 무엇을 어디에 저장할까

기각 목록 데이터와 원본 파일을 한 저장소에 수 MB PDF·이미지와 목록 데이터는 크기와 수명이 달라 함께 관리하기 어렵습니다.

채택 목록은 Query 캐시 영속화, 원본 파일은 기기 파일 + SQLite 인덱스 각 데이터의 특성에 맞게 저장하고 정리할 수 있습니다.

네트워크가 없을 때 어떻게 동작할까

기각 항상 서버 응답을 기다림 오프라인에서는 빈 화면만 보입니다.

채택 NetInfo로 연결 상태를 감지하고 offlineFirst로 캐시를 먼저 표시 바로 보여주고, 다시 연결되면 최신 내용으로 갱신합니다.

해결 React Query onlineManager에 NetInfo를 연결하고 쿼리·뮤테이션을 offlineFirst로 설정했습니다. 클라이언트 데이터는 Redux, 서버 데이터는 React Query에 AsyncStorage를 연동해 저장하고, PersistQueryClientProvider로 앱을 다시 켜도 즉시 렌더링했습니다. 바우처 파일 캐시는 0바이트 손상 파일을 다시 받고, 90일 동안 열지 않은 파일은 정리했습니다.

성과 한 번 연 여행은 비행기 모드에서도 일정과 바우처를 볼 수 있고, 다시 연결되면 동행자가 수정한 내용으로 갱신됩니다.

기술 React Native (Expo), TanStack Query Persist Client, Redux, AsyncStorage, NetInfo, expo-sqlite

라이더타운 V2

라이딩 기록 서비스 피벗 & 서버리스 전환

오토바이 라이딩 기록과 크루 실시간 위치 공유 서비스입니다. 유저 지표에서 정비 예약보다 라이딩 기록의 리텐션이 10배 이상 높아 라이딩 중심으로 피벗했습니다.

기간	2024.06 - 2024.12 (7개월)
팀	영업 담당 공동창업자와 2인 팀
역할	공동창업 / 풀스택 개발
담당	React Native 앱 개발, Firebase 서버리스 백엔드 구축·운영
기술	React Native, Node.js, GCP, Firebase, react-native-background-geolocation, Mapbox

[App Store ↗](#) [Google Play ↗](#)

이 프로젝트의 문제 해결 사례

- **CASE 06** 화면이 꺼져도 끊기지 않는 주행 기록

화면이 꺼져도 끊기지 않는 주행 기록

네이티브 백그라운드 위치 수집과 속도·정지 상태 기반 동적 전송 주기로, 휴대폰 화면을 끈 채 달라도 기록이 이어지게 했습니다.

70% ↑

실시간 경로 전송량 절감

거리 샘플링 + polyline 인코딩

2~30초

상황별 전송 주기

속도·정지·백그라운드 상태별

문제 라이더는 휴대폰을 거치대에 꽂고 화면을 끈 채 달리는데, 화면이 꺼지면 위치 수집과 기록 시간이 멈추거나 어긋났습니다. 1초마다 좌표를 통째로 보내는 방식은 배터리 소모와 발열이 컸습니다.

고민 백그라운드 위치 수집

기각 기본 Geolocation(watchPosition) 백그라운드로 가면 OS가 JS 실행을 멈춥니다.

채택 react-native-background-geolocation 네이티브 서비스가 수집하고 좌표를 기기에 저장합니다.

기록 시간 계산

기각 setInterval로 1초씩 더하기 백그라운드에서 타이머가 밀려 실제보다 짧게 기록됐습니다.

채택 시작 시각과 현재 시각의 차이로 매번 계산 타이머 지연과 무관하게 정확합니다.

위치 전송 주기

기각 1초 고정 전송 신호 대기나 휴식 중에도 계속 보냈습니다.

채택 속도·정지 여부·앱 상태에 따른 동적 주기 필요할 때만 자주 보냅니다.

해결 전송 주기를 30km/h 이상 2초, 10~30km/h 3초, 저속 4초, 백그라운드 5초, 15초 이상 정지 시 30초 하트비트로 나눴습니다. 경로는 속도별 거리 샘플링(20m/40m/60m)으로 점을 줄이고 polyline 문자열로 인코딩해 최소 4초 간격으로 보냈습니다.

성과 화면을 끈 상태에서도 기록이 이어졌고, 실시간 경로 전송량을 70% 이상 줄였습니다.

기술 React Native, react-native-background-geolocation, Firebase Realtime Database, Mapbox

라이더타운 V1

오토바이 정비 플랫폼 창업 · 라이딩 기록 및 그룹 음성 통화

오토바이 정비 견적·예약 중개 플랫폼입니다. 적은 인원으로 빠르게 출시하기 위해 앱 화면을 웹뷰로 제공하는 하이브리드 구조로 만들었습니다.

기간	2022.11 - 2023.12 (1년 2개월)
팀	영업 담당·백엔드 개발자와 3인 창업팀
역할	공동창업 / 풀스택 개발
담당	React-Next.js 웹뷰와 React Native 앱 개발, 백엔드·인프라 공동 구축
기술	React, Next.js, React Native, Redux, Node.js, Express, WebSocket, Mediasoup v3, WebRTC, Java, Spring Boot, Redis

[Web ↗](#) [App Store ↗](#) [Google Play ↗](#)

이 프로젝트의 문제 해결 사례

- **CASE 07** 웹뷰 하이브리드 앱의 느린 첫 화면과 네이티브 연동

웹뷰 하이브리드 앱의 느린 첫 화면과 네이티브 연동

React SPA를 Next.js SSR로 전환해 웹뷰 첫 화면을 3.5초에서 0.8초로 줄이고, 화면 이동과 테마를 네이티브와 맞췄습니다.

78%

초기 번들 감소

2.5MB → 550KB

77%

웹뷰 첫 화면 단축

3.5초 → 0.8초

문제 앱 화면을 웹뷰로 제공했는데, React SPA의 JS 번들이 2.5MB를 넘어 웹뷰 첫 진입 시 3.5초 이상 화이트스크린이 보였습니다. 웹 라우터로 이동하면 뒤로가기와 스와이프 제스처도 앱과 다르게 동작했습니다.

고민 첫 로딩을 어떻게 줄일까

기각 라이브러리 교체·dynamic import 분할 첫 로딩이 크게 줄지 않았습니다.

기각 이동할 페이지의 웹뷰를 미리 띄워두는 프리로딩 기기 메모리를 많이 차지했습니다.

채택 Next.js SSR 전면 전환 서버에서 첫 화면을 그려 초기 번들 의존을 줄였습니다.

페이지 이동을 어디서 처리할까

기각 웹뷰 안에서 웹 라우터로 이동 뒤로가기·스와이프 제스처가 네이티브와 달랐습니다.

채택 웹뷰 안에서는 postMessage로 네이티브 화면, 브라우저에서는 웹 라우터 환경마다 자연스러운 이동 방식을 씁니다.

해결 Next.js SSR로 마이그레이션하고, 브릿지 메시지를 Android는 document·iOS는 window에서 받도록 처리해 앱의 다크모드를 웹에 동기화했습니다.

성과 초기 번들을 2.5MB에서 550KB로, 웹뷰 첫 화면을 3.5초에서 0.8초로 줄였습니다.

기술 React, Next.js, React Native WebView