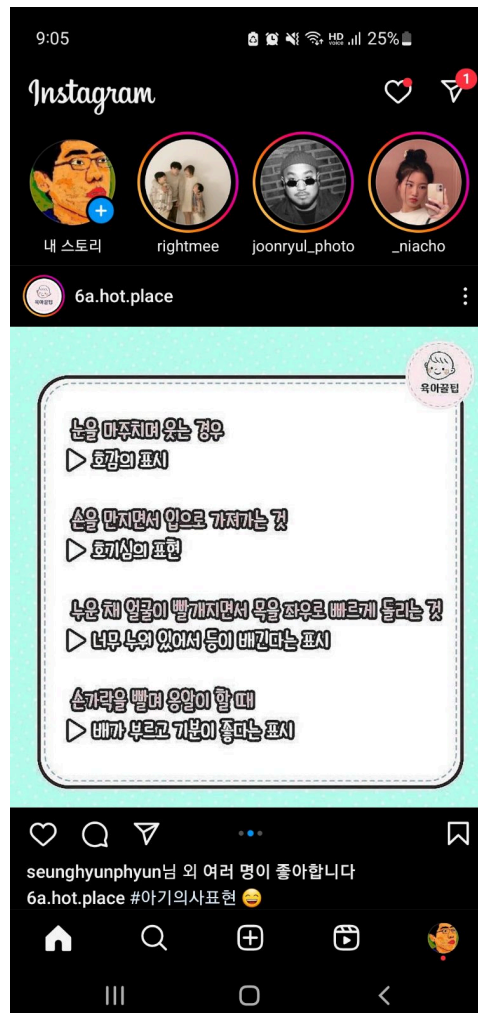


# Day4 Components

📎 자료	<u>Vue</u>
☰ 구분	Vue
☰ 과목	

## Components

컴포넌트는 Vue.js, React.js 등의 프론트엔드 프레임워크를 사용하는 큰 이유 중에 하나라고 생각한다.



위 사진은 인스타그램 웹 페이지 이다.

우리가 프레임워크 없이 HTML, CSS, JavaScript 만으로 웹 서비스를 개발을 한다고 가정해 보자.

처음에는 간단한 페이지로 시작하겠지만, 기능이 많아지고 디자인도 복잡해 질수록 HTML 한 파일 안에 수천, 수만 줄의 코드가 들어가게 될 것이다.

그렇게 되면 유지 보수도 어렵고, 협업도 힘들어 진다.

예를들어 두 가지 상황을 한번 생각해 보자.

첫번째 상황.

인스타그램 팀에서, 오른쪽 위의 하트 아이콘을 별모양 아이콘으로 바꾼다고 가정하자.

1. `index.html` 파일을 연다.
2. 몇천줄의 코드 안에서 하트를 찾는다.
3. 다른 아이콘으로 교체한다.

겉으로 보면 별 문제 없어 보이지만, 하트 하나 교체하려고 몇천 줄의 소스코드를 뒤적여야 한다는 것이 말이되나? 그리고 이 과연 안전한 개발 방식일까?

두번째 상황.

내가 인스타그램 팀에 입사한 신입이라고 가정하자.

디자이너가 나에게 `<footer>` 태그에 들어갈 새 아이콘을 줬고

다른 선배들은 메인 Feed영역을 작업 중이다.

1. 나는 `index.html` 파일에서 `<footer>` 를 작업한다.
2. 그런데 선배들도 같은 파일에서 Feed영역을 수정하고 서버에 먼저 업로드했다.
3. 그러나, `<footer>` 를 작업하는 나는 그걸 모른 채 수정한 푸터를 서버에 올린다.
4. 결과적으로 `<footer>` 는 반영됐지만, Feed쪽 수정은 날아가버린다.

