

Front-end Engineer  
정태욱  
PORTFOLIO

---



# 소개

반갑습니다. 프론트엔드 엔지니어 정태욱입니다.  
주로 React.js, Typescript를 사용하여 웹 애플리케이션을 개발합니다,  
Webpack을 사용한 프론트 환경 설정과 Next.js에 관심이 많습니다.  
On-premise 환경에서 웹 애플리케이션을 개발한 경험이 있습니다.

Phone.  
010-6661-8337

Email.  
scv7282@gmail.com

Address.  
경기도 성남시 분당구 구미동 성우스타우스

Github.  
<https://github.com/peacepiece7>

Blog.  
<https://web-log-wheat.vercel.app>

# 보유 기술



사용 경험이 있는 기술 및 기타 도구



# 목차

## Projects

### 1. RFP 기반 웹사이트 공유 페이지 리뉴얼

전자 문서 서비스를 제공하는 B2B SaaS (주)Jober의 RPF를 기반으로 공유 페이지를 리뉴얼하는 기업 연계 프로젝트입니다. PM, UX/UI, BE, FE 팀이 참가했으며 프론트엔드 리더를 맡았습니다.

### 2. 개인 기술 블로그

Next.js 13.4의 서버 컴포넌트와 ISR로 동작함으로  
SSG의 단점인 리빌드와, SSR의 단점인 느린 FMP와 prefetch 시 빈번한 API 호출을 보완했습니다.

### 3. 전자 제품 데이터 B2C 서비스

반도체 데이터 시트를 제공하는 B2C (주)Interbird의 데이터와 Puppeteer를 사용한 크롤러를 기반으로  
전자 부품 정보를 제공하는 웹 사이트를 제작했습니다.

On-premise 환경에서 동작하며, Next.js 12, Express.js, MySQL을 사용했습니다.

# RFP 기반 웹사이트 공유 페이지 리뉴얼

전자 문서 서비스를 제공하는 B2B SaaS (주)Jober의 RPF를 기반으로 공유  
페이지를 리뉴얼하는 기업 연계 프로젝트입니다.



프로젝트명 : Jober chip

기간 : 2022.08.15 ~ 2023.10.06

인원 : FE 5명, BE 5명, UX/UI 2명, PM 2명

Glithub : 깃허브 저장소로 이동

배포 :

랜딩 페이지로 이동

개인 공유 페이지 비회원으로 방문

시연 영상으로 이동

역할

- 프론트엔드 리드
- 프론트 전체 개발 과정 참여

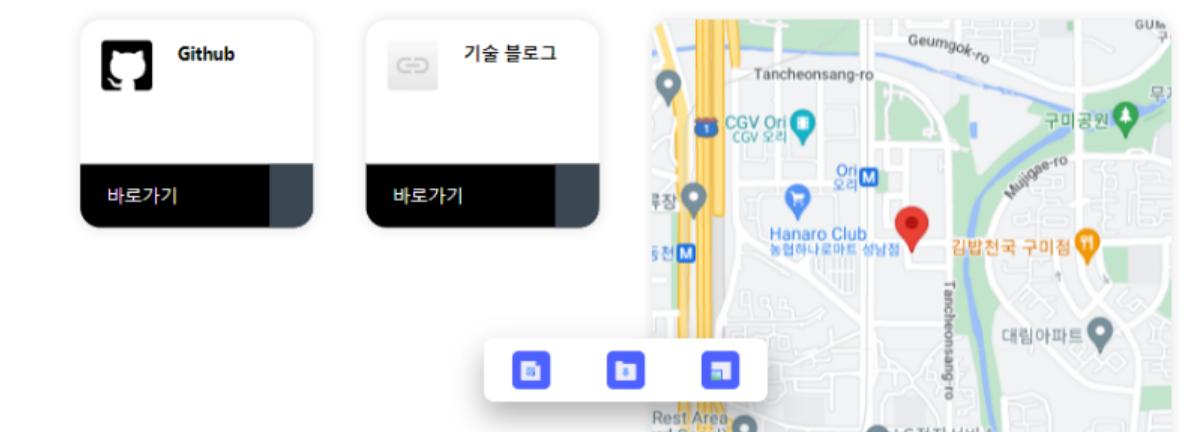
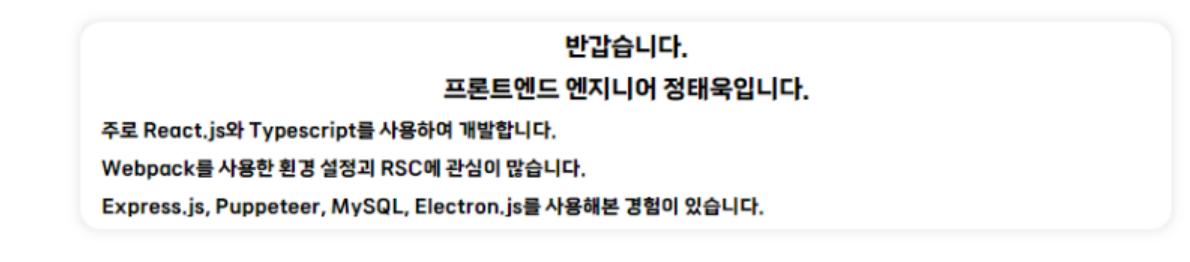
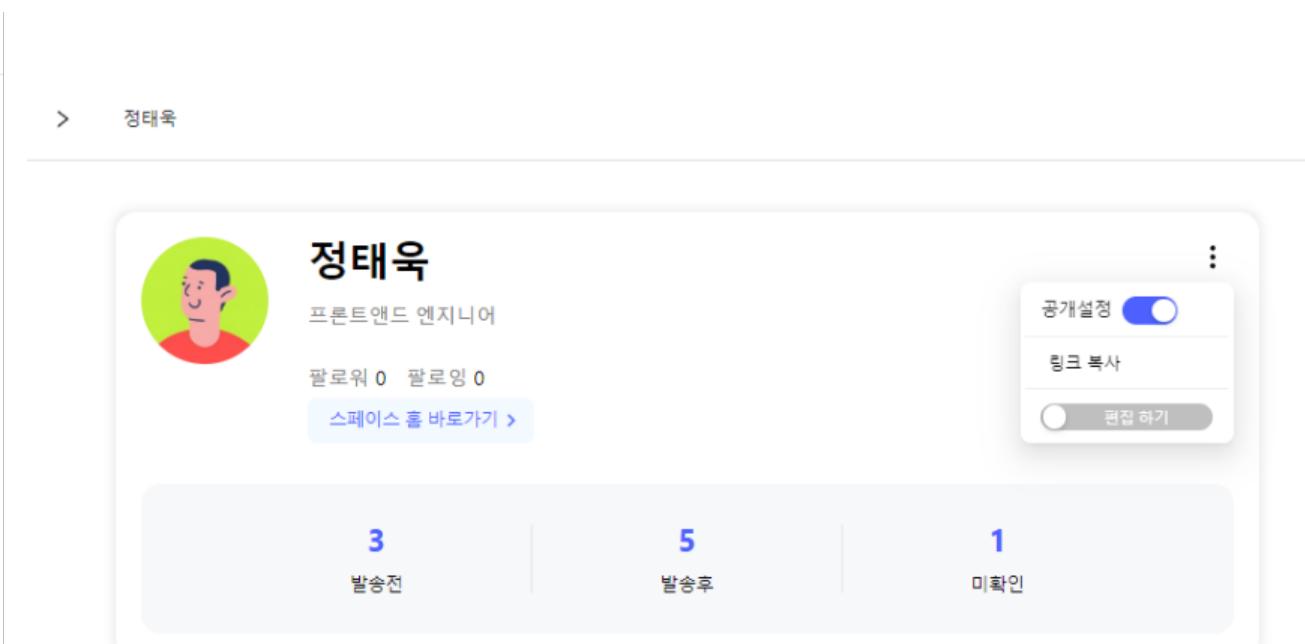
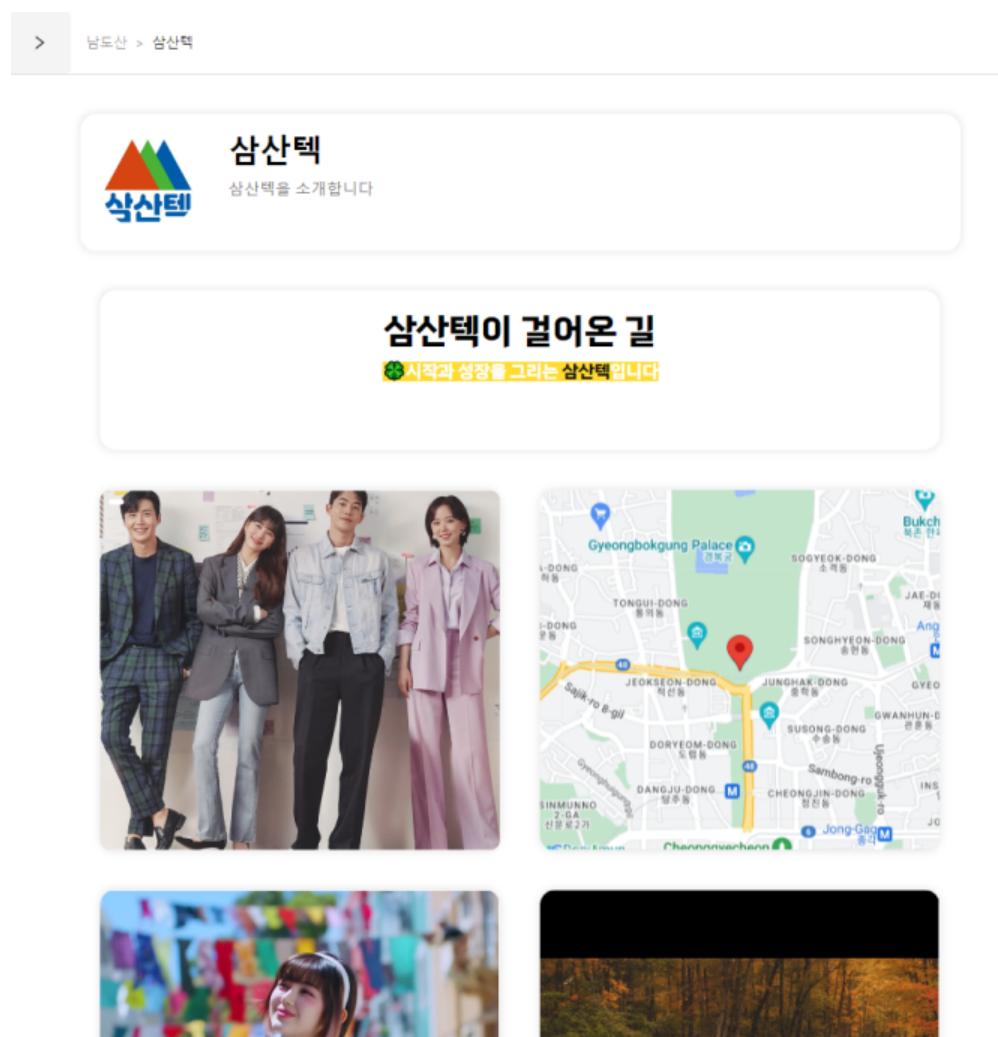
사용기술

