# Webpack Module-Federation 을 이용한 Micro-Frontend 아키텍처 구현

# Micro-Frontend Architecture Implementation Using Webpack Module-Federation

본 논문은 2022 학년 12 월에 제출된
한국외국어대학교 컴퓨터공학부
졸업논문이다. 2021.09.28

지도교수: 김낙현

서명: _____

참여한 캡스톤 설계 (해당자만)

설계명: 마이크로프론트엔드 UI/UX
프레임워크 구축

팀원명: 황수민, 정주현, 남기훈, 김준성

**황수민 (Sumin-Hwang)** [1]

[1] 한국외국어대학교, 컴퓨터공학부, kie6974@gmail.com

**한글 요약:** 기존에 사용되고 있는 Monolithic frontend 아키텍처는 단일 프로그램으로 결합되는 단일 계층 소프트웨어 응용 프로그램이다. 이러한 방식은 부분적인 기능 개발과 개선 및 업데이트에 많은 시간이 걸린다는 치명적인 단점이 존재한다. 이러한 문제를 해결하고자 등장한 아키텍처가 Micro-Frontend 아키텍처이다. Micro-Frontend 아키텍처를 구현하는 방법에는 Single-SPA, Vue-genesis, Webpack 등 다양한 방법이 존재하는데, 본 논문에서는 Webpack 5 에서 제공하는 module-federation 을 이용하여 Micro-Frontend 구조의 댓글 컴포넌트를 구현한다. 이러한 아키텍처는 제품을 더 작고 단순한 어플리케이션의 단위로 분할하여 독립적이고 자율적인 팀 개발 환경을 제공하는 장점을 가진다.

**핵심어:** Micro-Frontend, Webpack, 프론트엔드

**영문 요약:** The monolithic frontend architecture in use is a single-layer software application that combines into a single program. This method has a fatal disadvantage that partial function development, improvement, and update take a lot of time. The architecture that emerged to solve these problems is the Micro-Frontend architecture. There are various methods for implementing the Micro-Frontend architecture, such as Single-SPA, Vue-genesis, and Webpack, and in this paper, the module-federation provided by Webpack 5 is used to implement the Micro-Frontend structure Comment component. This architecture has the advantage of providing an independent and autonomous team development environment by dividing products into smaller and simpler application units.

**Keywords:** Micro-Frontend, Webpack, Frontend

## 1. 서론 - Introduction

기존의 Monolithic frontend 는 사용자 인터페이스와 데이터 엑세스 코드가 단일 플랫폼에서 단일 프로그램으로 결합되는 단일 계층 소프트웨어 응용 프로그램이다. 그러나 이러한 방식은 개발자로 하여금 독립적인 개발 진행을 어렵게 하고, 부분적인 기능 개선과 업데이트에 많은 시간이 걸린다는 단점이 있다. 이러한 문제를 해결하고자 등장한 것이 Micro-frontend 아키텍처이다.

Micro-frontend 란 제품을 더 작고 단순한 어플리케이션의 단위로 분할을 통해 독립적이고 자율적인 팀 개발 환경을 제공하는 하나의 접근 방식이다. 위의 아키텍처는 사용하기가 간단하고, 기존의 monolithic frontend 와 비교하였을 때 관리 측면에서 간편하다는 특징을 가지며, 독립적인 개발을 보장하여 어플리케이션을 공동으로 작업하기 용이하다는 장점이 있다.

본 논문에서는 오픈 소스 자바스크립트 모듈 번들러인 Webpack 에서 지원하는 plugin 중 하나인 module-federation 을 이용하여 Micro-frontend 댓글 컴포넌트를 제작한다. 이러한 형태로 제작된 컴포넌트는 독자적인 개발과 배치가 가능하며, 여러 곳에서 코드를 재사용하기에 용이하다. 또한, 각각의 컴포넌트에 서로 다른 기술(javascript, React, Vue, Angular 등)을 사용하는 것이 가능하다.