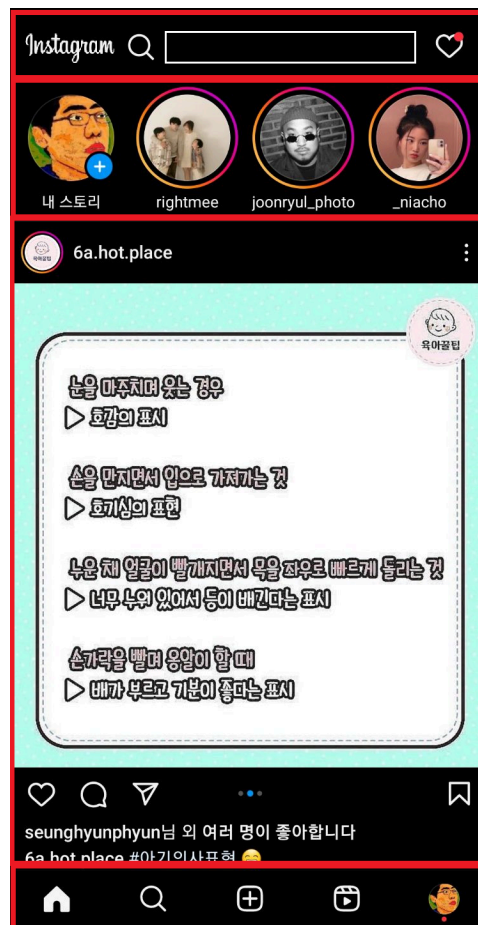
이러한 문제를 방지하기 위해, 수정한 코드만 따로 전달 하던가, 또는 Git 사용해 버전관리를 하는 등

하나의 HTML 파일을 여러명에서 동시에 수정하는데 다양한 방법이 연구되었다.

그러다가, 어떤 똑똑한 사람이 이렇게 생각한 것이다.

하나의 페이지를 여러 개의 파일로 나눠버리면 되지 않을까?

그게 바로 컴포넌트다. 레고를 생각하면 쉬운데, 각자 부품을 만들고, 싹 다 모아서 조립한다.

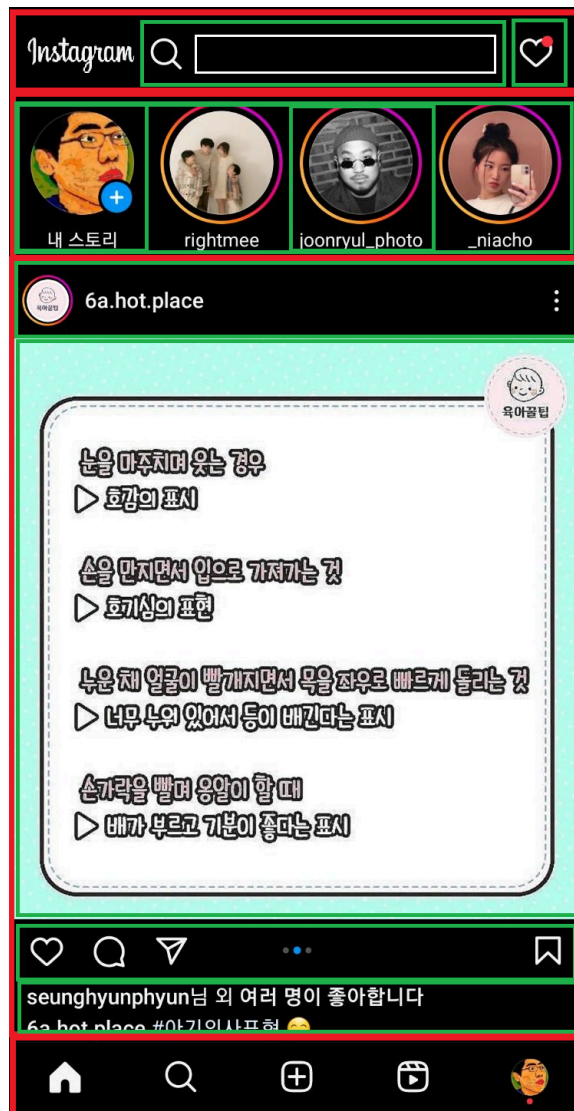


방금 전의 화면은 다음과 같이, 총 네 개의 컴포넌트로 나눌 수 있다.