Front-end : React.js, Typescript, `@tanstack/react-query(v4)`, zustand, SCSS, webpack, Express.js, `@loadable/components`, `@loadable/server`

Back-end : Java17, Springboot, Spring Security, JPA, gradle

Database : MySQL

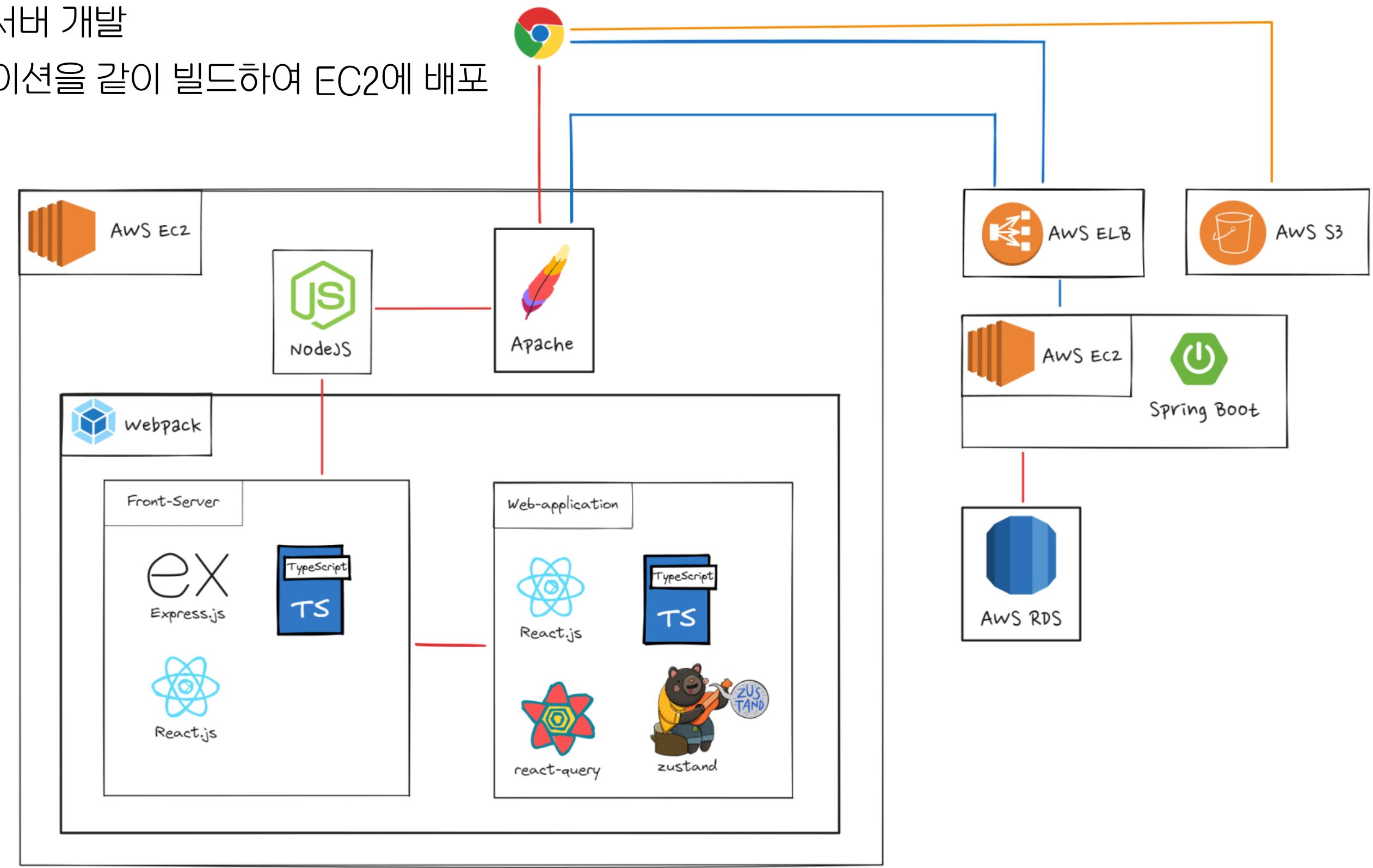
Deploy : AWS EC2, AWS S3



# 아키텍처

Express.js를 사용하여 프론트엔드 서버 개발

프론트 서버와 React.js 웹 애플리케이션을 같이 빌드하여 EC2에 배포



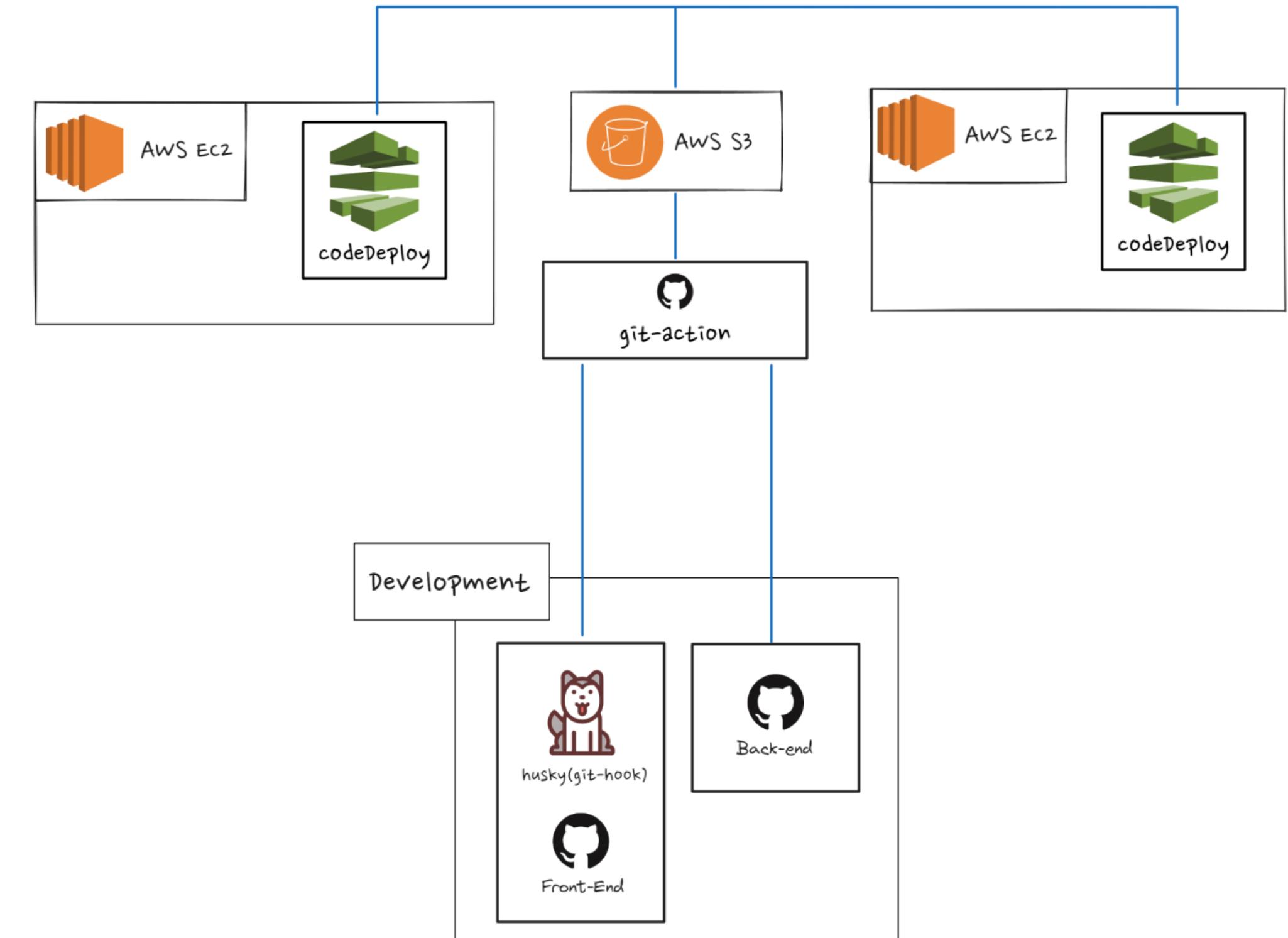
# 담당 가능

## CI/CD 파이프라인

husky를 사용하여 pre-commit/push git-hook으로 검사

git-actions를 사용하여 빌드 후

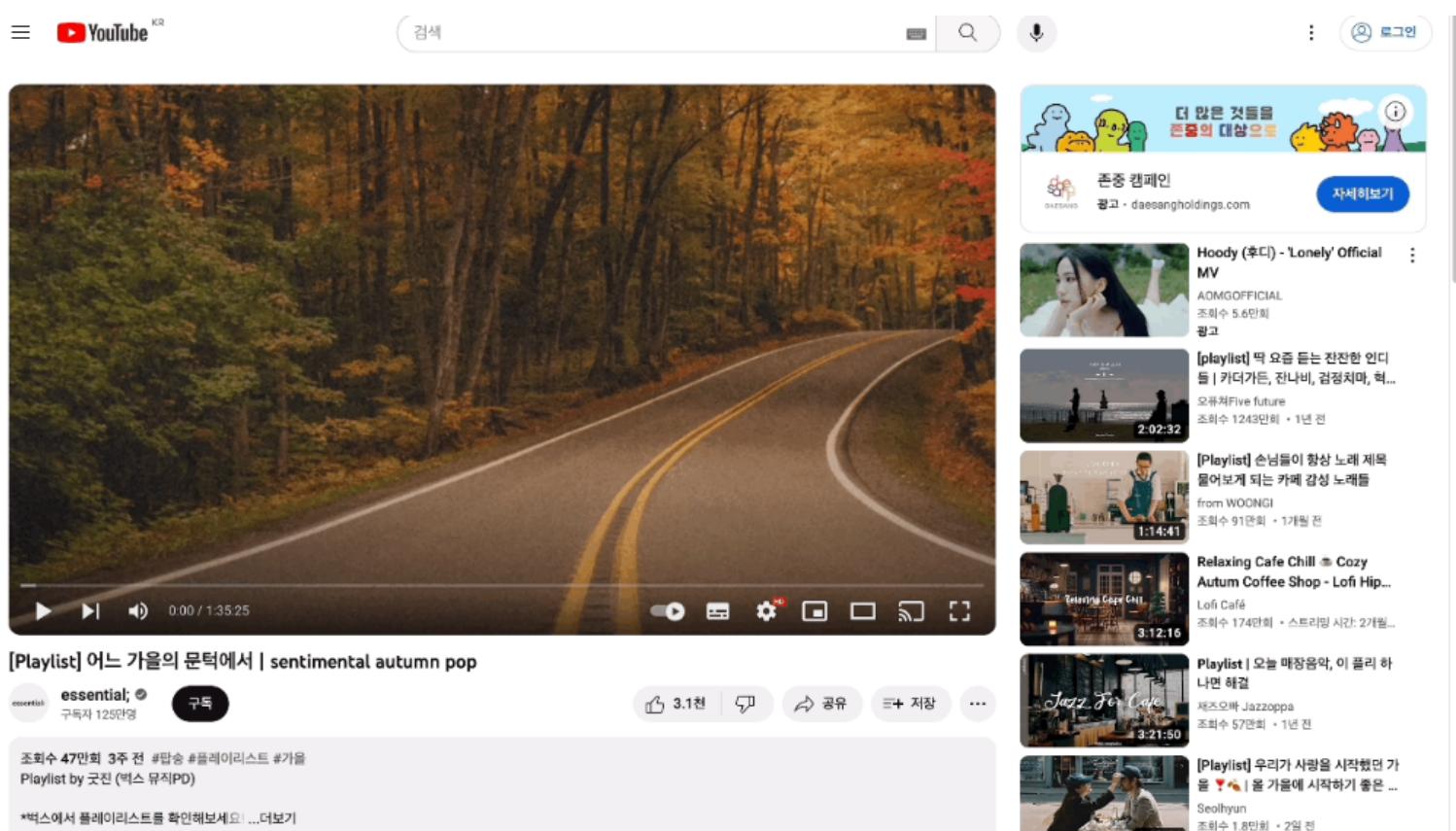
S3, codeDeploy로 EC2에 지속적 배포



# 담당 가능

## react-grid-layout를 사용하여 DnD 구현

Zustand를 사용해 layout 상태를 저장하고 layout이 변경이 감지될 경우 debounce를 걸어 일정 시간 후 변경 사항이 자동으로 저장되도록 구현  
[react-grid-layout을 사용한 DnD 코드 링크](#)



## 코드 충돌을 줄이기 위한 공통 로직 처리

팀원들간 코드 conflict를 최소화 하기위해 프로젝트 내에서 block이라 정의된 요소와 관련된 각종 공통 로직을 따로 처리하는 여러 SwitchCase를 구현  
[SwitchCase 풀더 링크](#)  
[block 탑 설정 코드 링크](#)

```

12 export interface BlockBaseWithBlockFormProps<T extends BlockType> extends Partial<BlockBaseWithBlockProps<T>> {}

13
14 function getEditFormComponent<T extends BlockType>({ block }: BlockBaseWithBlockFormProps<T>) {
15   switch (block?.type) {
16     case TEXT:
17       return <TextBlockForm block={block} />
18     case IMAGE:
19       return <ImageBlockForm block={block} />
20     case LINK:
21       return <LinkBlockForm block={block} />
22     case VIDEO:
23       return <VideoBlockForm block={block} />
24     case MAP:
25       return <GoogleMapBlockForm block={block} />
26     case PAGE:
27       return <PageBlockForm block={block} />
28     case TEMPLATE:
29       return <TemplateBlockEditForm block={block} />
30     default: {
31       if (process.env.NODE_ENV === 'development') {
32         throw new Error('Please add a component')
33       }
34       return null
35     }
36   }
37 }
```

# 담당 가능

## Mock API 개발

API 개발이 완료되기 전까지 프론트엔드에서 대기해야하는 상황을 해결하기 위해 Mock API 개발  
Express.js를 사용하여 MOCK API 생성하고  
Mock data는 json 포맷의 파일로 관리



### 관련 코드

[mock data 파일 링크](#)

[프론트 서버 index.ts 코드 링크](#)

```

dist
└── mocks
    ├── development
    │   ├── logo.json
    │   ├── space.json
    │   ├── templates.json
    │   ├── tree.json
    │   └── user.json
    └── production
        ├── logo.json
        ├── space.json
        ├── templates.json
        ├── tree.json
        └── user.json
node_modules
public
scripts
└── server
    ├── api
    ├── render
    ├── routes
    └── utils
index.ts

```

```

13 config.output.path = config.output.path.replace(`dist/dist/`, `dist`)
14
15     return config
16 }
17
18 const webpackDevMiddleware = require('webpack-dev-middleware')
19 const webpackHotMiddleware = require('webpack-hot-middleware')
20 const compiler = webpack(webpackConfig)
21 app.use(
22     webpackDevMiddleware(compiler, {
