

# [이력서] 김재환

## 소개

- JS / TS / React / Vue 같은 프레임워크를 사용하는 **4년차 프론트엔드 개발자**로 스타트업에서 웹서비스를 기획 / 개발하여 **와디즈 펀딩 509%** 달성하고 **20억 투자를 유치**하며 **팀 리더** 역할을 겸하였습니다.
- **팀과 회사에 기여**하는 개발을 하는 것을 좋아하여 팀의 편의를 위한 테스트 툴 제작, 노션 커뮤니케이션 시스템 구축, 역량 강화 스터디 진행을 하며 혼자만의 역량 강화가 아닌 팀의 강화와 팀 간의 협업을 중심으로 개발해왔습니다.

✉ ahhancom@gmail.com

🔗 [GitHub](#)

🌐 [포트폴리오 웹사이트](#)

## 경력

### TeamGrit

2021 ~ 2023.05

(2년 1개월)

프론트엔드 웹 기획 / 개발 & 배포 / 관리  
실시간 미디어 & 로봇 컨트롤 기능 개발  
Typescript와 Vue 를 사용한 반응형 웹 개발

### 130T

2020 ~ 2021

(1년 3개월)

React Electron App 제작  
블록 코딩 교육용 앱 개발 & 교육  
기존 스크래치3 앱 개조

## 프로젝트 상세

### 멀티 유저 로봇 운영 플랫폼 개발

2022.11.10 ~ 2023.05.01 (주)팀그릿

#### 주요 업무

- 하드웨어, 미디어 등록 / 관리 시스템 개발
- 플레이어 관리 시스템 개발
- 프론트엔드 환경 구성 / 웹 배포 및 관리

## 해결해야했던 문제

### 문제1) 미디어의 스레드 침해

#### 배경

- 기존의 프로젝트에서 더욱 강력한 아키텍처를 채택하여 N개의 미디어 스트림을 관리하다 보니 미디어 스트림이 메인 스레드에 영향을 끼쳐 UI가 버벅대는 등 일관된 사용자 경험을 보장할 수 없어졌습니다.

#### 해결과정

- `webWorker` 를 사용하면 스레드를 효과적으로 분리하여 메인 스레드에 영향을 미치거나 전체 사용자 경험을 방해하지 않는다는 것을 알게 되었습니다.
- `encoder` , `decoder` 의 기능이 미디어 기능 중 가장 많은 메모리를 사용하여 해당 기능을 worker 스레드로 분리하였습니다.

#### 성과

- `webWorker` 를 도입하여 미디어와 메인 스레드를 완전히 분리하고 이전에 미디어 관련 문제로 인해 발생했던 UI 끊김 오류를 완전히 제거했습니다.
- 메인 스레드의 메모리 16MB를 worker 스레드로 분리하였습니다.

#### 영향 및 학습 경험

- 복잡한 시스템에서 스레드 간의 상호 작용을 이해하고 관리하는 것의 중요성을 배웠습니다.

### 문제2) 비효율적인 이벤트 관리

#### 배경

- `webWorker` , `webCodec` 및 `webBluetooth` 와 같은 여러 모듈이 포함된 웹 애플리케이션을 개발하는 동안 효율적이고 중앙 집중화된 이벤트 관리 방법이 필요하다는 생각이 들었습니다.

#### 해결과정

- 디자인 패턴에 대한 깊은 관심으로 저는 옵저버 패턴이 전역 이벤트 관리 요구 사항에 가장 적합하다는 것을 확인했고 옵저버 패턴을 시스템에 통합하여 조각난 이벤트 관리 시스템에 일관성과 구조를 가져왔습니다.
- 옵저버 패턴의 남용 가능성에 대한 초기 우려에도 불구하고 이벤트 관리 시스템을 개선하는 데 있어 그 가치를 인식하고 이를 구현하기 위한 결정적인 조치를 취했습니다.

#### 성과

- 옵저버 패턴을 사용하여 유효하지 않거나 잘못 관리되는 이벤트를 크게 줄였습니다.
- 중복 인스턴스 생성을 방지하기 위해 싱글톤 패턴을 도입하여 자원을 절약하고 시스템의 효율성을 높였습니다.