각각의 컴포넌트는 각각의 `.vue` 파일이 되는데,

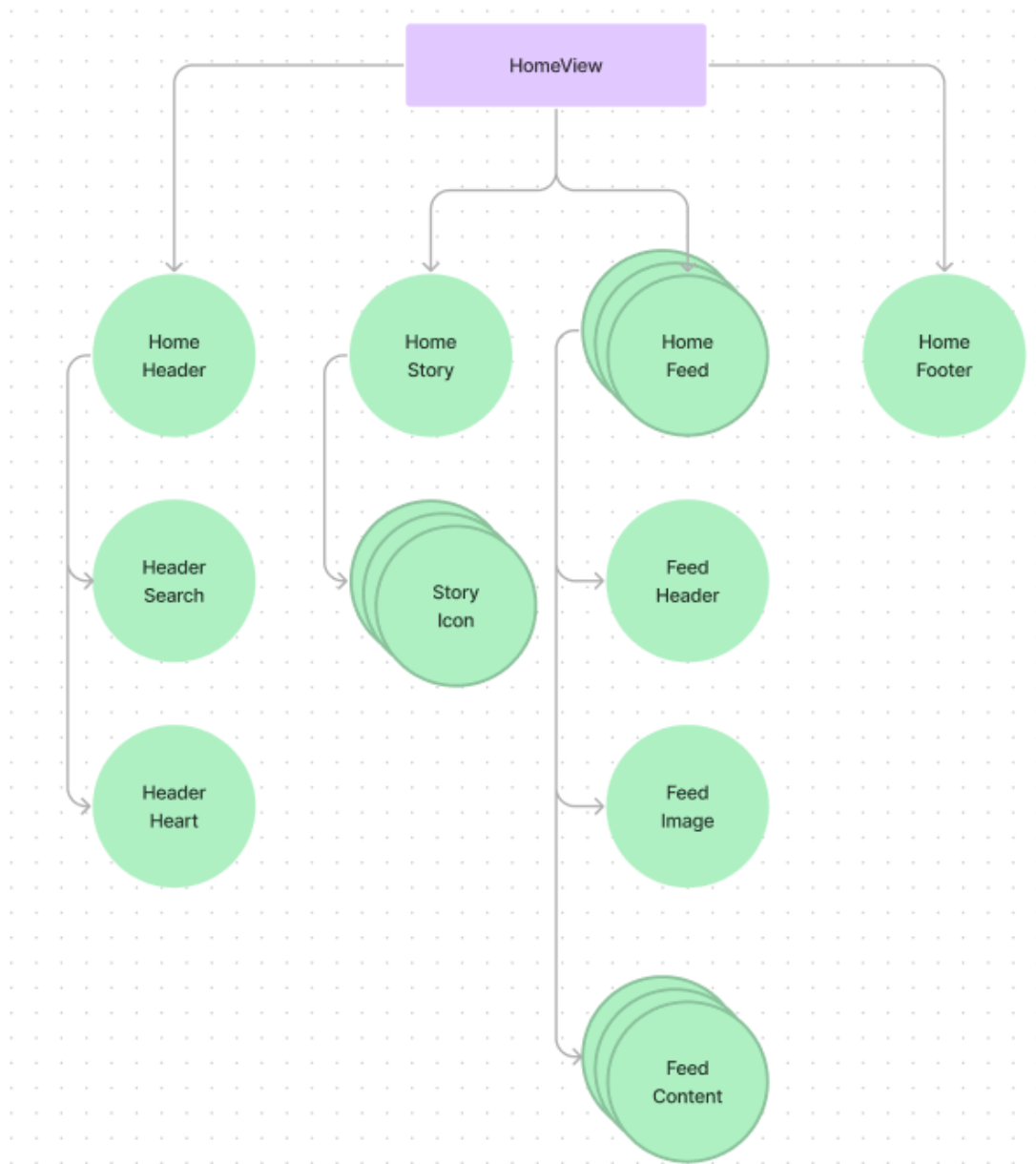
예를들자면 각각 `AppHeader.vue` , `AppStory.vue` , `AppFeed.vue` , `AppFooter.vue` 로 이름 짓겠다.

그리고 각각의 컴포넌트는 필요에 따라서 각각의 자식 컴포넌트를 둘 수도 있다.