23         writeToDisk: true,
24         publicPath: webpackConfig[0].output.publicPath
25     })
26 )
27 app.use(webpackHotMiddleware(compiler))
28
29 app.use(cors())
30 app.use(express.json())
31 app.use(express.urlencoded({ extended: true }))
32 app.use(compress())
33 app.use(express.static('public'))
34 app.use(express.static('dist'))
35
36 app.use('/api/auth', authRouter)
37 app.use('/api/space', spaceRouter)
38 app.use('/api/template', templateRouter)
39
40 app.use('/', (req, res) => {
41     res.send('Hello, World!')
42 })
43
44 module.exports = app

```

```

.github
├── .husky
└── .vscode
dist
└── mocks
    ├── development
    │   ├── logo.json
    │   ├── space.json
    │   ├── templates.json
    │   ├── tree.json
    │   └── user.json
    └── production
node_modules
public
scripts
server
src
└── .babelrc.js
├── .env
└── .eslintrc.json
.gitignore
└── .gitignore
└── .prettierrc
└── .prettierrc
└── ecosystem.config.js

```

```

1 {
2     "space1": {
3         "pageId": "space1",
4         "pageProfileImage": "http://localhost:7282/profile_1.png",
5         "title": "user1의 스페이스",
6         "description": "안녕하세요",
7         "children": [
8             {
9                 "objectId": "block_1_1",
10                "y": 0,
11                "x": 0,
12                "h": 1,
13                "w": 2,
14                "type": "LINK",
15                "src": "http://www.naver.com",
16                "title": "네이버",
17                "description": "네이버로 이동해봅시다.",
18                "visible": true
19             },
20             {
21                 "objectId": "block_1_2",
22                 "y": 1,
23                 "x": 0,
24                 "h": 1,
25                 "w": 2,
26                 "type": "LINK",
27                 "src": "http://www.youtube.com",
28                 "title": "내가만든 유튜브 영상",
29             }
30     }
31 }

```

# 담당 가능

## SSRSuspense 컴포넌트

react-dom/server가 Suspense를 일부 지원하지 않는 문제가 발생하여 server side에서 Suspense가 fallback을 반환하도록 Suspense를 래핑하는 컴포넌트 개발

```
export const SSRSuspense = ({ children, fallback }: ComponentProps<typeof Suspense>) => {
  const [isMounted, setIsMounted] = useState(false)
  const [isBrowser, setIsBrowser] = useState(typeof window !== 'undefined')

  useEffect(() => {
    setIsMounted(true)
    setIsBrowser(typeof window !== 'undefined')
  }, [])

  if (isMounted && isBrowser) {
    return <Suspense fallback={fallback}>{children}</Suspense>
  }

  return fallback
}

export function withSSRSuspense<T>(
  WrappedComponent: ComponentType<T>,
  options: { fallback: NonNullable<React.ReactNode> | null }
) {
  return (props: T) => {
    return (
      <SSRSuspense fallback={options.fallback}>
        <WrappedComponent {...(props as any)} />
      </SSRSuspense>
    )
  }
}
```