Vue3 + pinia / TypeScript / playwright / vitest / msw / tailwindcss / webBluetooth / S3 / Route53 / CloudFront

## 한국-일본 원격 로봇 대회용 웹 구축

2022.01.01 ~ 2022.10.07 (주)팀그릿

### 주요 업무

- 하드웨어, 미디어 컨트롤 / 관리 시스템 개발
- 프론트엔드 환경 구성 / 반응형 웹 배포 및 관리
- 로그인 / 회원가입 / 게시판 구현, PG사 결제 시스템 연동

### 해결해야 했던 문제

#### 문제1) 네트워크 부하

##### 배경

- 초기 솔루션은 미디어 통신을 위해 'webRTC'에 의존하였지만 새로운 서버 솔루션을 사용하기 위해 'WebSocket'을 통해 'JPEG' 미디어를 전송했습니다. 그러나 이 접근 방식은 프레임당 크기를 크게 증가시켜 서버 비용이 높아지고, 사용자의 네트워크 부하가 증가했습니다.

##### 해결과정

###### 1. 이미지 압축

이를 해결하기 위해 미디어 인코딩 및 디코딩을 위한 브라우저 API인 `webCodec` 을 사용하여 이미지 압축을 구현했습니다.

*Side effect*) `webCodec` 이 영상을 효과적으로 압축했지만, 실시간 전송으로 인해 프레임 밀림과 간헐적인 끊김 현상이 발생했습니다.

*Trouble Shooting*) 프레임이 3개 이상 쌓이는 것을 방지하는 `queue` 알고리즘을 도입하여 원활한 실시간 전송을 보장함으로써 이를 해결했습니다.

###### 2. 적응형 해상도 구현

일본과 같이 네트워크 상황이 불확실한 지역의 다양한 네트워크 상황을 고려하여 네트워크 대역폭을 기반으로 해상도를 동적으로 조정하는 모듈을 개발했습니다.

##### 성과

- `encoder` 및 `decoder` 모듈을 성공적으로 개발하여 **프레임당 크기를 10배 줄였습니다.** 이로 인해 서버 및 사용자 네트워크 비용이 크게 절감되었습니다.
- 네트워크 대역폭에 따라 적응형 해상도를 활성화하여 사용자 경험을 개선하며 느린 네트워크 조건에서도 실시간 보기를 보장합니다.

#### 문제2) 비효율적인 QA 시스템으로 인한 생산성 감소 현상

##### 배경

- 기존의 QA 프로세스에는 수동으로 웹 사이트를 탐색하고 직접 하드웨어 제어 테스트를 수행하고 `console.log` 출력을 검사하여 프로토콜을 확인하는 작업이 포함되었습니다.  
이러한 반복 프로세스는 시간이 많이 걸렸고 생산성에도 큰 영향을 미쳤습니다. 따라서 중복을 피하고 생산성을 향상시키기 위한 자동화 솔루션을 탐색하게 되었습니다.

## 해결과정

### 1. E2E 테스트를 통한 QA 자동화

왜 **Playwright**를 사용했는가

QA 프로세스를 자동화하기 위해 **Playwright**를 사용하여 로그인 및 회원 등록과 같은 일상적인 웹 작업에 대한 E2E 테스트를 작성했습니다.

### 2. 하드웨어 검증 테스트 페이지 제작

하드웨어 개발자가 하드웨어를 직접 테스트하지 않아도 되도록 테스트 페이지를 만들었습니다. 버튼을 한 번만 누르면 이 페이지는 프로토콜을 확인하기 위한 하드웨어 요청을 시작하고 로봇의 모든 기능이 제대로 작동하는지 확인합니다.

## 성과

- QA 자동화를 성공적으로 구현하여 시간을 크게 절약하고 생산성을 향상시켰습니다.
- 하드웨어 검증을 위한 전용 테스트 페이지를 개발하여 개발자가 하드웨어를 직접 테스트할 필요성을 줄이고 효율성을 더욱 높였습니다.

## 영향 및 학습 경험

- QA 프로세스 자동화의 영향은 현저하게 향상된 생산성입니다. 수동 단계를 제거함으로써 테스트에 소요되는 시간을 줄이고 더 많은 작업을 위한 리소스를 확보했습니다. QA 프로세스에서 자동화의 가치와 이를 통해 개발자가 보다 중요한 문제에 집중할 수 있는 시간을 확보할 수 있는 방법을 이해했습니다.