(빨강 - 부모컴포넌트 // 초록 - 자식컴포넌트)

즉, 다음과 같은 트리 형태를 띠게 된다.



각각의 트리 요소는 하나의 파일, 즉 하나의 컴포넌트다.

- 위 트리를 자세하게 살펴 보면, `HomeFeed` 나 `StoryIcon` 처럼 여러개가 겹쳐져 있는 컴포넌트가 보이는데, 이는 `v-for` 를 통한 반복을 의미한다.
- 컴포넌트의 이름은 루트 컴포넌트 `App.vue` 를 제외하고 다음의 원칙에 따라 짓는 것을 추천한다.
  - 두 개의 명사를 결합해서 컴포넌트 이름을 만든다.
  - 대문자로 시작한다.
  - 두 개의 명사를 생각하기 어렵다면,  
첫번째 명사는 부모를 나타낼 수 있도록 할 것을 추천한다.

ex) StoryIcon ⇒ 부모는 HomeStory 컴포넌트

이런 방식으로 하나의 웹페이지를 컴포넌트 방식으로 따를 경우,  
처음 언급했던 두 가지 문제가 해결이 될 것이다.

첫번째 상황.

인스타그램 팀에서, 오른쪽 위의 하트 아이콘을 별모양 아이콘으로 바꾼다고 가정하자.

1. HeaderHeart.vue 파일을 연다.
2. 하트를 바꾼다.
3. 다른 컴포넌트는 건드리지 않는다.

두번째 상황.

당신은 인스타그램 팀에 입사한 신입이다. 디자이너는 당신에게 <footer> 태그에 들어갈 새  
아이콘 리스트를 줬고, 다른 선배 개발자들은 메인 Feed영역을 작업 중이다.

1. 한쪽에서는 HomeFooter.vue 를 작업한다.
2. 다른 한쪽에서는 HomeFeed.vue 를 작업한다.
3. 별개의 파일을 작업하므로 안전하다. 각각 작업한 두 개의 컴포넌트 파일을 변경하고 업로드한다.

즉, 컴포넌트 방식으로 개발할 때, 유지보수와 협업이 유리해진다.

자 이제 실습 해보자. 직접 자식 컴포넌트를 만들어보자.

먼저, 원하는 디렉터리를 vscode로 열어보자. 그리고 다음과 같이 입력해 프로젝트를 생성한다.

```
$ npm create vue@latest
```

Need to install the following packages:

create-vue@3.16.4

Ok to proceed? (y) y

## Vue.js - The Progressive JavaScript Framework

// Yes 선택 (스페이스바)

- | ■ Pinia (state management)
- | ■ Router (SPA development)
- | ■ Prettier (code formatting)
- | ■ ESLint (error prevention)

// No 선택

◆ Install Oxlint for faster linting? (experimental)

| ○ Yes / ● No

Scaffolding project in C:\Users\SSAFY\Desktop\vue-project...

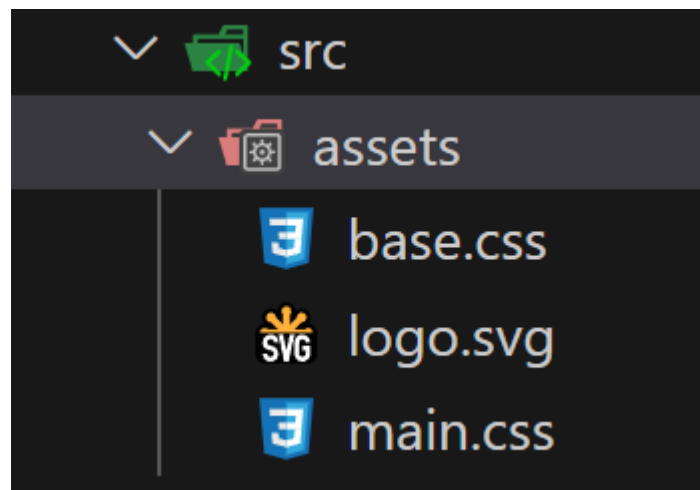
```
$ cd vue-project/
```

```
$ npm install
```

```
$ npm run dev
```

브라우저를 켜고 <http://localhost:5173> 으로 접속해서 빌드가 잘 되는지 확인하자.