You, 4 days ago • feat

# 성능 개선

## SSR/CSR 병행 지원

공유 페이지 접속시 특정 CASE에 따라 SSR과 CSR를 병행해서 사용함으로

사용자 고유 정보와 일부 링크가 페이지 소스에 포함되도록 개발하고

react-router-dom의 `useNavigate`이나 `Link`로 이동할 경우

CSR로 동작하여 최소한의 API만 호출

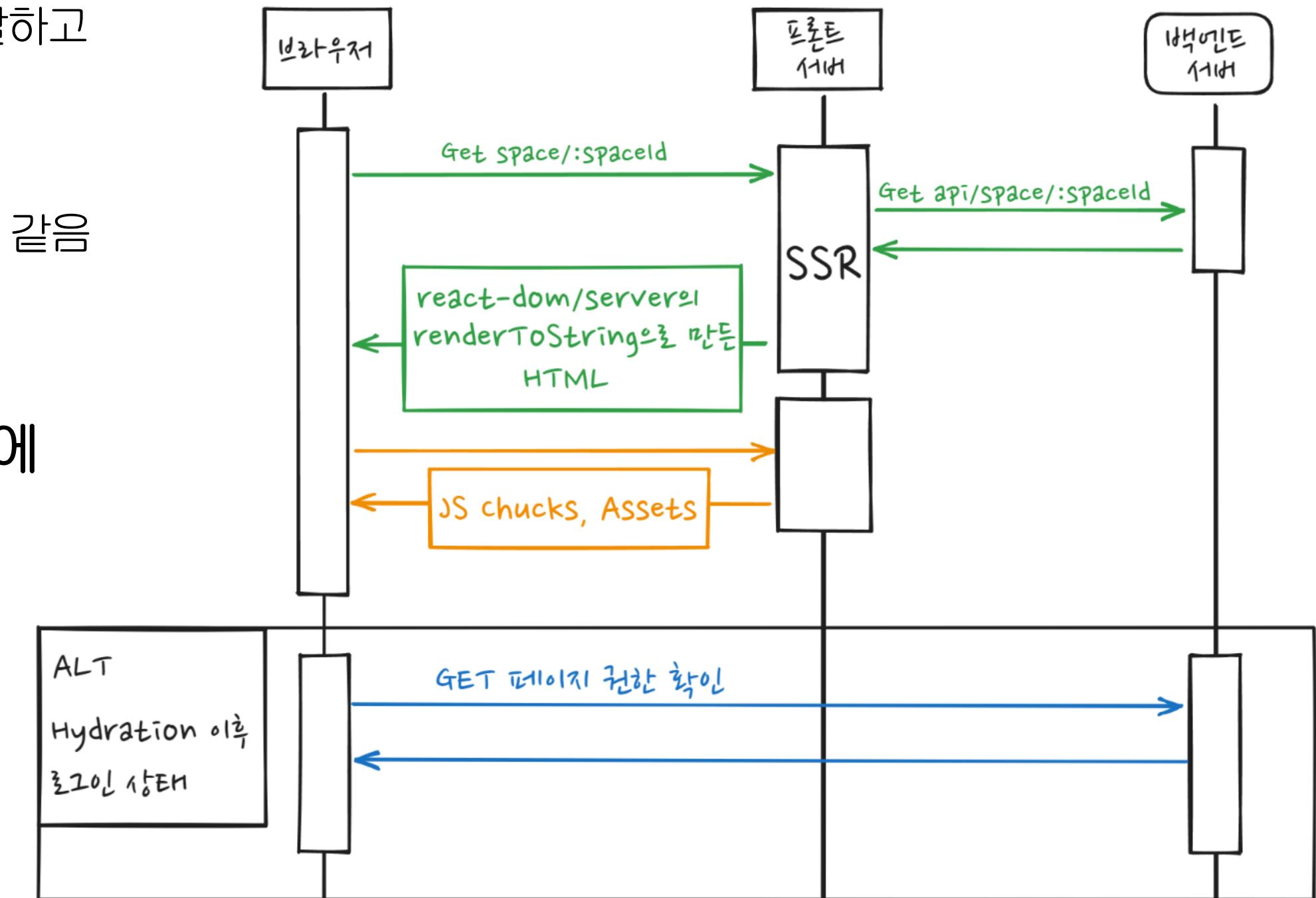
접속 방식에 따라 SSR/CSR이 달라지는 특정 케이스는 다음과 같음

(우측 그림 참고)

Case 1. 브라우저에 URL을 입력하여 공유 페이지에 접근할 경우 (Server Side Rendering)

오른쪽 Sequence diagram과 같이 react-dom/server의

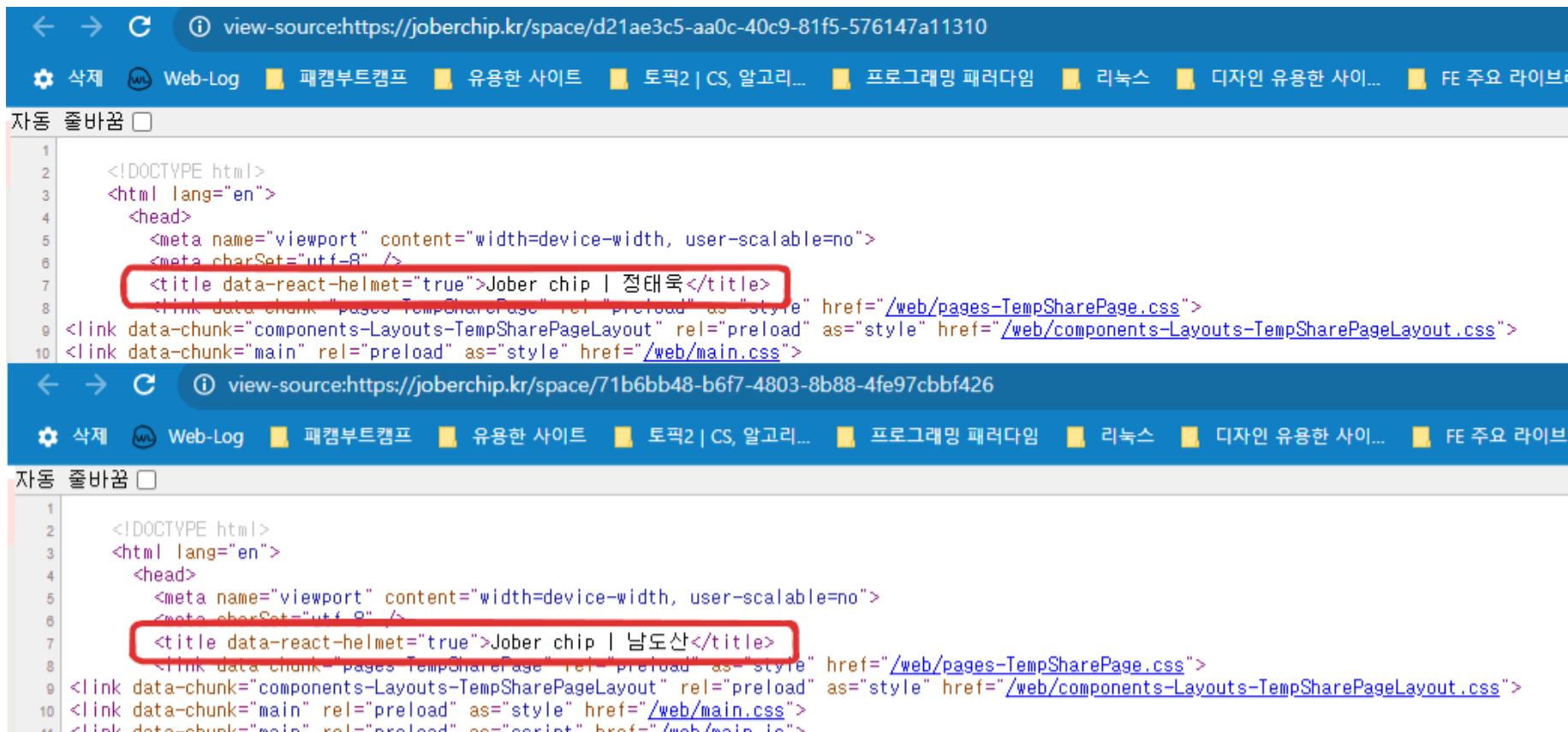
`renderToString`을 사용하여 SSR로 공유 페이지를 로드



# 성능 개선

## Case 2. react-router-dom의 useNavigate Link를 사용하여 페이지를 이동할 경우 (Client Side Rendering)

오른쪽 Sequence diagram과 같이 Hydration 이후 백엔드 서버에 API를 요청하여 페이지 정보를 가져옴



```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="viewport" content="width=device-width, user-scalable=no">
  <meta charset="utf-8" />
  <title data-react-helmet="true">Jober chip | 정태록</title>
  <link data-chunk="pages-TempSharePage" rel="preload" as="style" href="/web/pages-TempSharePage.css">
<link data-chunk="components-Layouts-TempSharePageLayout" rel="preload" as="style" href="/web/components-Layouts-TempSharePageLayout.css">
<link data-chunk="main" rel="preload" as="style" href="/web/main.css">

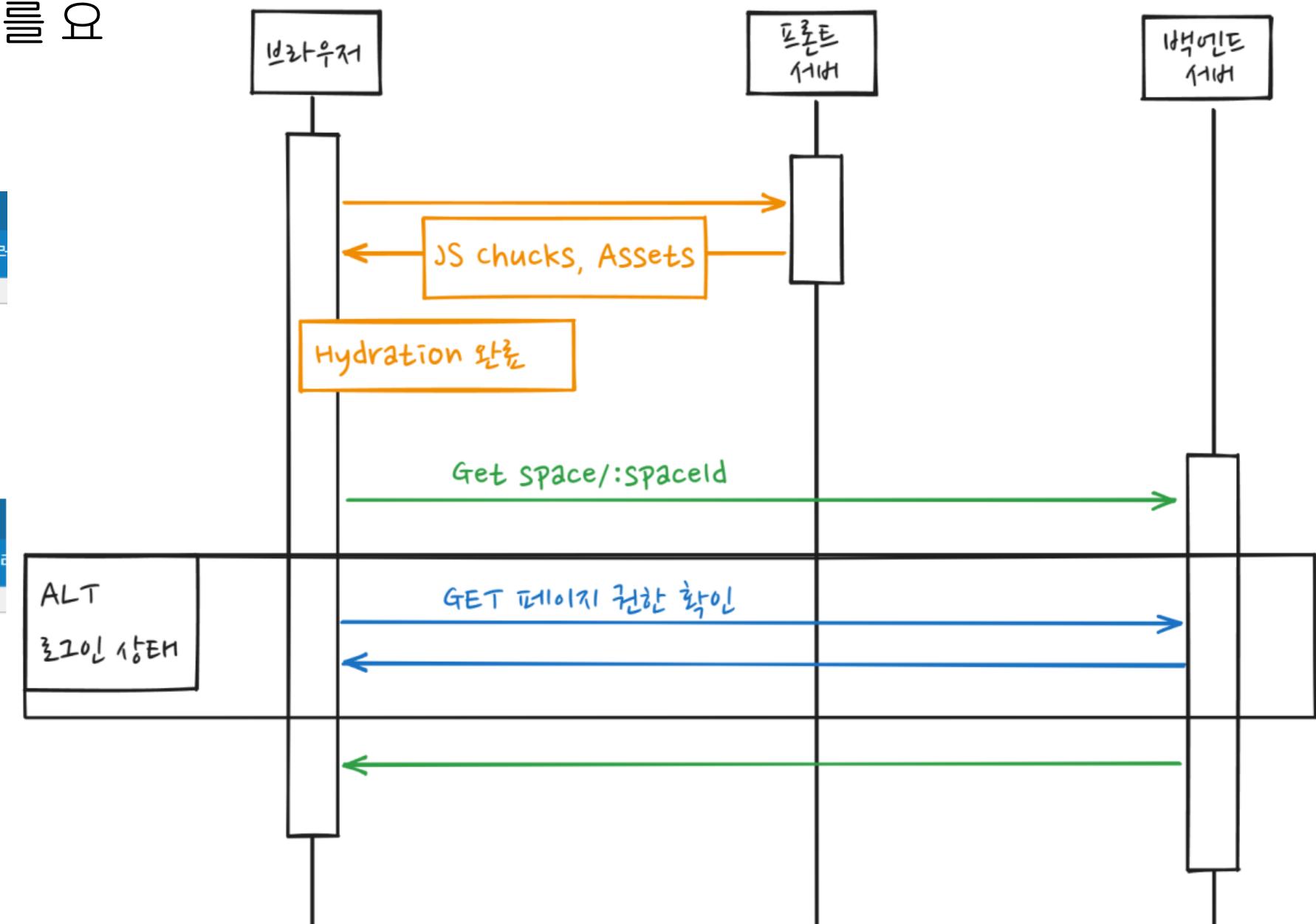
```