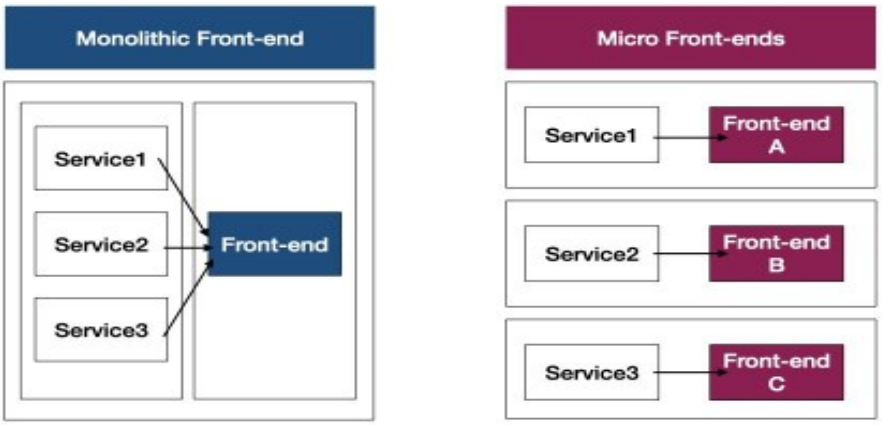
논문의 구성은 Monolithic Front-end 와 Micro-frontend 에 대한 비교와, Micro-frontend 구조 및 특성에 대한 설명을 시작으로, 일반적인 Micro-frontend 아키텍쳐 구현을 위한 Server-Side Composition 의 Reverse proxy 와 Client-Side Composition 에 대한 구조와 개념을 언급한다. 이 중 Client-Side Composition 방식의 Webpack module-federation 의 장점과 간단한 이용방법을 알아본 뒤, 이를 이용하여 댓글 컴포넌트를 구현하는 방법에 대해서 설명하는 순서로 구성되어 있다.

## 2. 연구 방법 및 결과

### 2.1. Micro-frontend 아키텍처

#### 2.1.1. Micro-Frontend

**a. Micro-Frontend 구조 및 특징**




**(a) Monolithic Front-end** **(b) Micro Front-ends**

(b) Micro-frontend 아키텍처의 특징

- Micro-frontend 아키텍처는 사용하기 간단하고, 비교적 관리하기가 쉽다.
- 독립적인 개발팀이 훨씬 쉽게 frontend 어플리케이션을 공동으로 작업하기 용이하다.
- 하나의 코드를 여러 곳에서 재사용이 가능하다.
- 각각의 컴포넌트에 서로 다른 기술(JavaScript, React, Vue, Angular 등)을 사용 가능하다.
- 그림 1.(b)와 같이 기존의 단일화된 코드에서 분할된 애플리케이션의 간소화된 코드가 팀에 합류하는 개발자들의 접근성을 낮추는 것에 기여한다.

