# Day6

| | |
|---|---|
| 자료 | Vue |
| 구분 | Vue |

오늘의 핵심 주제는 router 그리고 navigation guard 입니다.

먼저 router에 대한 이해와 사용법이 익숙해야지

navigation guard  내용을 받아 들이기 쉬울 것입니다.

따라서 오늘 교안은  router 에 대한 이해를 위해서 작성하였습니다.

## router 1

먼저 알아야 할 것은 Vue.js 는 기본적으로 하나의 HTML, 즉 `index.html` 파일만 가진다는 것이다. 즉, 기존의 다른 웹개발 방식에서 보이는 `login.html` , `signup.html` , `board.html` 등의 여러 페이지가 존재하지 않기 때문에, 다른 페이지로 이동할 수 없다. 즉, 화면은 본질적으로 하나의 HTML 파일이다.

그러나, 여러가지 화면을 가진, 그리고 각 화면마다 URL 을 가진 웹 페이지는 우리에게 너무나 익숙한 개념이다. 하나의 HTML 만을 가진 Vue.js 에선 이 기능을 직접 제공하지 않고, 서드파티 패키지인  Vue Router 를 사용하도록 권장하며, Vite 프로젝트 생성 시 Vue Rouer 를 사용할지 선택할 수 있다.

```
$ npm create vue@latest

Yes / No 선택시 router 에 관련 설정만 Yes 로 선택하자
```

`src/` 에서 필요없는 파일들을 삭제하겠다.

`src/assets/` 디렉터리부터 정리하겠다.

우선 `base.css` 와 `logo.svg` 를 삭제하고, `main.css` 의 모든 내용을 지운 후, 다음과 같이 입력한다.

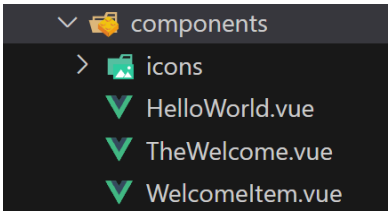
```
* {
  box-sizing: border-box;
  margin: 0;
}
```

`main.css` 는 전역 스타일링 파일이다.

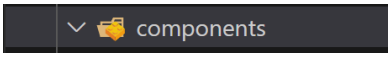
- `box-sizing` 은 박스의 크기를 화면에 표시하는 방식을 변경하는 속성이다. 테두리가 있는 경우에는 테두리의 두께로 인해서 원하는 크기를 찾기가 어려운데, `box-sizing` 속성을 `border-box` 로 지정하면 테두리를 포함한 크기를 지정할 수 있기 때문에 크기를 예측하기가 더 쉽다. 기본적으로 프로젝트를 하나 만들면, 모든 엘리먼트에 이 값을 지정하고 시작한다.
- `margin: 0` 은 각 태그에 기본적으로 잡혀있는 공백을 제거해준다.

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

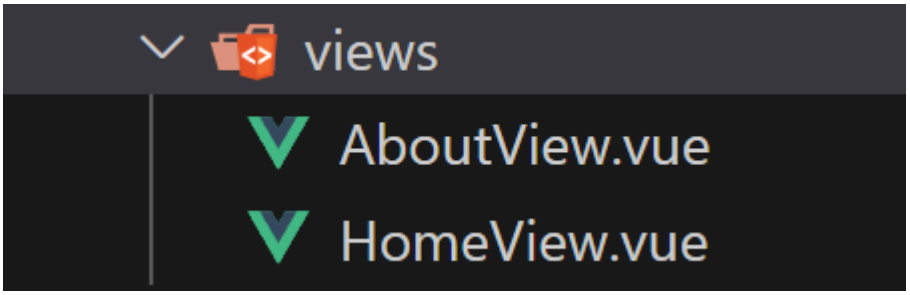
[삭제전]




[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.


```
<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.


About 링크를 클릭하면 화면이 바뀌면서,

URL 도 `http://localhost:5173/` 에서, `http://localhost:5173/about` 으로 바뀌는 것을 확인 할 수 있을 것이다. 그리고 눈여겨 봐야 할 부분은, 페이지가 바뀌어도 새로고침이 없다.

즉, SPA 의 특성은 계속 유지하면서, URL 로 화면 전환이 가능하도록 만든 것이 바로 Vue Router 다.

먼저, Vue Router 의 설정 파일인 `router/index.js` 로 향하자.

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    },
    {
      path: '/about',
      name: 'about',
      // route level code-splitting
      // this generates a separate chunk (About.[hash].js) for this route
      // which is lazy-loaded when the route is visited.
      component: () => import('../views/AboutView.vue')
    }
  ]
})

export default router
```

`import` 구문을 자세히 보면, `views/` 디렉터리의 컴포넌트를 가져온다는 것을 알 수 있다.