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="viewport" content="width=device-width, user-scalable=no">
  <meta charset="utf-8" />
  <title data-react-helmet="true">Jober chip | 남도산</title>
  <link data-chunk="pages-TempSharePage" rel="preload" as="style" href="/web/pages-TempSharePage.css">
<link data-chunk="components-Layouts-TempSharePageLayout" rel="preload" as="style" href="/web/components-Layouts-TempSharePageLayout.css">
<link data-chunk="main" rel="preload" as="style" href="/web/main.css">
<link data-chunk="main" rel="preload" as="script" href="/web/main.js">

```



# 문제점과 해결방안

## HMR(Hot Module Replacement)시 VDOM 트리가 교체되지 않고 쌓이는 문제

HMR를 사용하기 번들링을 두 번씩 해서 dist/web/\*, dist/node/\* 디렉터리 안에 static assets를 넣는 형태로 개발하였으나 dist/node/ chunks의 key와 dist/web/\* chunks의 key가 일치하지 않아

VDOM 트리가 교체되지 않고 계속해서 쌓이는 문제가 발생하여

MHR를 accept한 뒤 HydrateRoot의 첫번째 요소를 지우는 방식으로

문제를 해결

해당 이슈를 정리한 링크

```

18 void loadableReady() => {
19   hydrateRoot(
20     document.getElementById('root') as HTMLDivElement,
21     <BrowserRouter>
22       <QueryClientProvider client={queryClient}>
23         <App />
24         {/* <ReactQueryDevtools initialIsOpen={false} /> */}
25       </QueryClientProvider>
26     </BrowserRouter>
27   )
28 }
29
30 // https://webpack.kr/api/hot-module-replacement/
31 if (module.hot) {
32   if (process.env.NODE_ENV === 'development') {
33     module.hot.accept()
34     document.querySelector('#root > *')?.remove()
35   }
36 }
37

```

# 문제점과 해결방안

## SSR시 hydration 후 컴포넌트의 내용이 사라지는 문제

Next.js에서 `<script id="__NEXT_DATA__" type="application/json">{"props": {"pageProps": ...}}</script>`

형태로 SSR시 pre-rendering하는 것에서

아이디어를 얻음

우측 상단의 두 이미지처럼 API의 응답 결과를  
HTML 파일에 serialize 하여 삽입하고

우측 하단 이미지처럼 contextAPI를

사용한 hook을 만들어 hydration 이후 컴포넌트의  
내용이 유지되도록 개발함으로 문제를 해결

```
function useServerSideProps(): { isServerSide: boolean }
function useServerSideProps<T>(key: string): { isServerSide: boolean; source: T }
function useServerSideProps<T>(key?: string) {
  const [isServerSide, setIsServerSide] = useState(typeof window === 'undefined')
  const ctx = useContext(ServerSideContext)

  useEffect(() => {
    setIsServerSide(typeof window === 'undefined')
  }, [])

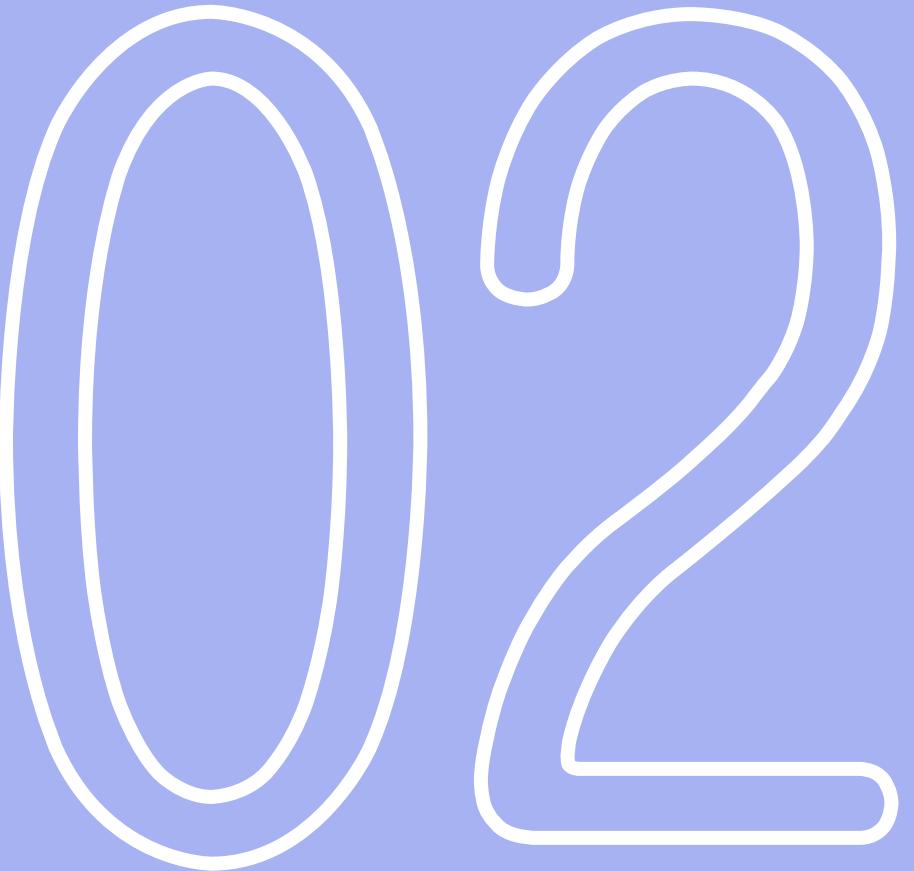
  if (!key) return { isServerSide }

  // * This scope is only for server side rendering
  if (isServerSide) {
    const source: T = JSON.parse((ctx[key] as string) || '{}')
    return { isServerSide, source }
  }