그다음, `src/` 에서 필요없는 파일들을 삭제하겠다.

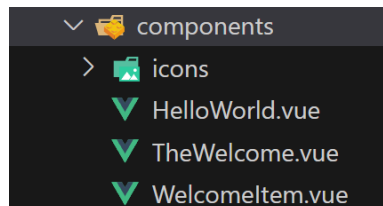


`src/assets/` 디렉터리부터 정리하겠다.

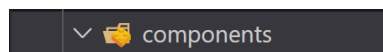
우선 `base.css` 와 `logo.svg` 를 삭제하고, `main.css` 의 모든 내용을 지운다.

다음, `src/components/` 하위 모든 파일, 디렉터를 삭제한다.

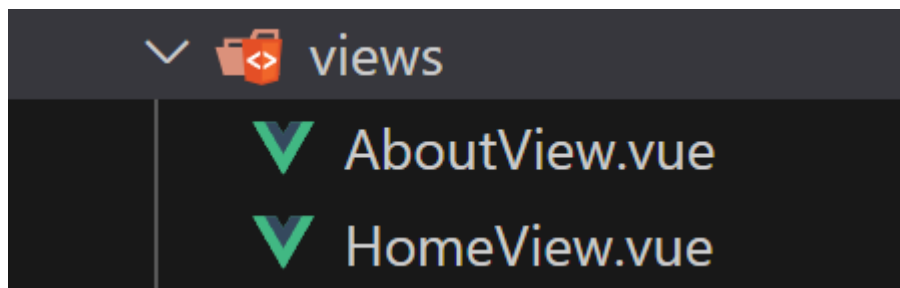
[삭제전]



[삭제후]



다음, `src/views/` 아래의 `HomeView.vue` 와 `AboutView.vue` 를 다음과 같이 변경한다




```
<template>
  <h1>About</h1>
</template>
```

```
<template>
  <h1>Home</h1>
</template>
```



마지막으로, `src/App.vue` 를 다음과 같이 변경한다.

The logo for App.vue, featuring a green stylized 'V' icon followed by the text 'App.vue' in a light blue font.

```
<script setup>
import { RouterLink, RouterView } from 'vue-router'
</script>

<template>
  <header>
    <nav>
      <RouterLink to="/">Home</RouterLink> |
      <RouterLink to="/about">About</RouterLink>
    </nav>
  </header>
  <RouterView />
</template>
```

여기까지 완료한 후, 서버를 동작시킨다.

```
$ npm run dev
```

`localhost:5173` 로 접속해보자.

A screenshot of a web application. At the top, there is a navigation bar with two links: 'Home' and 'About', both underlined. Below the navigation bar, the word 'Home' is displayed in a large, bold, black font.

`Home` , `About` 을 누를 때마다 화면이 바뀌는 것을 확인할 수 있다.

먼저 `HomeView.vue` 에 대해서 살펴보자.

```
<template>
  <h1>Home</h1>
</template>
```

- `HomeView.vue` 처럼, 뒤에 `.vue` 확장자가 붙어 있으면 하나의 컴포넌트로 취급한다.  
컴포넌트를 지칭할 때는 뒤에 확장자를 생략하고 `HomeView` 라고 부르곤 한다.
- 특히 `HomeView` 처럼 `views/` 폴더 안에 위치하면서, `-View` 라고 네이밍된 컴포넌트는 **라우트 컴포넌트**라고 부른다.  
라우트 컴포넌트에 대한 설명은 라우터를 학습할 때 자세하게 살펴 볼 것이다.

자 본론으로 들어가서 인스타 그램 예시처럼 자식 컴포넌트를 만드는 것을 실습 해보자.

## 자식 컴포넌트 만들기 1단계: 파일을 만든다.

`HomeView` 컴포넌트는 총 세 개의 자식 컴포넌트를 가질 것이다.

`src/` 폴더(디렉터리) 안에

`components/` 디렉터리에 `HomeHeader.vue` , `HomeMain.vue` , `HomeFooter.vue` 를 아래 양식으로 만든다.