**b. Micro-frontend 구현 방법**

**Server-Side Composition – Reverse Proxy**

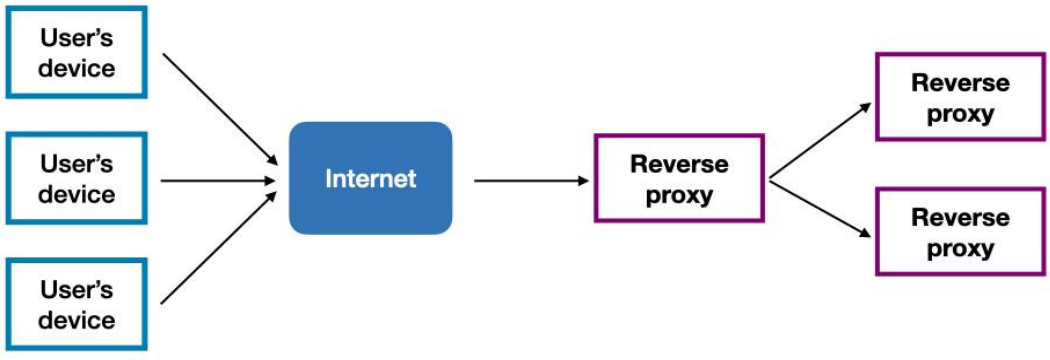



**그림 2. Reverse Proxy 동작방식**

그림 2 에서는 Reverse Proxy 의 동작을 설명한다. Forward Proxy 와 반대로 하나 이상의 웹 서버 앞에 존재하는 클라이언트의 요청을 가로채는 서버이다. 이는 프록시가 클라이언트 앞에 있는 Forward Proxy 와 다른 점이다.

리버스 프록시는 다음과 같은 목적으로 사용된다.

- 로드 밸런싱(Load balancing) :
  사용자가 많은 웹사이트일 경우, 리버스 프록시의 로드 밸런싱 솔루션을 제공하여 들어오는 트래픽을 서로 다른 서버로 균등하게 분산시켜 단일 서버가 과부화 되는 것을 방지할 수 있다.

- 외부 공격으로부터의 방어 :
  리버스 프록시가 설치된 웹 사이트나 서비스는 원본 서버의 ip 주소를 표시할 필요가 없기 때문에 DDOS 공격과 같은 공격을 어렵게 한다.

- GSLB(Global Server Load Balancing) :
  웹사이트가 전 세계의 여러 서버에 배포될수 있으며, 리버스 프록시는 클라이언트를 지리적으로 가장 가까운 서버로 보낸다. 이를 통해 요청과 응답이 이동해야 하는 거리가 줄어들어 로드 시간을 최소화 할 수 있다.

- 캐싱 :
  리버스 프록시는 콘텐츠 캐시에 저장하여 더 빠른 성능을 제공할 수 있다. 예를 들어 파리에 있는 사용자가 로스엔젤레스에 있는 웹 서버의 리버스 프록시 웹사이트를 방문할 경우, 사용자는 실제 파리의 로컬 리버스 프록시 서버에 연결할수 있으며, 이 서버는 L.A 의 오리진 서버와 통신한다. 이때 프록시 서버는 응답 데이터를 일시적으로 저장하여, 이후 사이트를 검색하는 파리 사용자는 파리의 리버스 프록시 서버에서 로컬로 캐시된 버전을 가져와 훨씬 빠른 성능을 얻을 수 있다.

- SSL 암호화 :
  각 클라이언트의 SSL(또는 TLS) 통신을 암호화하고 해독하는 것은 오리진 서버의경우, 비용이 많이 들 수 있다. 이때, 리버스 프록시를 성하여 들어오는 요청을 해독하고, 나가는 든 응답을 암호화하여 원본 서버에서 중요한 리소스를 확보할 수 있다.

일반적으로 리버스 프록시(Reverse proxy)를 이용하여 사용자가 엑세스하는 경로를기준으로 트래픽을 다른 응용 프로그램으로 전달하여 Micro-frontend 아키텍처를 구현할 수 있다.[1]

**Client-Side Composition[1]**

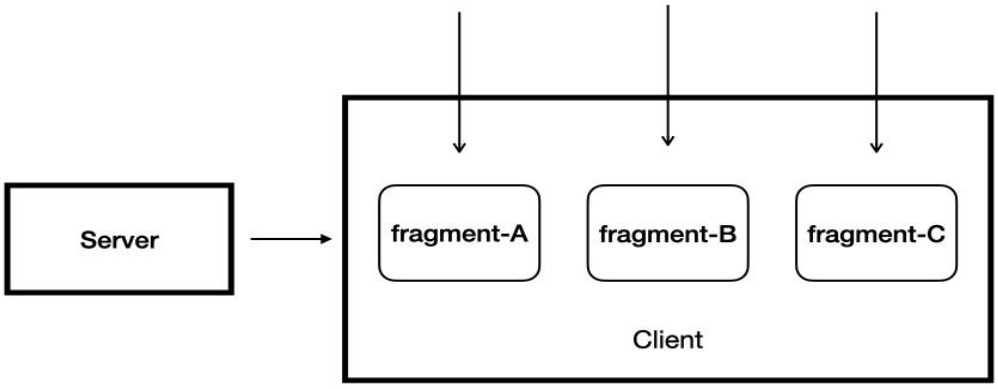



**그림 3. Client-Side Micro-Frontend**

그림 3. Client-Side 에서 Micro-Frontend 의 동작을 보여준다. 각각 별개의 기능을 수행하는 fragment 들이 Client-Side 에서 함께 동작한다.