  const serverSideData = document.getElementById('__SERVER_DATA__')?.textContent ?? '{}'
  const data = JSON.parse(htmlEntitiesDecoder(serverSideData))
  const source: T = JSON.parse(data[key] ?? '{}')
  return { isServerSide, source }
}
```

# 개인 기술 블로그

Next.js 13.4의 서버 컴포넌트와 ISR로 동작함으로  
SSG의 단점인 리빌드와, SSR의 단점인 느린 FMP와 prefetch 시  
빈번한 API 호출을 보완했습니다.



프로젝트명 : web-log

기간 : 2023.04.01 ~ 현재

Github :

깃허브 저장소로 이동

배포

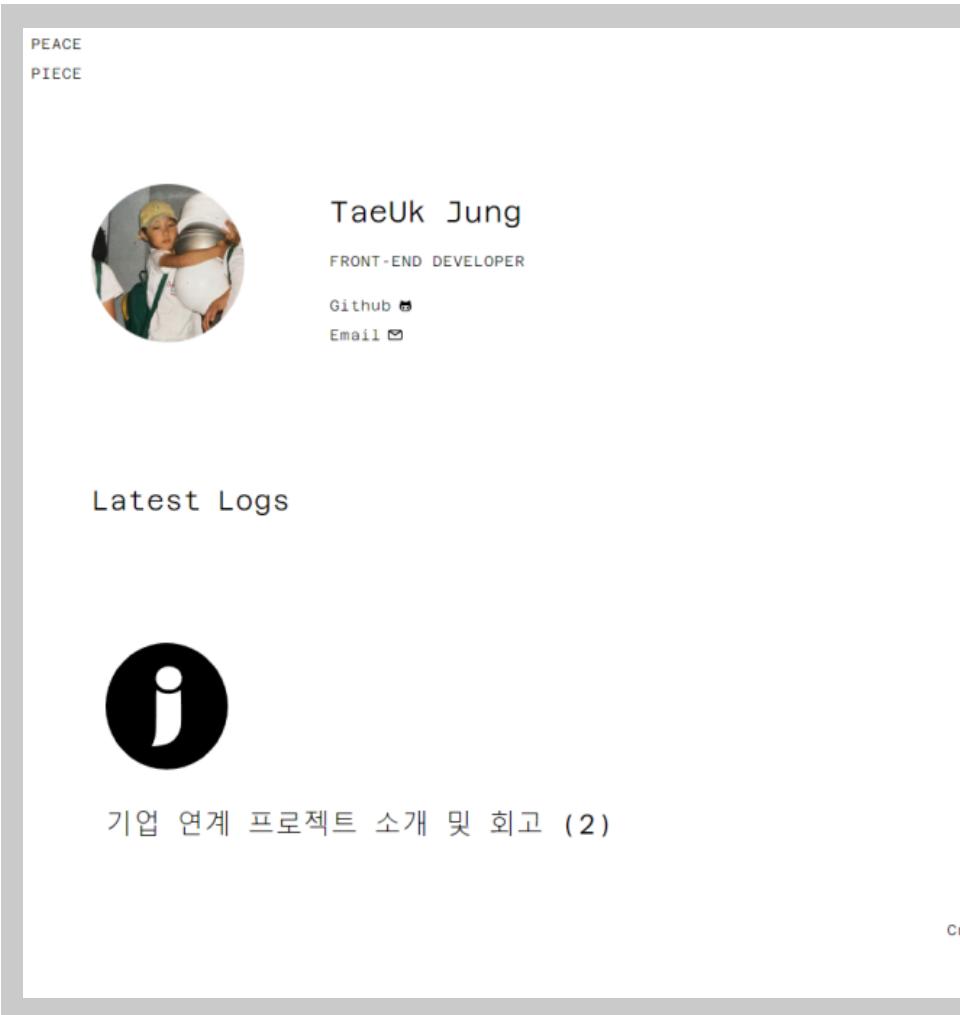
렌딩 페이지로 이동

사용기술

Front-end : Next.js 13.4, Typescript, Tailwind CSS

Database : Firebase storage/Firebase

Deploy : Vercel



숨막조를 위한 서버 환경 구성하기

## Table of Contents

- AWS EC2 인스턴스 생성
- SSH 접속
- Node.js 설치
- PPA를 이용한 설치
- apt로 설치
- NVM (Node Version Manager)로 설치
- NPM 업데이트
- Git 설치

Fast campus 숨막조가 같이 프로젝트를 진행하기 위해 서버를 설치하고 프로토콜을 정해온 과정까지를 작성했습니다.  
운영체제는 ubuntu 22.04-and64, Node.js는 14버전을 설치했습니다.

### AWS EC2 인스턴스 생성

- EC2로 인스턴스 생성
- 안바운드 규칙에 SSH, HTTP, HTTPS 허용
- key pair 생성

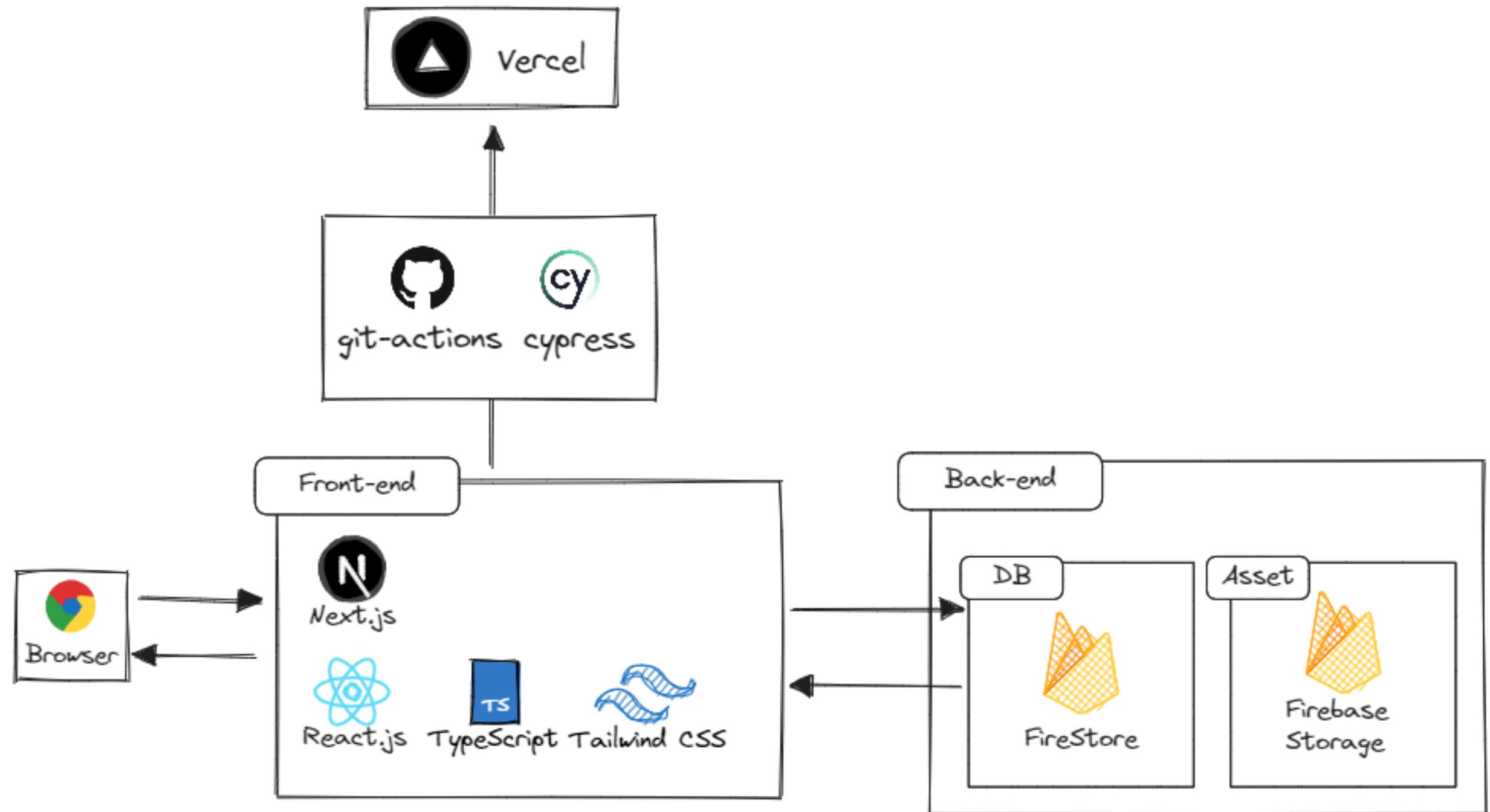
### SSH 접속

"SSH에 대해서 모르는 경우 참고해 주세요"

"다른 사람의 컴퓨터(서버)에 접속하기 위해서 사용하는 대표적인 프로그램으로 구글 원격 데스크톱이나 윈도우의 원격 연결, anyDesk, teamViewer 같은 프로그램이 있습니다. 하지만 이런 프로그램은 보안 층면에서 악수가 있습니다. 누가 컴퓨터를 훔쳐서 전송하거나 아이

# 아키텍처

- 무료 크래딧 한도 내에서 DB, 웹 애플리케이션 호스팅이 되도록 Vercel, Firebase SDK 사용
- DB 호출량을 최소한으로 줄일 수 있도록 Next.js를 사용하여 ISR로 배포
- git-actions, cypress를 사용하여  
블로그 배포 전 E2E 테스트 진행



다양 가능

## ISR에 맞게 fetch 기능 구현

서버 컴포넌트에서 프론트 서버로 GET 요청을 할 경우 빌드 시점에는 프론트 서버가 닫혀있기 때문에 SDK를 직접 호출하도록 구현

[getLogsFetcher를 작성한 코드 링크](#)

```
You, 3 hours ago | 1 author (You)
1 import 'server-only'
2 import { getDocument, getDocuments } from '@/service/firebase/collection'
3 import { getStorageContent } from '@/service/firebase/storage'
4 import { fetcher } from '@/utils/api'
5
6 export const getLogsFetcher = async <T>(url: string, options?: RequestInit) =>
7   if (process.env.NODE_BUILD === 'build') {
8     return await getDocuments<T>('logs')
9   }
0   const { data }: { data: T } = await fetcher(url, options)
1   return data
2 }
```

# markdown viewer/editor

markdown-it을 사용하여 markdown으로 작성할 수 있는 editor와 viewer 구현

Admin Edit Post

기업 연계 프로젝트 소개 및 회고 | [Reset](#)

[React](#) [Webpack](#) [Log](#) [Add Tag](#) [Reset](#)

[Reset Content](#)

```
# [개요](#개요)
PR과 UI/UX, BE, FE가 모두 참여한 프로젝트로
9 ~ 10월 약 한 달간 진행된 기업 연계 프로젝트의 회고를 말씀 드립니다.

# [회고](#회고)
## [ 들어 봄비 ](#들어_봄비)
"글 유 퍼이지"라는 말이 자주 나오는데 이는 slack의 "Space"와 비슷한 개념이다.
이 페이지는 접속하는 사용자를 신뢰 후 퍼이지에 대한 권한이 주어진다.

"Block"이란 글 유 퍼이지 내 사용되는 요소를 통하는 말이다.

![explain](https://peacepiece7.github.io/blog-static-assets/fcm_final_project/review_1.png?large)

## [암축스케줄링 즐기기](#암축스케줄링_즐기기)
### [Block을 단일 풀의](#Block을_단일_풀의)
Block은 단일 풀에 나눠야 풀能把 있게 만들 수 있음
풀을 많이 활용해나갈수록 풀을 나눌 가능성이 커짐
설계 단계부터 계획까지 활용 풀을 했다.

'BlockWith<T>'로 block의 타입을 스위치 해서 컴포넌트별로 필요한 타입을 스위치 할 수 있게 했던 것.
// models/space.ts
export type BlockWith<T> = 
  T extends TText
  ? TextBox
  : T extends TImage
  ? ImageBlock
  : T extends TLink
  ? LinkBlock
  : T extends TPage
  ? PageBlock
  //...rest of cases
  : never
```
아래와 같은 예제 스위치 케이스를 만들어서 풀에 대한 글을 토대로 카드와 컴포넌트별 활용을 달렸다.
// components/SwitchCase/ViewerBox.tsx
<SwitchCase>
  <case value={type}>
    <SwitchCase>
      <case value="Text">
        <TextBlock>
        </TextBlock>
      </case>
      <case value="Image">
        <ImageBlock>
        </ImageBlock>
      </case>
      <case value="Link">
        <LinkBlock>
        </LinkBlock>
      </case>
      <case value="Page">
        <PageBlock>
        </PageBlock>
      </case>
    </SwitchCase>
  </case>
</SwitchCase>
```

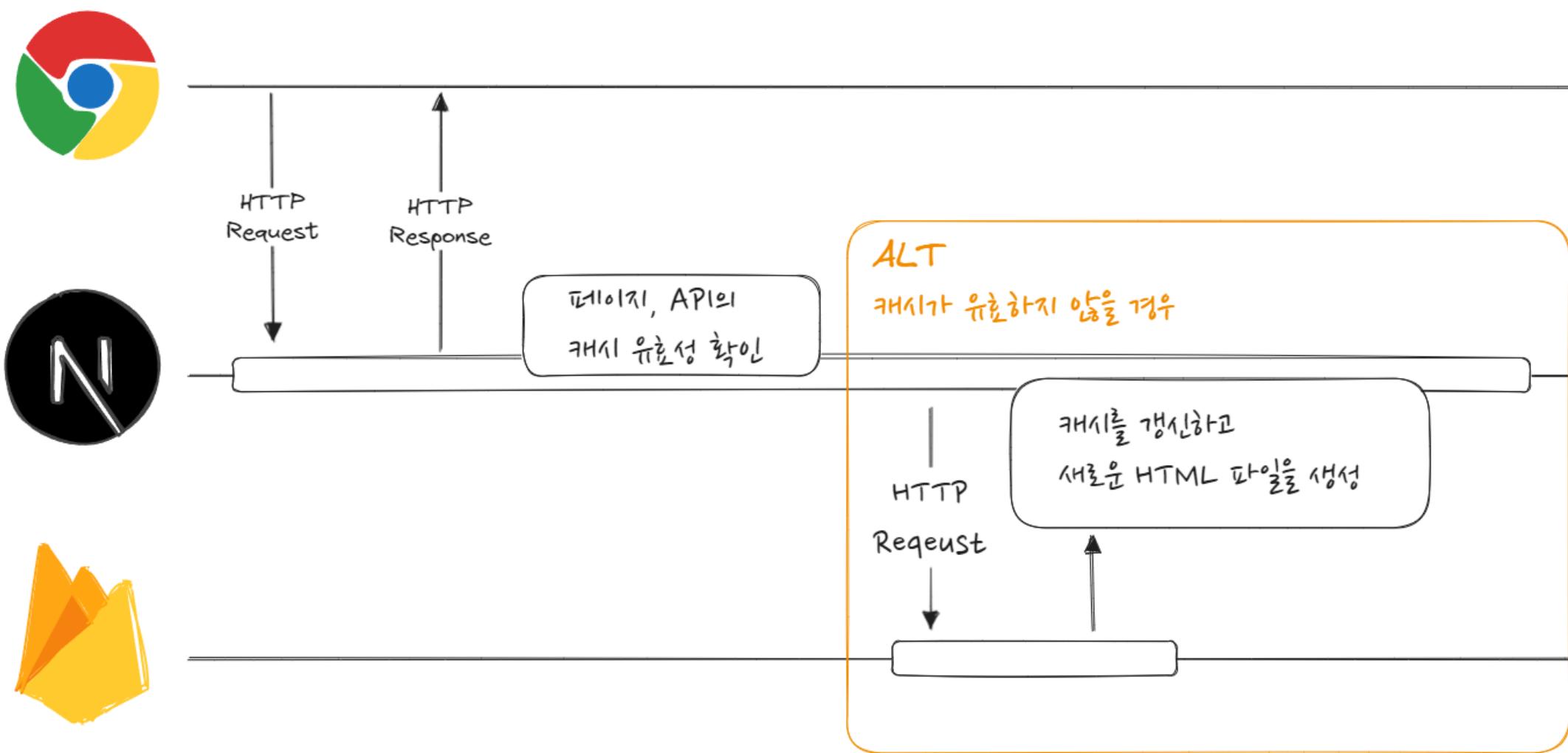
```



# 성능 개선

## ISR (Incremental Static Regeneration)

- 우측 사진과 같이 route segment를 사용해서 full route cache를 하고 fetch API에 revalidate을 주어 data cache 적용  
활성 사용자 수에 관계 없이 첫 요청만 firebase SDK가 호출되도록 ISR 구현