우리 수업에서는 `routes` 배열만 다룰 예정이다. `routes`은 객체 배열로 이루어져 있는데, 각각의 객체의 프로퍼티가 의미하는 바를 알아보자.

- `path` : URL 을 의미한다. `/` 는 루트라는 뜻이며, `localhost:5173` 접속 시 기본적으로 보여줄 페이지
- `name` : 곧 살펴보게 될 `RouterLink` 에서 보낸 `name` 의 이름에 해당함

- `component` : route 에 해당하는 컴포넌트

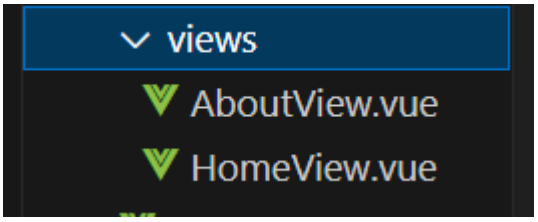
용어 정리를 잠시 하고 가자.

router : route 를 관리하는 객체

route : 각각의 화면

그리고 각각의 route 에 연결되어 있는 컴포넌트를 앞으로 "라우트 컴포넌트" 라고 부를 것이며, 각 화면을 대표하기에 각 화면의 루트 컴포넌트가 된다.

라우트 컴포넌트는 각 화면의 루트가 되는 매우 특별한 컴포넌트이기에 `components/` 디렉터리가 아니라 `views/` 디렉터리에서 따로 관리할 것이다. 또한 네이밍 시 맨 뒤에 `View` 를 붙여준다.




- 위 코드에서 두가지의 객체 사용방식을 보여주는데, `component` 부분이 다르다.


공식문서에서는  Lazy Loading Routes 라고 표현하며, 동적 import 로 구현 가능하다. 이 방식을 사용할 경우, 페이지 로딩 시간을 줄이는데 도움이 된다는 이유로 Vue Router 에선 모든 컴포넌트에 동적 import 를 사용할 것을 권장한다.

https://router.vuejs.kr/guide/advanced/lazy-loading.html

따라서, 우리도 동적 import 방식으로 통일하여 작성하도록 하겠다. 헷갈리지 않도록 다음과 같이 수정하도록 하자.

```
import { createRouter, createWebHistory } from 'vue-router'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: () => import('../views/HomeView.vue')
    },
    {
      path: '/about',
      name: 'about',
      component: () => import('../views/AboutView.vue')
    }
  ]
})

export default router
```

분석을 위해 `App.vue` 로 향하자.

```
<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>
```

`<RouterLink>` , `<RouterView>` 라는 태그 (사실은 컴포넌트)를 확인할 수 있다.

`<RouterLink :to="/경로">`

- 생김새와 사용법은 `<a href="경로">` 와 같아 보인다.
- 유추를 해면, `Home` 링크를 클릭하면 해당 경로로 향하는 것을 알 수 있다.

`<RouterView />`

- `<RouterView />` 부분은 사용자가 선택한 화면 (Home | About) 이 보여지는 영역이다.

사실 위 코드보다 아래 코드, path 가 아니라 name 을 호출을 할 수도 있다.