그리고 `h2` 태그 안에는 현재 컴포넌트 이름을 각각 작성하자.

```
<template>
  <h2>HomeHeader</h2>
</template>
```

```
<template>
  <h2>HomeMain</h2>
</template>
```

```
<template>
  <h2>HomeFooter</h2>
</template>
```

## 자식 컴포넌트 만들기 2단계: 자식을 **import** 한다.

라우터 컴포넌트 **HomeView**의 최 상단에, **<script setup>** 태그를 생성한 후 다음과 같이 작성한다.

```
<script setup>
import HomeHeader from "@/components/HomeHeader.vue";
import HomeMain from "@/components/HomeMain.vue";
import HomeFooter from "@/components/HomeFooter.vue";
</script>

<template>
  <h1>Home</h1>
</template>
```

- **@** 는 **src/** 디렉터리를 의미하는 약어다.

## 자식 컴포넌트 만들기 3단계: 부모에 자식 컴포넌트를 붙인다.

현재 **HomeView** 컴포넌트의 **template** 부분에 자식 컴포넌트를 붙이자.

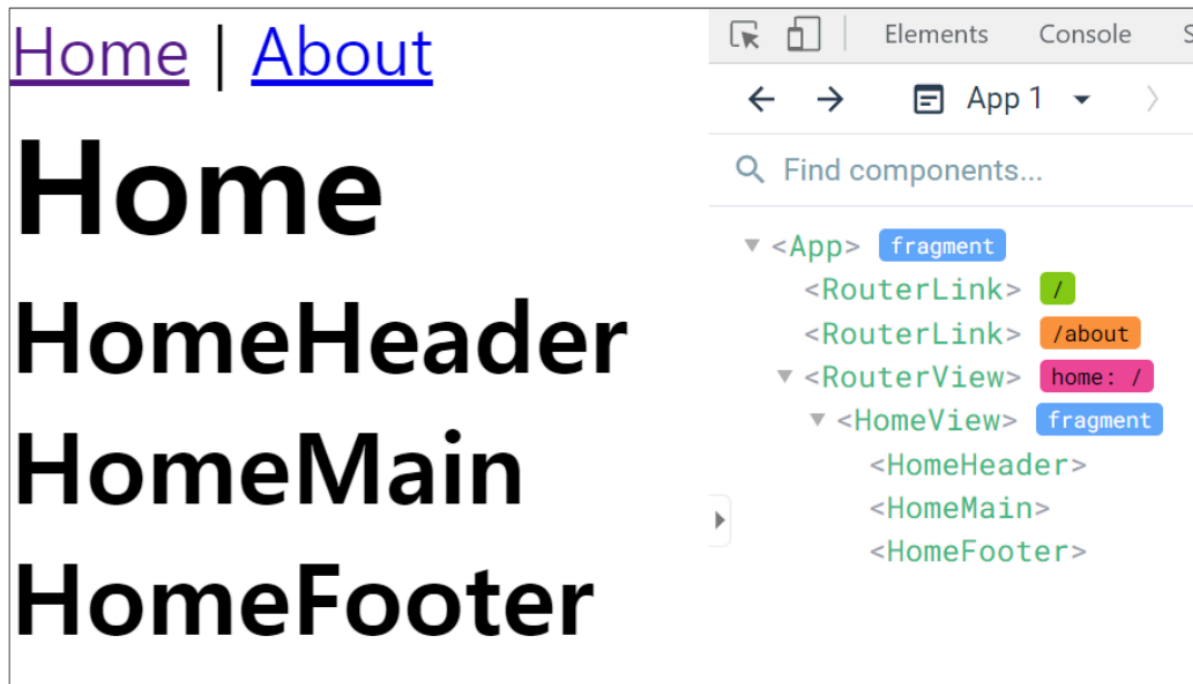
```
<script setup>
import HomeHeader from "@/components/HomeHeader.vue";
import HomeMain from "@/components/HomeMain.vue";
import HomeFooter from "@/components/HomeFooter.vue";
</script>

<template>
  <h1>Home</h1>

  <HomeHeader />
  <HomeMain />
  <HomeFooter />

</template>
```

컴포넌트를 붙일 땐, `<template>` 에서 일반 태그와 구분 되도록 PascalCase 를 사용하며, 생성과 함께 닫아준다.