## 문제3) 기존 개발 환경의 5가지 문제점

### 배경

기존 개발 환경에서 코드의 가독성, 확장성, 안정성 및 개발 속도에 영향을 미치는 여러 문제에 직면했고, 가장 우선순위가 높은 5가지를 선정했습니다.

- **JavaScript**의 한계
- 비효율적인 **CSS** 개발 방법
- 웹의 크기가 커짐에 따라 **Webpack** 과의 번들링 속도가 느려짐
- **RestAPI**로 인한 과도하고 불필요한 데이터 로드
- 빈번하고 정리되지 않은 API 호출

### 해결 과정

#### • 코드 퀄리티 향상

**JavaScript**에서 **TypeScript**로 전환하여 코드 가독성과 확장성을 개선하고 정적 타이핑을 통해 안정성을 확보했습니다.

#### • CSS 프레임워크 도입

**SCSS**에서 **TailwindCSS**로 전환하며 반응형 웹 디자인 제작이 간소화되어 개발 속도가 약 30% 향상되었습니다. 그러나 이로 인해 코드 가독성이 떨어지고 Scope 단위에서 더 많은 **CSS**를 사용하게 되었습니다. 이를 해결하기 위해 전역으로 사용하는 데 중점을 두었습니다.

#### • Vite 도입

번들링의 속도 향상을 위해 **Webpack**을 Rollup과 Esbuild를 사용하는 **Vite**로 교체하였습니다.

#### • GraphQL 구현

백엔드 개발자와 협의 후 **RestAPI** 대신 **GraphQL** 을 사용하기로 결정했습니다. 이를 통해 필요한 데이터만 추출하고 수신할 수 있었습니다.

- **MVVM 파이프라인 구축**

API가 호출되는 위치를 더 잘 추적하기 위해 MVVM 파이프라인을 구축했습니다. API 호출은 **vuex** 스토어에서 이루어졌고 데이터는 스토어에서 관리되었습니다. 이 접근 방식은 view에서 API 호출을 금지하고 대신 API 남용을 방지하기 위해 스토어 데이터를 호출합니다.

## 성과

- 프로그래밍 및 스크립팅 언어, 번들링 도구 및 API를 전략적으로 변경하여 코드 가독성, 확장성, 안정성 및 개발 속도를 크게 개선했습니다.
- **GraphQL** 및 MVVM 파이프라인의 구현으로 데이터 추출 및 관리가 간소화되어 필요한 데이터만 사용되도록 하고 API 남용을 방지했습니다.

## 영향 및 학습 경험

- 프로그래밍 언어, 번들링 도구 및 API의 변화로 인해 개발 환경이 크게 개선되었으며 궁극적으로 생산성에 반영되었습니다. 프로젝트에 적합한 도구와 언어를 선택하는 것의 가치, 구조화된 데이터 관리의 이점, API 호출의 더 나은 제어 및 구성을 위해 MVVM 아키텍처로 개발할 때의 이점을 배웠습니다.

webBluetooth / webCodec / Vue3 + vuex / TS / Cypress / Tailwind css / Amplify / graphql / i18n / lamport API / vite / S3 / Route53 / CloudFront

## WebRTC 레이싱 대회 웹 구축

2021.06.01 ~ 2021.11.10 (주)팀그릿

### 역할

- webRTC 하드웨어, 미디어 컨트롤 시스템 개발
- 하드웨어 데이터 기반 레이싱 UI 개발
- 프론트엔드 환경 구성 / 반응형 웹 배포 및 관리
- 로그인 / 회원가입 / 게시판 / 예약 시스템 개발

### 해결해야했던 문제

#### 문제1) 멀티 스크린 Canvas

##### 배경

- 실시간 대화형 인터페이스를 개발하는 작업에 착수했습니다. 목표는 스티어링 휠, 계기판, 지도 카메라와 같은 제어 UI 요소를 매끄럽게 통합하여 **Canvas** 를 사용한 경주용 자동차 제어 디스플레이를 표현하는 것이었습니다.

## 해결과정

- 제어 디스플레이를 캔버스 화면에 렌더링 하기 위해 `captureStream` 메서드와 함께 `webRTC` 의 기능을 활용했습니다.
- 사용자 참여는 우리 디자인의 주요 고려 사항이었습니다. 이와 같이 휠은 왼쪽 및 오른쪽 화살표 키에 반응하고 게이지 바와 대시보드는 위쪽 및 아래쪽 화살표 입력으로 조정되는 시스템을 설계했습니다.