```

export const dynamic = 'force-static'
export const revalidate = PAGE_REVALIDATE_TIME

export default async function Home() {
  const [logs, thumbs] = await Promise.all([
    getLogsFetcher<Log[]>('api/logs', {
      next: {
        revalidate: API_REVALIDATE_TIME,
        tags: [LOGS_TAG]
      }
    }),
    getThumbsFetcher<Thumb[]>('api/thumbs', {
      next: {
        revalidate: API_REVALIDATE_TIME,
        tags: [LOGS_TAG]
      }
    })
  ])

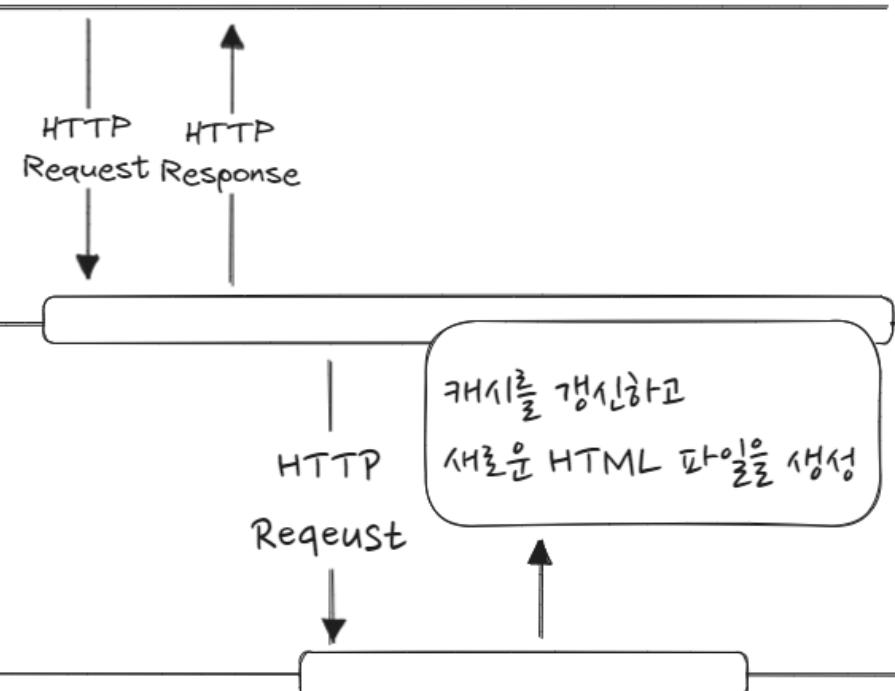
  if (!logs || !thumbs) notFound()
}

```

# 성능 개선

## ISR (Incremental Static Regeneration)

2. 포스트가 추가, 삭제, 수정될 경우 next/cache의 revalidatePath를 사용하여 revalidate time이 무효화되어 캐시가 갱신되도록 구현함으로 데이터가 변경될 때마다 revalidate time을 기다리거나 새로 빌드해야하는 과정을 스킵할 수 있게 됨



```

// * 컨텐츠 내용 수정
export async function POST(request: Request) {
  try {
    const { storagePath, content }: UpdateContent = await updateStorageContent(storagePath, content)
    // * 캐시 삭제
    // revalidateTag(LOGS_TAG)
    revalidatePath('/')

    return NextResponse.json(
      { status: 'success', message: '컨텐츠가 수정되었습니다.' },
      {
        status: 200,
        headers: {
          'Access-Control-Allow-Origin': '*'
        }
      }
    )
  } catch (error) {
    console.error(error)
    return NextResponse.json({ status: 'error', message: '컨텐츠 수정 중 오류가 발생했습니다.' }, { status: 500 })
  }
}
  
```

# 문제점과 해결방안

## React.cache를 사용한 캐싱 문제

React.cache를 사용해서 firebase SDK를 직접 호출할 경우

아래 문제점이 발생

- 반영구적인 데이터 캐싱이 되지 않음
- 렌더링 시 최소 한 번의 호출 발생
- SSG로 개발시 firebase SDK 사용량이 무료 사용량을 넘어감

이를 해결하기 위해 **firebase SDK를 route handler에 wrap하고**

**fetch API를 사용해서 데이터를 캐싱** 추가적으로 빌드 시 프론트 서버  
가 꺼져있음을 고려하여 우측 사진과 같이 get 요청에 대한 fetcher 함  
수를 구현

React.cache 캐싱 관련 문제 정리 링크

```

5
6 // * Update Log API
7 export async function POST(request: Request) {
8   try {
9     const log: Log = await request.json()
10    const { id, ...logData } = log
11    await setDocument<LogDocument>('logs', id, logData)
12    // * 캐시 삭제
13    // revalidateTag(LOGS_TAG)
14    revalidatePath('/')
15    return NextResponse.json(
16      { state: 'success', data: null, message: '로그가 수정되었습니다.' },
17      {
18        status: 200,
19        headers: {
20          'Access-Control-Allow-Origin': '*'
21        }
22      }
23    )
24  } catch (error) {
import 'server-only'
import { getDocument, getDocuments } from '@/service/firebase/collection'
import { getStorageContent } from '@/service/firebase/storage'
import { fetcher } from '@/utils/api'