- 컨테이너들이 개별적으로 개발되고 배치가 가능하다.
- 다양한 패키지를 필요에 따라 컨테이너에 패치 할 수 있다.
- 빌드와 배치를 공유하지 않고, 서로 다른 기술을 사용할 수 있다.

본 논문에서는 Client-Side Composition 방식 중 하나로, 최근 잘 사용되는 Webpack module-federation 을 이용하여 구현할 예정이다.

### 2.1.2. Webpack

**a. Micro-Frontend 와 Webpack**

*Webpack 은 오픈소스 자바스크립트 모듈 번들러이다.*

webpack 은 js 모듈 뿐만 아니라 스타일시트, 이미지 등 모든 것을 자바스크립트 모듈로 로딩해서 사용한다.
이러한 웹팩에서 지원하는 Webpack plugin 중 하나인 Module-Federation 은 Micro Frontend 프레임워크를 구현하는 것에 도움을 준다.

**b. Module-Federation**

자바스크립트 응용프로그램이 다른 응용프로그램에서 동적으로 코드를 가져올 수 있도록 허용하는 자바스크립트 아키텍처이다.

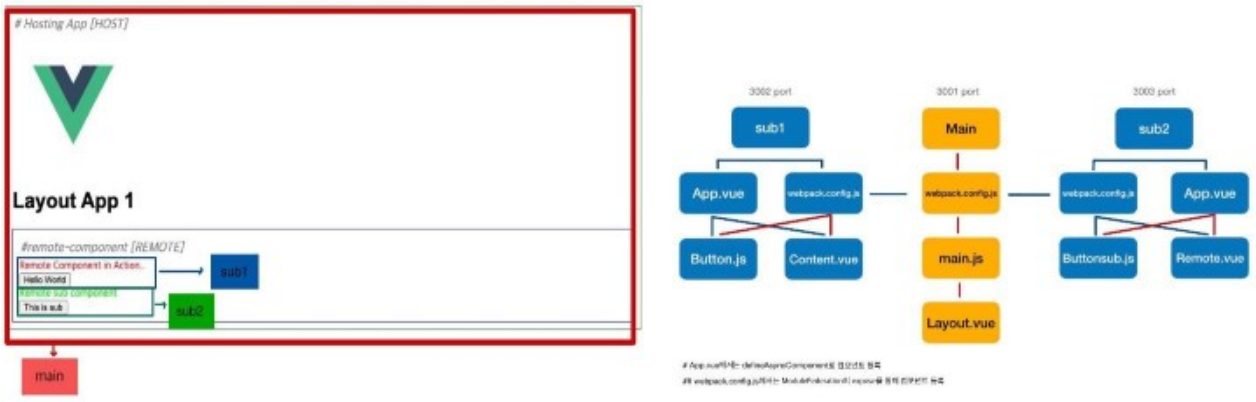
- 여러 개로 분리되어 있는 개별 빌드들이 하나의 어플리케이션 형태를 만들어 각 부분별로 독자적인 개발과 배치를 가능하게 한다.
- 간단하게 라이브러리를 제공 및 공유할 수 있다.
- 동적으로 웹 컴포넌트를 로드하고, 래퍼를 사용하여 웹 컴포넌트로 라우팅할 수 있다.
- 로드된 웹 컴포넌트는 lazy loading 을 사용 가능하다.