*Side effect*) 카메라 요소와 UI 요소를 하나의 캔버스에 출력하며 복잡해졌습니다. 특히 화면이 깜박이고 전반적인 성능이 저하되었습니다.

*Trouble Shooting*) 캔버스 레이어링(Canvas Layering)이라는 개념을 도입했습니다. 카메라 요소와 UI 요소를 자체 캔버스로 분리하고 이를 통해 하나의 `requestAnimationFrame` 메서드 내에서 초당 60프레임을 부드럽게 유지하면서 개별적으로 관리할 수 있었습니다.

*more*) 여러 `Canvas` 를 동시에 운영할 때 발생할 수 있는 성능 저하를 상쇄하기 위해 `OffScreenCanvas` 를 구현했습니다.

## 성과

- 캔버스를 분리하여 사용자 경험이나 성능을 희생하지 않고 다양한 UI 요소를 결합한 고성능 실시간 경주용 자동차 제어 디스플레이를 구현했습니다.
- `OffScreenCanvas` 로 인해 `Canvas` 가 worker 스레드로 전환되었으며 화면을 사전 렌더링하여 보다 부드러운 디스플레이로 사용자 경험을 크게 개선했습니다.

## 영향 및 학습 경험

- 이러한 경험을 통해 실시간 인터페이스 디자인에 대한 지식이 크게 발전시킬 수 있었고 여러 요소를 단일 캔버스에 통합하는 미묘한 문제를 이해할 수 있었습니다.

## 문제2) 배포환경의 부재

### 배경

웹 프로젝트를 처음 맡아 개발 환경부터 배포 환경까지 혼자 구축해야 하는 상황이었습니다. 목표로 하는 웹의 특징을 근거로 `SEO` 를 크게 신경 쓰지 않아도 되며 가볍고 서버 비용이 적은 환경을 구축해야 한다는 결과를 얻었습니다.

## 해결과정

- 상시 오픈 웹이 아닌 이벤트성 웹이기 때문에 `SSR` 이 아닌 `CSR` 을 선택하고 그에 맞게 프레임워크는 `nuxt` 가 아닌 `vue` 를 사용하게 되었습니다.
- `AWS` 환경을 이미 사용하고 있어서 환경의 통일성을 위해 `AWS` 를 선택했습니다. `S3` , `Route53` 을 사용하여 `CSR` 렌더링 웹 호스팅을 구성했습니다.

*Side effect*) `S3` 는 `HTTPS` 를 직접 지원하지 않습니다.

*Trouble Shooting*) `S3` 의 한계를 극복하고 정적 웹 사이트에 `HTTPS` 를 적용하기 위해 `CloudFront` 를 사용했습니다.

## 성과

- 가볍고 서버 비용이 낮은 환경으로 목적에 부합하는 웹을 구현하였습니다.
- `CloudFront` 의 구현은 `HTTPS` 제한을 해결했을 뿐만 아니라 `CDN` 를 사용하여 대기 시간 단축 및 높은 트래픽 로드를 효율적으로 처리하는 능력 향상과 같은 추가 이점을 제공했습니다.

## 영향 및 학습 경험

- 두 가지 렌더링 방법의 강점에 대한 균형 잡힌 견해와 프로젝트 요구 사항에 따라 현명하게 선택할 수 있는 능력을 갖추게 되었습니다.

### 문제3) 잠재적인 문제

#### 배경

- 해당 웹의 컨셉과 통일된 게시판을 제작해야 했습니다. 해당 게시판의 요구사항이 충족되는 게시판 라이브러리를 발견되지 않았고 이로 인해 기존 라이브러리를 수정할 것인지 아니면 처음부터 맞춤형 라이브러리를 개발할 것인지 문제가 생겼습니다.

#### 해결과정

- 라이브러리가 요구사항에 맞지 않아 직접 개발하는 것을 채택하였습니다.

*Side effect*) 처음부터 솔루션을 만드는 것은 복잡했습니다. 초기에 고려하지 않은 여러 변수가 있어 예상치 못한 부작용이 발생했습니다. 그 결과 개발 일정이 초기 예상보다 며칠 더 연장되었습니다.