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

<template>
  <header>
    <nav>
      <!-- <RouterLink to="/">Home</RouterLink> |
      <RouterLink to="/about">About</RouterLink> -->
      <RouterLink :to="{ name: 'home' }">Home</RouterLink> |
      <RouterLink :to="{ name: 'about' }">About</RouterLink>
    </nav>
  </header>
  <RouterView />
</template>
```

`<RouterLink :to="{ name: '이름' }">` 으로 사용 하는 것을 알 수 있다. 역시 생긴 것과 사용법은 a 태그 `<a href="경로">` 와 비슷하다.

단, `name` 사용 시엔 반드시 `v-bind:` 를 사용해야 한다. 즉, `to` 가 아니라 `:to` 이며, `RouterLink` 컴포넌트에 `props` 로 객체를 내려보낸다.

즉, 사용자가 링크를 클릭하면, 각각의 `name` 에 해당하는 컴포넌트가 `<RouterView />` 영역에 보여지는 것을 유추할 수 있다. 만약 사용자가 링크를 클릭하지 않고 URL 로 직접 접속하면 `path` 에 해당하는 컴포넌트가 보여질 것이다.

Vue Router 는 공식적으로 `path` 보다 `name` 을 추천한다. 다음과 같은 장점 때문이다.

Vue Router

Vue 3에 필요한 최신 공식 라우터

https://router.vuejs.kr/guide/essentials/named-routes.html

- 하드코딩된 URL 없음
- `params`을 자동 인코딩/디코딩
- URL에 오타 발생 방지
- 경로의 우선순위를 우회

따라서, 앞으로 `name` 방식의 Vue Router 사용 방법으로 기술하겠다.

[도전] : `BoardView` 라우트 컴포넌트를 만들고, 링크로 접속할 수 있도록 해보자. (직접 해보기)

그러나, 화면 전환은 링크로만 이루어 지는 것이 아니다.

만약 어떤 사람이 버튼을 눌렀을 때 다른 화면으로 가고 싶다면 어떻게 하는 게 좋을까?

`HomeView` 를 다음과 같이 작성하자.

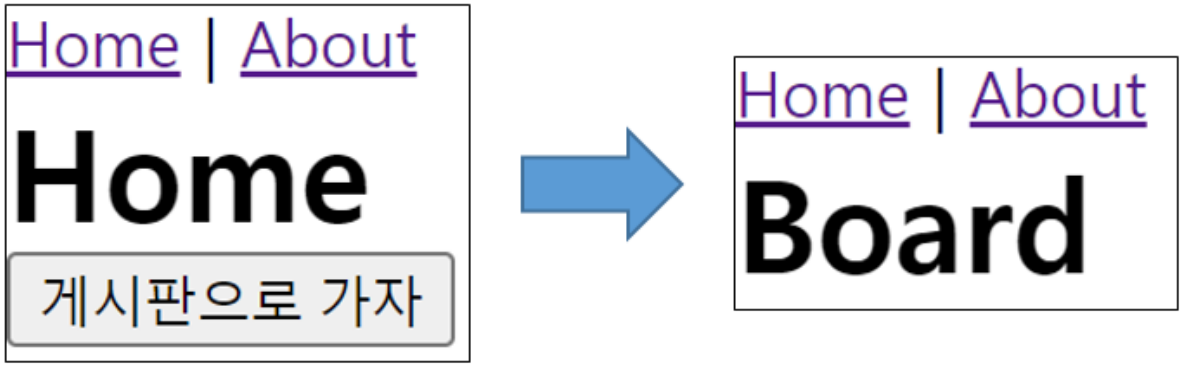
```
<script setup>
import { useRouter } from 'vue-router'

const router = useRouter();

function goToBoardRoute() {
  router.push({ name: "board" });
}
</script>

<template>
  <h1>Home</h1>
  <button @click="goToBoardRoute">게시판으로 가자</button>
</template>
```

`vue-router` 에서 제공하는 `useRouter` 컴포저블의 리턴으로 `router` 객체를 받는다.

```
[참고] 컴포저블? composable?
https://v3-docs.vuejs-korea.org/guide/reusability/composables.html
```

```
링크 들어가서 스크롤 내리면 “비동기상태 예제”가 있는데
궁금하면 참고사항으로 눈으로 한번 읽어보면 된다.
```

`router.push` 를 사용하며, 아규먼트는 템플릿에서 사용할때와 같다.




Lifecycle_Hooks__Router (1).pdf

# router 2 - Params

만약 다음을 구현하고 싶다면 어떻게 하는 게 좋을까?

`board/1` : 1번글 상세 페이지

`board/2` : 2번글 상세 페이지

우선, 디테일 컴포넌트를 만든다. 이름은 `BoardDetailView` 라고 하겠다.

```
<template>
  <h1>BoardDetail</h1>
</template>
```

`router/index.js` 에 다음과 같이 추가로 작성한다.

```
{
  path: '/board/:id',
  name: "detail",
  component: () => import('../views/BoardDetailView.vue')
}
```

`path` 부분이 좀 특별해졌는데, 받고 싶은 params 를 `:변수` 식으로 정의했다.

`router/index.js` 현재 상태는 다음과 같을 것이다.