*Module-federation 은 흔히 마이크로 프론트엔드라고 알려져 있지만, 이것 하나에만 국한된 기능이 아니다.*

프로젝트의 원활한 진행을 위해 Webpack Module-Federation 원문[2]은 한국어로 번역하여 참고하였다.[3]

### 2.2. Vue3 Webpack Module-federation 예제

Vue3 에서 Module-federation 을 어떻게 활용하는지 알아보기 위해, 먼저 Webpack 에서 예제로 제공하는 코드를분석하고 변형해 보도록 한다.[4]




**(a)** **(b)**

**그림 4. (a) 코드 실행 결과와 (b) 코드 구조**

그림 4.(a)를 참고하였을 때, 전체 main 프론트엔드에 sub1 과 sub2 컴포넌트가 각각 나타나는 것을 볼 수 있다. 이는 그림 4.(b)에서 3002 번 포트의 sub1 과 3003 번 포트 sub2 가 3001 번 포트인 main 에서 동작한다는 것을 확인할 수 있으며, 모두 별개의 포트에서 동작하지만 3001 번인 main 에서 나타난다는 것을 알 수 있다. 이는 webpack.config.js 에서 module-federation 플러그인을 이용하여 각각의 sub 컴포넌트를 main 에 연결하여 Micro-frontend 아키텍처 형태로 구현한 것이다.[4]

Webpack Module-federation 원문[2]에 따르면 main.js 에 등록되어있는 코드를 바로 이용할 경우, module-federation 이슈로 인해 에러가 나타나기 때문에, 별도로 bootstrap.js 파일에 main.js 의 코드를 쓰고, 본래의 main.js 파일에는 import('./bootstrap.js'); 를 통해 코드를 불러내는 방식으로 사용해야 한다는 것을 주의해야 한다.

### 2.3 Module-federation 댓글 컴포넌트 구현[5]

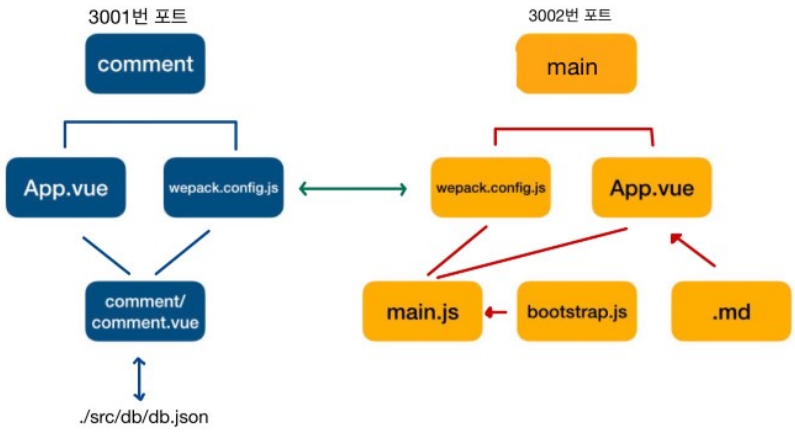



**그림 5. 댓글 컴포넌트와 main 페이지의 연결 구조**

그림 5.에서 3001 번 포트의 댓글 컴포넌트는 3002 번 포트의 main 과 webpack.config.js 의 module-federation 으로 연결되어있다. 하나의 main 에서 각각의 동작들이 개별적으로 동작하는 Micro-Frontend 구조임을 확인할 수 있다.

#### 실행 순서

1. main 의 App.vue 에서 사용할 댓글 컴포넌트를 comment 의 webpack.config.js 의 ModuleFederationPlugin 내부에 expose 로 등록한다.

2. comment 의 webpack.config.js 에 등록된 컴포넌트를 main 에서 이용하기 위해, ModuleFederationPlugin 내부의 remotes 에 해당 컴포넌트의 호스트를 등록한다.

3. main.js 로 import 되어있는 bootstrap.js 파일에서 웹팩 module federation 을 통해 가져온 comment 컴포넌트를 App.vue 에서 사용하기 위해 등록한다.