Vue 개발자 도구를 열어서 확인해보면, 부모 자식 관계가 잘 설정된 것을 확인할 수 있다.

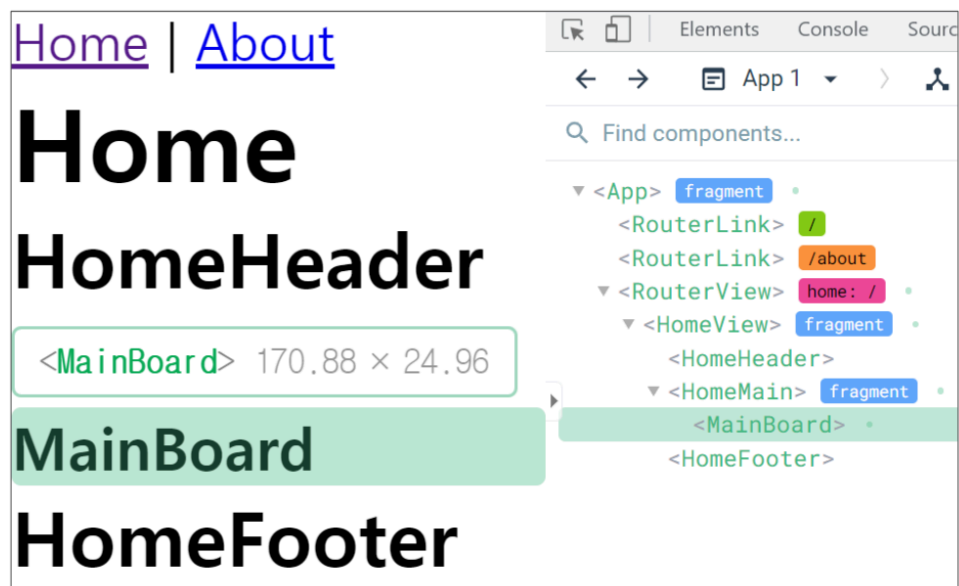
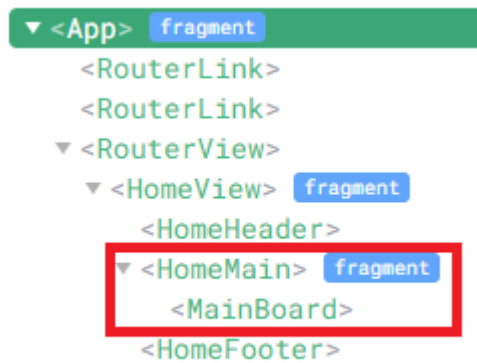
#### [도전과제]

도전: `HomeMain` 컴포넌트에, `MainBoard` 라는 자식 컴포넌트를 만들어 아래 그림과 같이 나오도록 해보자.

```
<template>
  <h3>MainBoard</h3>
</template>
```

( `HomeView` 컴포넌트에 자식을 만드는 것이 아니다!! `HomeMain`의 자식 이다. )

( `h3` 태그로 만들었음. )



위의 도전 과제를 직접 완성 해야지 다음 진도의 실행 결과를 확인 할 수 있으니, 반드시 직접 만들어 보자.

## scoped

scoped 에 대해서 알아보자.

하나의 컴포넌트는 기본적으로 다음과 같은 구조를 지닌다.

```

<template>
  <!-- html code -->
</template>

```

```

<script setup>
// js code
</script>

<style scoped>
/* css code */
</style>

```

이 기본적인 구조는 반드시 알아둬야 한다.

`<template>` : HTML

`<script setup>` : JavaScript

`<style scoped>` : CSS

즉, 하나의 컴포넌트에서 별도의 파일 분리를 하지 않고, HTML, CSS, JavaScript 를 같이 사용한다.

그런데, `<style>` 태그 안에 `scoped` 속성은 대체 무슨 뜻일까?