#### 성과

- 웹의 컨셉과 완벽하게 일치하는 디자인 게시판을 성공적으로 구현하였습니다.

#### 영향 및 학습 경험

- 이 프로젝트를 통해 처음부터 솔루션을 개발할 때 수반되는 복잡성과 예상치 못한 변수에 대해 심도 있게 이해했습니다. 비록 도전적 이기는 했지만, 잠재적인 문제에 대한 상세한 계획과 예측에 대한 통찰력과 자만하지 않는 마음가짐을 가지게 되었습니다.

Vue3 / JS / SCSS / WebRTC / canvas / webpack / S3 / Route53 / CloudFront

## 코딩교육용 스크래치3기반 앱 개발

2020.02.03 ~ 2021.04.13 (주)130T

#### 역할

- Electron & React 데스크톱 앱 개발
- Blockly 기반 전용 블록 개발
- 코딩 블록과 BLE 사용 하드웨어 컨트롤 기능 개발

#### 해결해야했던 문제

##### 문제 1) 첫 업무, 첫 개조 : 스크래치3 뜯어고치기

#### 배경

- 팀에 합류한 후 첫 번째 작업은 Scratch 3를 기반의 소프트웨어 개발이었습니다. 이미 완성된 프로그램을 개조해야 했기 때문에 오히려 어려운 작업이었습니다. 프로젝트가 일반적이지 않아서 최소한의 참고 자료로 어려움을 겪으면서 어려움이 가중되었습니다.

## 해결과정

- 초기 장애물을 극복하기 위해 먼저 순서도를 만들어 스크래치 3의 기본 구조를 이해할 수 있었습니다. 이 분석을 통해 전체 레이아웃과 작동 방식을 파악할 수 있었고 시작하는 데 필요한 통찰력을 얻을 수 있었습니다.

## 성과

- 개조를 성공적으로 하였습니다.

## 영향 및 학습 경험

- 이 경험은 이전에 익숙하지 않은 코드 구조를 이해하고 분해하는 능력을 크게 향상시켰습니다. 이미 개발된 소프트웨어를 분석하고 이해하는 능력을 통해 다양한 코딩 구조가 작동하는 방식에 대한 시야를 넓힐 수 있었습니다.

React & Redux / Electron / Blockly / JavaScript

---

## 토이 프로젝트

---

### Developic

개발 중 프로젝트

- 프로젝트 디자이너 1, 백엔드 개발자 2와 프론트엔드 개발자인 저를 포함한 총 4명의 인원으로 시작된 프로젝트입니다.
- 해당 서비스 GPT API를 사용하여 개발자들에게 언어와 난이도에 맞는 리팩토링 과제를 던져주어 리팩토링 공부를 할 수 있도록 도와주는 서비스입니다.

### 이슈

- **Preact → React로 변경** : yarn berry를 preact에서 사용하면 preact-vite 모듈과 babel 모듈의 호환 문제 에러가 발생.
- **SWC** : **babel** 은 **node** 로 개발하였고 **SWC** 는 **Rust** 로 개발하여 빌드 시간이 더 빠르고 **SWC** 를 사용해 본 경험을 쌓기 위해 사용.
- **yarn berry pnp mode** : zero-install로 인한 CI 속도 향상
- **Cusmtom hook 패턴 사용** : Container의 로직만 hooks로 관리하는 방법입니다

### CODE : GitHub

TS / React + recoil / StyledComponent + Tailwindcss / E2E : playwright / yarn berry(pnp) / msw / swc

---



## Skill Set

구분	skill
Framwork	Vue, React, Electron, PhaserJS
Language	HTML/JS, TypeScript
styles	Tailwindcss, Scss
CI	GithubAction
web Media	WebRTC, WebCodec
TEST	playwright, vitest, jest ,msw
AWS	Route53, CloudFront, S3, EC2
Package Manager	yarn berry(pnp)
API	REST, GraphQL (Apollo)

## 기타 경력 및 경험


- 웹 디자인 기능사 자격 보유
- 2012 부산 기능 경기 대회 동상 수상

## 학력

- 2015 ~ 경남정보대학교 - 컴퓨터 정보 계열
- 2020.03.02 ~ 2023.08 방송통신대학교 - 컴퓨터 과학과 8월 졸업 예정

 ahhancom@gmail.com

 [GitHub](#)

 [포트폴리오 웹사이트](#)