4. 각각의 프로그램을 로컬 호스트로 올리면, 3001 번에 등록된 컴포넌트를 3002 번 App.vue 에서 사용한다.

#### 프로젝트 Module-federation 기본 사용법[4-5]

**a. vue3-CLI + element UI + tailwindcss**

vue3-CLI 를 기반으로 element UI 와 tailwindcss 를 이용하여 프론트엔드 디자인을 진행한다. element-UI 는 vue 기반 UI toolkit 이며, tailwindcss 는 utility-first 컨셉을 가진 css 프레임워크이다. 편의를 위해 vue3 프레임워크에 element-UI 를 설치해둔 element-plus-starter[7]를 이용한다.

**b. Webpack 적용**

**webpack tailwindcss 적용**

webpack 에서 정상적으로 tailwindcss 가 적용될 수 있도록, 모듈을 추가하고 설정을 변경해야 할 필요가 있다.

- **yarn add -D tailwindcss@yarn:@tailwindcss/postcss7-compat postcss@^7 autoprefixer@^9** 를 통해 postcss plugin 을 사용할 수 있도록 한다.

```
{
      test: /\.css$/,
      use: [
        {
          loader: MiniCssExtractPlugin.loader,
            options: {
              hmr: !env.prod,
              publicPath:''
            },
        },
        "css-loader",'postcss-loader'
   ],
},
```

- 다음으로 webpack.config.js 파일에서 외부의 tailwindcss 가 들어올 때 읽어낼 수 있도록 postcss-loader 를 위와 같이 추가해준다.

**main>webpack.config.js**

```
 plugins: [
   new MiniCssExtractPlugin({
    filename: "[name].css",
   }),
   new ModuleFederationPlugin({
    name: "home",
    filename: "remoteEntry",
    remotes: {
   comment: "comment@http://localhost:3001/remoteEntry",

    },
    exposes: { },
     shared: require("./package.json").dependencies,
   }),
   new HtmlWebpackPlugin({
    template: path.resolve(__dirname, "./index.html"),
    chunks: ["main"],
   }),
   new VueLoaderPlugin(),
  ],
```

- webpack.config.js 파일 plugins 내부의 ModuleFederationPlugin 에서 사용할 3001 번 포트 댓글 컴포넌트를 remotes 를 통해 등록한다. 이를 통해 comment 에서 expose 로 등록된 컴포넌트를 remotes 를 통해 불러온다.

**main>bootstrap.js**

```
import ElementPlus from "element-plus";
import "element-plus/lib/theme-chalk/index.css";
import { createApp, defineAsyncComponent } from "vue";
import App from "./App.vue";
import "./assets/css/tailwind.css";

const Comment = defineAsyncComponent(() => import("comment/Comment"));

const app = createApp(App);

app.component("comment-element", Comment);

app.use(ElementPlus);
app.mount("#app");
```

- 불러낸 comment 컴포넌트를 사용하기 위해 defineAsyncComponent()를 통해 comment 를 import 한다.
- import 한 comment 컴포넌트를 main 에서 활용하기 위해 app 에 등록한다. 이때 컴포넌트의 태그 이름을 직접 지정할 수 있다. 여기서는 comment-element 로 태그명을 지정했다.

**comment>webpack.config.js**

```
plugins: [
   new MiniCssExtractPlugin({
     filename: "[name].css",
   }),
   new ModuleFederationPlugin({
     name: "comment",
     filename: "remoteEntry",
     remotes: {


     },
     exposes: { "./Comment": "./src/components/comments/comment.vue",},
      shared: require("./package.json").dependencies,
   }),
   new HtmlWebpackPlugin({
     template: path.resolve(__dirname, "./index.html"),
     chunks: ["main"],
   }),
   new VueLoaderPlugin(),
```

],

- comment 를 컴포넌트로 외부에서 사용할 수 있도록 ModuleFederationPlugin 내부의 exposes 에서 "./Comment": "./src/components/comments/comment.vue"로 등록한다.
- 이후에 외부에서 컴포넌트를 사용할 때, comment/Comment 로 호출할 수 있다.