`scoped` 가 없다면 전역 스타일링,

`scoped` 가 있다면 지역 스타일링을 말하며 일반적으로 컴포넌트에서 붙여서 사용한다.

차이를 직접 확인 해 보자.

먼저, `HomeView` 라우터 컴포넌트에서 `<style>` 태그를 `scoped` 없이 작성하겠다.

아래 코드를 참고하자.

```

<script setup>
import HomeHeader from "@/components/HomeHeader.vue";
import HomeMain from "@/components/HomeMain.vue";
import HomeFooter from "@/components/HomeFooter.vue";
</script>

<template>
  <h1>Home</h1>
  <HomeHeader />

```

```

<HomeMain />
<HomeFooter />
</template>

<style>
h2 {
  color: red;
}
</style>

```

무슨 뜻인가? 모든 `<h2>` 태그의 글자 색은 빨간색으로 하라는 뜻이다.

그런데 자세히 보면, `HomeView` 컴포넌트엔 `<h2>` 태그가 존재하지 않는다! 그러나, 결과는 다음과 같다.

[Home](#) | [About](#)

# Home

## HomeHeader

## HomeMain

## MainBoard

## HomeFooter



자세히 보면, `<h2>` 태그를 가진 자식 컴포넌트의 모든 색깔은 빨간색으로 처리 되었다.

왜 이런 현상이 생기는 것일까? 소스파일에 코드가 수천 줄이 있더라도

Vue.js 프로젝트는 결국 하나의 HTML 문서로 ( `index.html` )로 이루어져 있다.

이를 SPA, Single Page Application 이라고 하는데, 페이지가 단 하나라는 뜻이다.

이는 어떠한 컴포넌트 파일이든 상관 없이, 모든 컴포넌트 파일의 태그에 접근해서 스타일링 할 수 있다는 것을 뜻한다. 현재 `<h2>` 태그는 각각 `HomeHeader` , `HomeMain` , `HomeFooter` 컴포넌트에 있으므로, 해당 태그는 모두 빨간색이 된다.

그렇다면, style 태그에 `scoped` attribute를(속성) 추가하면 어떻게 될까?

이번엔 `HomeMain` 에서 `<style>` 태그를 만들되, `scoped` 를 붙여서 다음과 같이 만들어주자.

```
<script setup>
import MainBoard from "@/components/MainBoard.vue";
</script>

<template>
  <h2>HomeMain</h2>
  <MainBoard />
</template>

<style scoped>
```



```
h2 {  
  color: blue;  
}  
</style>
```



`scoped` 를 붙이면, 스타일의 영역을 해당 컴포넌트로 한정 짓는다. 다른 컴포넌트에 `<h2>` 가 있든 말든, 오로지 현재 컴포넌트의 `<h2>` 에만 스타일링을 적용한다.

`scoped` 를 붙이지 않으면 모든 `<h2>` 파란색으로 바뀌는 것을 확인 할 수 있을 것이다.

그럼 둘은 언제 어떻게 사용 하는 것이 좋을까? 일반적인 사용법은 다음과 같다.

`<style>` : 루트 컴포넌트( `App.vue` )에서 전역 스타일링 할 때 사용

`<style scoped>` : 일반 컴포넌트에서 사용

ex) 만약 웹 서비스의 모든 폰트를 Noto Sans KR 로 바꾸고 싶다면

`App.vue` 에서 `scoped` 없이 적용 해야 할 것이다.

- Vite 에서 기본적으로 만들어주는 템플릿을 자세히 보면, `App.vue` 에서도 `scoped` 를 붙여 지역 스타일링을 하는 것을 볼 수 있으며, 전역 스타일링을 할 일이 있다면 `src/assets/main.css` 에서 작업하는 방식을 취하는 것을 짐작 할 수 있다.

- 따라서 차후에는 이 방식을 따라, 모든 컴포넌트에 `scoped` 를 붙여 지역 스타일링하고, 전역 스타일링할 일이 있다면 `main.css` 에서 작업하도록 하겠다.
- 다만 지금은 `scoped` 가 무슨 뜻인지 예시를 들어 설명을 한번 했을 뿐이다.

<끝>