export const getLogsFetcher = async <T>(url: string, options?: RequestInit) => {
  if (process.env.NODE_BUILD === 'build') {
    return await getDocuments<T>('logs')
  }
  const { data }: { data: T } = await fetcher(url, options)
  return data
}

```



## 프로젝트명 : Allmanual

기간 : 2022.07.24 ~ 2023.08.11

인원 : 1명 ~ 3명

역할

- Elastic search와 IDC에 서버를 설치하는 과정을 제외한 모든 개발

배포

<https://allmanual.com>

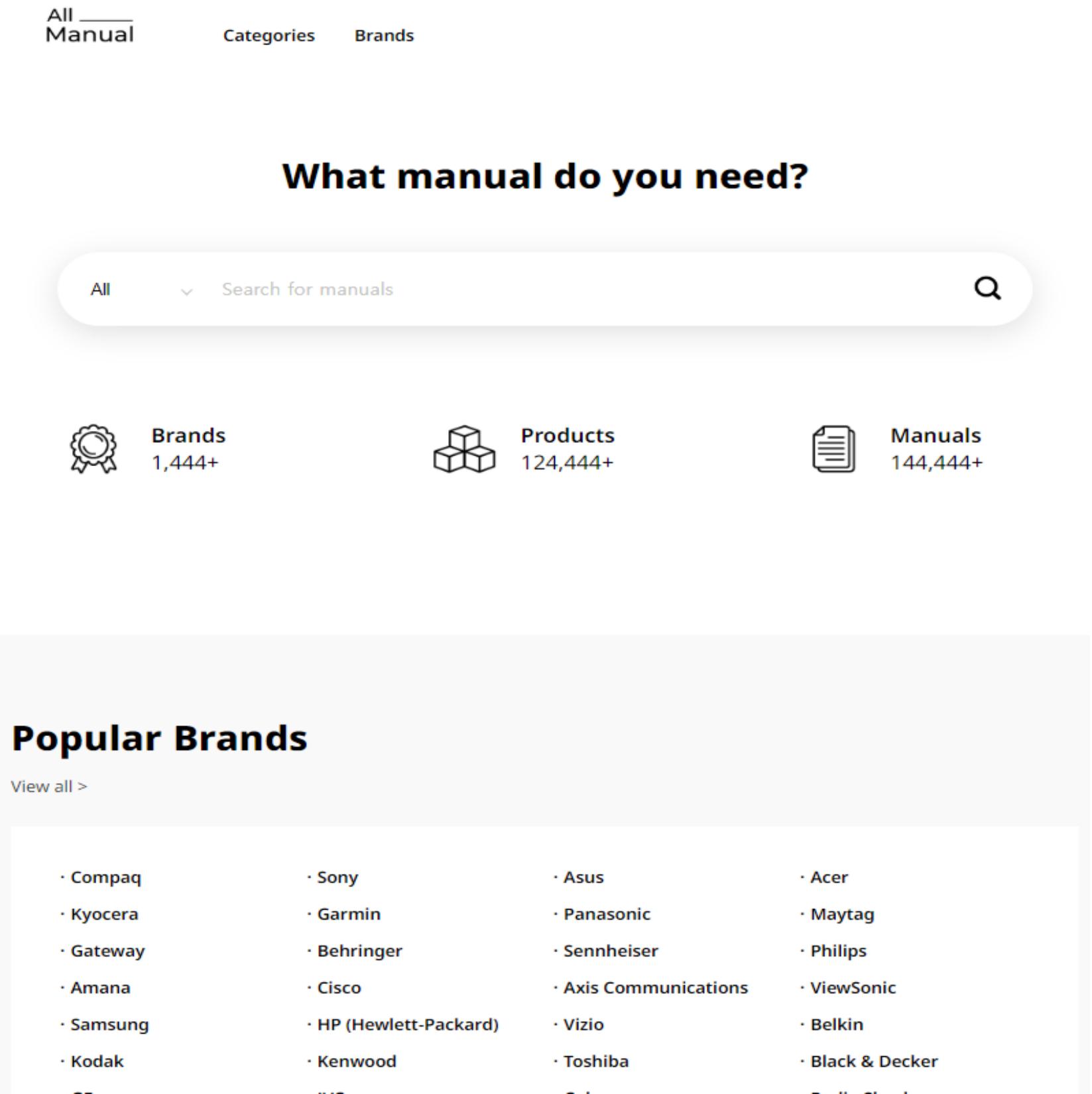
사용기술

Front-end : Javascript, CSS Module, Next.js, react-query

## Back-end : Express.js

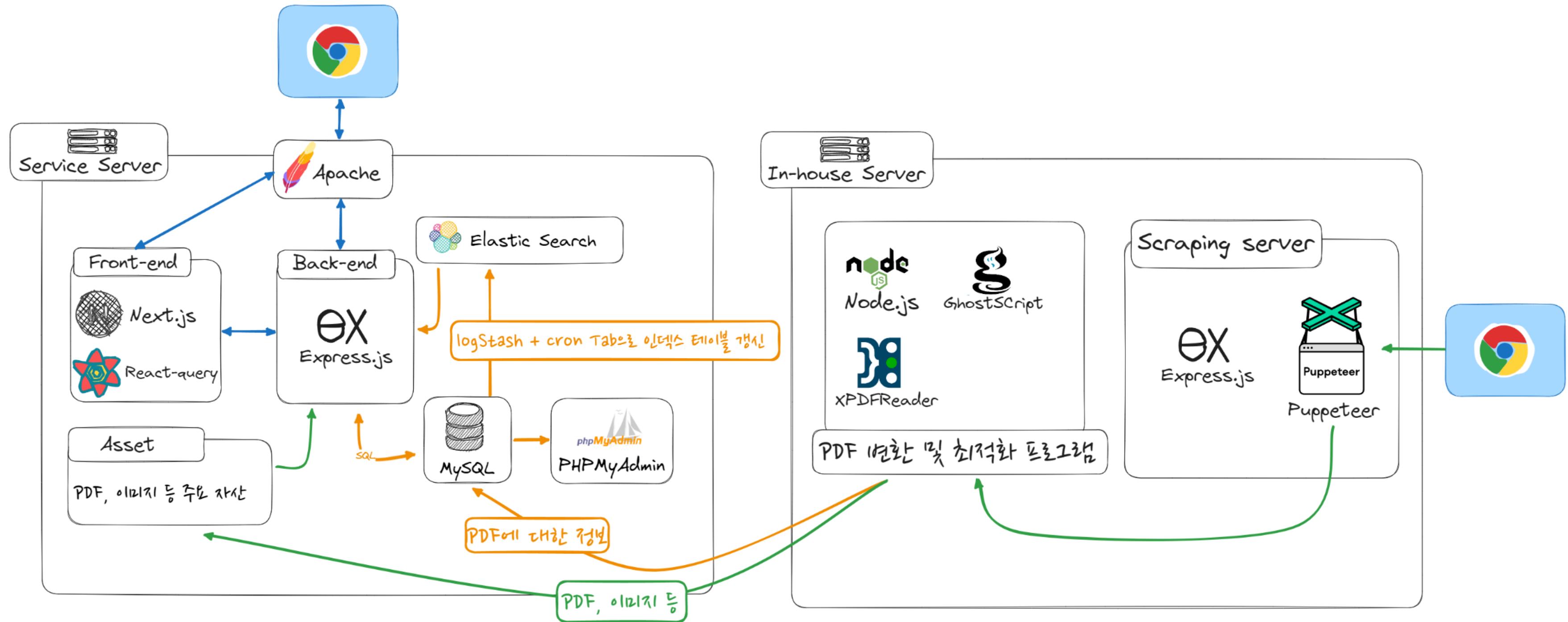
# Database : MySQL Deploy : IDC

Etc : XPDFReader, GhostScript, Puppeteer



# 아키텍처

- On-premise 환경에서 하나의 서버를 사용해야 하지만 프로젝트가 커짐에 따라 BE, FE를 나눌 수 있도록 설계
- 많은 페이지가 정적으로 소비되고, 일부 동적인 페이지도 존재하기 때문에 유연하게 대처할 수 있도록 Next.js 사용



# 담당 가능

## 빌드 시 사이트맵을 생성하는 스크립트 개발

기존 sitemap을 google console에 수동으로 복사해서 붙여넣는 방식에서  
 sitemap preloader를 만들고, 사이트맵 색인 파일로 사이트맵을 관리하면서 빌드 시 자동으로 사이트맵이 추가되도록 개발  
 빌드 시 prebuild 스크립트의 sitemapPreloader.js 스크립트 실행

```
"prebuild": "jsdoc2md ./typedef/index.js >> ./README.TYPEDEF.md && node ./sitemapPreloader.js",
"postbuild": "next-sitemap --config sitemap.config.js",
...
```

백엔드에서 url 추출 후 완성된 사이트맵을  
 public 디렉터리에 저장하고 사이트맵 링크를  
 sitemap.config.js에 추가

next-sitemap으로 사이트맵 색인 파일인 sitemap.xml 생성

```
async function init() {
  await buildBrandSitemap()
  await buildBrandsSitemap()
  await buildCategorySitemap()
  await buildViewPageSitemap()
  const config = {
    siteUrl: 'https://allmanual.com',
    sitemapSize: 4000,
    changefreq: 'weekly',
    priority: 1.0,
    generateRobotsTxt: true,
    autoLastmod: false,
    exclude: ['/search'],
    robotsTxtOptions: {
      additionalSitemaps: [],
    },
  }
  fs.readdirSync('./public')
    .filter((file) => file.includes('sitemap-') && file.includes('.xml'))
    .map((sitemap) => config.robotsTxtOptions.additionalSitemaps.push(`https://allmanual.com/${sitemap}`))
  fs.writeFileSync('./sitemap.config.js', `/** @type {import('next-sitemap').IConfig} */\nmodule.exports=${JSON.stringify(config)}`)
}
```

# 담당 가능

## PDF 가공 스크립트 개발

Node.js에서 제공하는 child\_process를 사용해  
여러 라이브러리를 스크립트 프로세스안에서 동기적으로 실행

### 아래 기능을 수행하는 PDF 가공 스크립트 개발

- PDF 목차 추출
- 워터마크 추가
- 메타 데이터 삽입
- PDF를 HTML로 변환
- PDF를 이미지로 변환
- HTML 구조 최적화 (uglify)

```

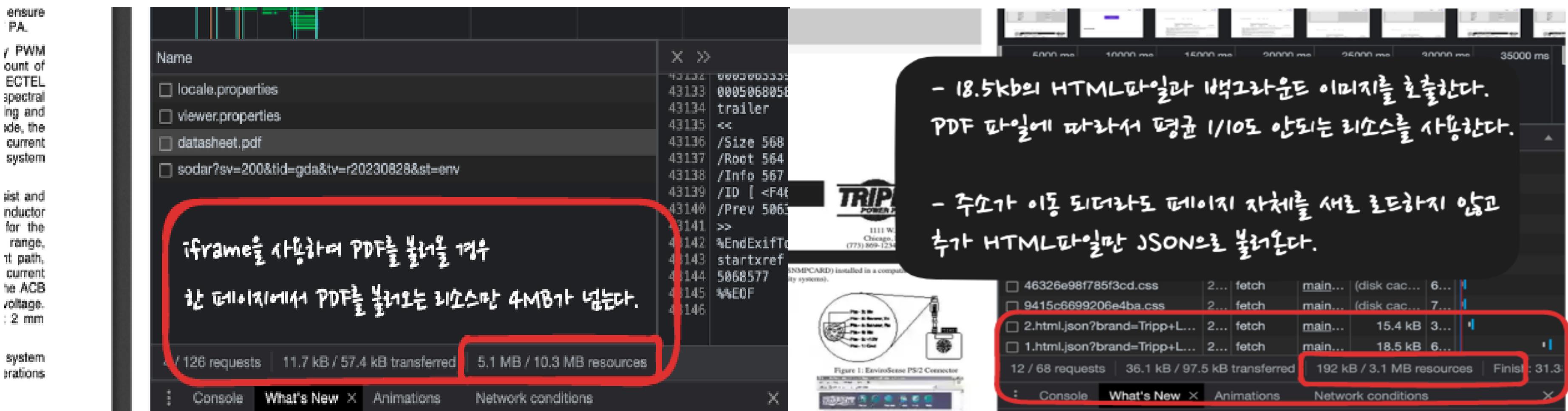
async function init() {
  try {
    const rl = readline.createInterface({
      input: process.stdin,
      output: process.stdout,
    })
    const dir = await new Promise((resolve, reject) => {
      rl.question('index.js : Enter a directory : ', (answer) => resolve(answer))
    })
    let mfrs = fs.readdirSync(dir, { withFileTypes: true }).filter((v) => v.isDirectory())
    console.log(`\n\n 가공 경로에 공백이나 한글이 포함되면 워터 마크가 지워지지 않습니다.`)
    const start = new Date()
    await new Promise((res) => setTimeout(() => res(true), 2000))
    for (let { name: mfr } of mfrs) {
      console.log(`${dir}/${mfr}`)
      await new Promise((res) => setTimeout(() => res(true), 2000))
      await decryptPassword(`${dir}/${mfr}`)
      await extractTOCPrev(`${dir}/${mfr}`)
      await extractTOCNext(`${dir}/${mfr}`)
      await convertPDFtoHTML(`${dir}/${mfr}`)
      await convertPDFtoPNG(`${dir}/${mfr}`)
      await convertLargeImage(`${dir}/${mfr}`)
      await reducePDFHeight(`${dir}/${mfr}`)
      await convertPDFtoPS(`${dir}/${mfr}`)
      await erasePSWaterMark(`${dir}/${mfr}`)
      await convertPS2PDF(`${dir}/${mfr}`)
      await pngToWebp(`${dir}/${mfr}`)
      await XMLtoHTML(`${dir}/${mfr}`)
      await optimizeHTMLStructure(`${dir}/${mfr}`)
      await cleanUp(`${dir}/${mfr}`)
      await findErrorFiles(`${dir}/${mfr}`)
    }
  }
}

```

# 성능 개선

## PDF 용량 최적화

- PDF를 HTML, CSS 파일로 분리, 최적화하여 네트워크 리소스가 50% 이상 감소
- PDF의 내용이 HTML Tag로 페이지 소스에 추가 SEO 향상



감사합니다