## 3. 결과 분석

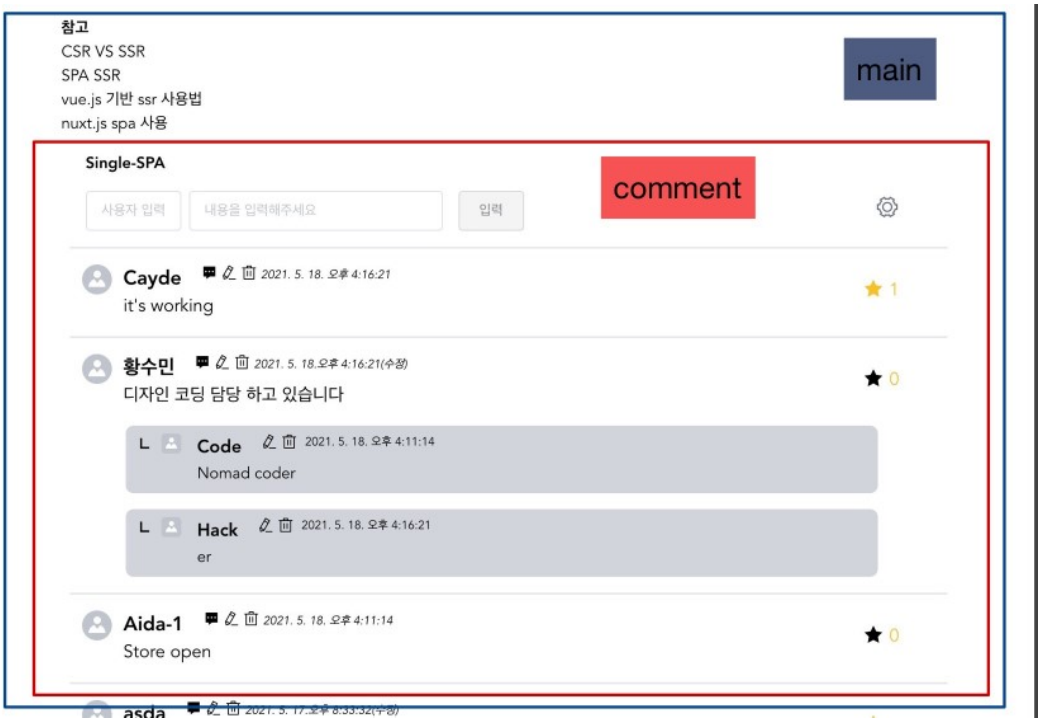



**그림 6. comment 컴포넌트 적용 결과 하면**

그림 6 의 화면은 별도의 마크다운이 나타나는 main 의 하단 부분에서 comment 컴포넌트가 Micro-frontend 아키텍쳐 형태로 나타난 것이다. 하나의 화면에서 동작하며, 입력, 수정, 삭제, 대댓글 추가 등의 기능이 정상적으로 동작하지만, 각각 다른 포트에서 동작한다.

그림 5 를 참고할 때 각각의 컴포넌트는 별도로 제작되었고, webpack 을 통해 연결된 상태이다. 기존의 comment 컴포넌트를 3001 번 포트에서 동작시킬 시에 정상적으로 댓글 등록과 삭제, 수정, 대댓글 추가 기능이 동작하며, 저장된 데이터가 db.json 으로 저장되는 것을 확인할 수 있다. 이를 3001 번 포트에서 main 에서 사용할 때도 마찬가지로, 이전과 동일한 기능을 사용 가능하며, tailwindcss 를 통해 comment 컴포넌트의 프론트엔드 사이즈를 main 에서 조절 가능하다.