```
import { createRouter, createWebHistory } from 'vue-router'


const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: () => import('../views/HomeView.vue')
    },
    {
      path: '/about',
      name: 'about',
      // route level code-splitting
      // this generates a separate chunk (About.[hash].js) for this route
      // which is lazy-loaded when the route is visited.
      component: () => import('../views/AboutView.vue')
    },
    {
      path: '/board',
      name: 'board',
      component: () => import('../views/BoardView.vue')
    },
    {
      path: '/board/:id',
      name: "detail",
      component: () => import('../views/BoardDetailView.vue')
    }

  ]
})

export default router
```

이 상태로 URL 에 `/board/1` 식으로 입력해보자.


이건 우리가 원했던 결과가 아니다. 적어도 몇 번 페이지인지는 나와야 한다.

`BoardDetailView` 를 다음과 같이 수정하자.

```
<script setup>
import { useRoute } from "vue-router";

const route = useRoute();
</script>

<template>
  <h1>여기는 {{ route.params.id }}번 글의 상세페이지입니다.</h1>
</template>
```

이번에는 `useRouter` 가 아니라 `useRoute` 컴포저블을 사용하겠다. `route.params` 객체에 접근해, 우리가 `:id` 라고 이름붙인 변수를 리턴하는 것을 알 수 있다.

`/board/1` 과 `/board/2` 로 접속해 각각 결과를 확인하면 다음과 같다.



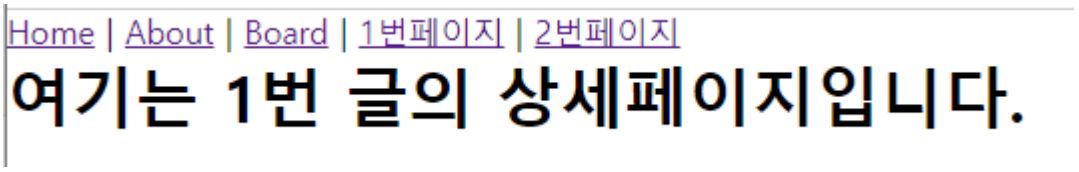
- `router` 와 `route` 는 다르다. `router` 는 `push` 사용할 때, `route` 는 `params` 사용할 때다.
- 게시판 목록에서 특정 게시글을 클릭하면 접속되는 상세 페이지를 구현할 때, 최초 접속 시 `route.params.id` 를 받아와 서버에 상세 페이지 JSON 데이터를 비동기 요청하는 경

우가 흔하다.

만약, 링크로 1번 페이지에 접속하려면 다음과 같이 작성하면 된다.

```
<RouterLink :to="{ name: 'detail', params: { id: '1' } }">1번페이지</RouterLink> |
<RouterLink :to="{ name: 'detail', params: { id: '2' } }">2번페이지</RouterLink>
```

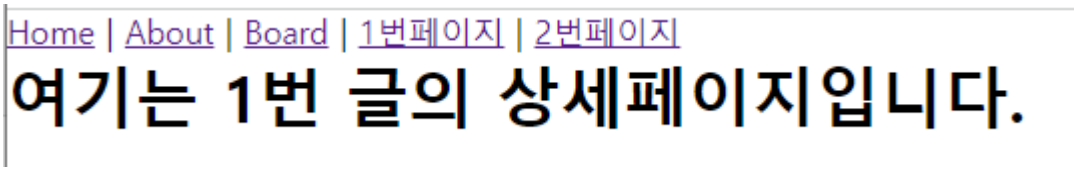

1번페이지 그리고 2번페이지 클릭시 이동이 잘 되는지 확인하자.




만약에 Board에서 버튼 하나 달고, 버튼을 클릭을 하여 1번 페이지에 접속하는 코드를 구현하려면 다음과 같이 작성하면 되겠다.

```
<!-- BoardView.vue -->

<script setup>
import { useRouter } from 'vue-router'

const router = useRouter();

function goToDetail(n) {
  router.push({ name: "detail", params:{id:n} });
}

</script>

<template>
```

```
  <h1>Board</h1>
  <button @click="goToDetail(1)">1번 게시글로 가자</button>
</template>
```


# Navigation Guard

- Vue router를 통해 특정 URL에 접근할 때 다른 url로 redirect를 하거나 해당 URL로의 접근을 막는 방법
- Ex) 사용자의 인증 정보가 없으면 특정 페이지에 접근하지 못하게 함

Navigation Guards | Vue Router

The official router for Vue.js.

https://v3.router.vuejs.org/guide/advanced/navigation-guards.html

## 네비게이션 가드의 종류

- 전역 가드
  - 애플리케이션 전역에서 동작
- 라우터 가드

    - 특정 URL에서만 동작
- 컴포넌트 가드
    - 라우터 컴포넌트 안에 정의

이제 라이브 따라 하면서 한번 익혀보시면 좋겠습니다 !!