현재는 데이터가 comment 컴포넌트에서 사용되고 있기 때문에 항상 동일한 데이터를 가져오지만, 코드를 재사용할 때, main 마다 다른 데이터를 가지고 있다면, 이를 통해 다양한 곳에서 댓글 기능을 수행하는 컴포넌트의 코드를 재사용 가능하다. 이는 Micro-frontend 아키텍쳐의 장점이기도 하며, 이러한 특징을 통해 개발자들의 독립적인 개발이 보장된다. 예를 들어, 댓글 컴포넌트와 게시판 컴포넌트, 검색 컴포넌트 등 다수의 기능을 각각 별도로 개발하여 main 에서 원하는 위치에 필요한 기능을 추가할 수 있다. 이러한

comment 컴포넌트와 유사한 기능을 수행하는 기존의 서비스로는 DISQUS 가 존재하는데, 이는 본 프로젝트에서 개발한 comment 컴포넌트와 달리, 필요애 따른 커스터마이즈가 불가능하며, 데이터를 DISQUS 자체에서 관리하기 때문에 보안 이슈가 발생할 수 있다. 그러나 webpack 5 module federation 으로 개발된 컴포넌트는 원하는 곳에 보관된 데이터를 가져와 사용하는 방식으로, 데이터를 직접 관리할 수 있다.

## 4. 결론 및 토론

기존의 monolith 아키텍처의 단점을 보완하기 위해 등장한 Micro-frontend 아키텍처를 구현하기 위해 webpack module federation 을 이용하여 댓글 컴포넌트를 구현하고, 실행해보았다. 그 결과, webpack 을 통해 연결한 서로 다른 서버에서 동작하는 컴포넌트가 필요에 따라 main 에서 본래의 기능을 정상적으로 수행하며 동작하는 것을 확인할 수 있었다. 현재 구현된 댓글 컴포넌트는 별도의 회원가입 서비스와는 연결되지 않은 상태이기 때문에 이용자의 제한이 없이 누구나 접근할 수 있는 상태이다. 따라서 추후에 등록된 유저들만 댓글 작성이 가능하도록 기능을 추가할 필요가 있다. 또한 코드의 재사용이 용이할 수 있도록 데이터를 가져오는 방식을 변경해야 한다.

module-federation 은 비교적 최근(2020 년 10 월 10 일)에 webpack 5 가 릴리즈 되면서 추가된 기능이기 때문에 아직까지 다양한 시도가 나타나지는 않았다. 그러나 사용 방법이 single-SPA 나 vue-genesis 등 Micro-frontend 아키텍처를 구현할 수 있는 다른 프레임워크들과 달리 상대적으로 이용이 간편하고, module-federation 이외에도 다양한 기능을 활용할 수 있다는 장점이 있기에 활용 가치가 높다고 판단한다.

**Acknowledgments**: 논문 지도를 해주신 김낙현 교수님, 함께 프로젝트를 진행해주신 애버커스 장동현 멘토님 그리고 박세진 멘토님께 감사의 말씀을 전합니다.

## 5. 참고문헌 - References

[1]Rany-Elhousieny,Micro-Frontnds:What,Why,andHow,2021.3.21,https://www.linkedin.com/pulse/micro-frontends-what-why-how-rany-elhousieny-phd%E1%B4%AC%E1%B4%AE%E1%B4%B0

[2]Webpack, Module-Federation 원문, 2020.10.10, https://webpack.js.org/concepts/module-federation/

[3]황수민•정주현,Module-Federation 번역,2021.5.27,https://github.com/Lajancia/webpack/blob/main/module-federation.md

[4]Webpack,Vue3-Webpack-Module-federation 코드,2021.5.27,https://github.com/Lajancia/webpack_vue3_main_sub

[5]Webpack,Webpack-Module-federation-댓글컴포넌트구현,2021.5.21,https://github.com/Lajancia/webpack_vue3_comment

[6]Element,element-UI, 2020.5.19,https://madewithvuejs.com/element-ui

[7]Element-Plus,element-plus-starter,2020.10.26,https://github.com/element-plus/element-plus-starter