Vue Router

**학습목표**

Vue router

**목차**

[1. router 프로젝트 2](#_Toc103761306)

[1) 개요 2](#_Toc103761307)

[2) router 프로젝트 생성 2](#_Toc103761308)

[3) 포트 번호 변경 2](#_Toc103761309)

[2. Vue Router 3](#_Toc103761310)

[1) router 설치 3](#_Toc103761311)

[2) 생성된 코드 4](#_Toc103761312)

[3) AboutView 컴포넌트 등록 수정 8](#_Toc103761313)

[3. TodoView 구현 9](#_Toc103761314)

[1) src/views/TodoView.vue 생성 9](#_Toc103761315)

[2) src/router/index.js 수정 9](#_Toc103761316)

[3) src/App.vue 수정 10](#_Toc103761317)

[4) 실행 11](#_Toc103761318)

[4. 연습문제 11](#_Toc103761319)

# router 프로젝트

## 개요

자바스크립트로 구현된 프론트엔드 앱은 보통 한 페이지에 모든 기능을 다 구현한다. (Single Page App)

한 페이지에서 다른 페이지로 넘어갈 때, 자바스크립트 앱은 종료된다.

페이지를 새로 고침할 때도, 자바스크립트 앱은 종료된 후 재시작한다.

여러 화면이 필요한 앱을 자바스크립트로 구현할 때도, 한 페이지에 모든 화면을 다 구현한다.

페이지 하나에 여러 화면을 구현하기 위해 Vue Router 라이브러리가 필요하다.

즉 여러 화면으로 구성된 Single Page App을 구현할 때, Vue Router 라이브러리가 필요하다.

router 라이브러리를 이용하여, 각각의 화면을 컴포넌트로 구현하면,

페이지가 넘어가지 않고도 화면을 전환할 수 있다.

즉 사용자가 보기에는 페이지가 넘어간 것 처럼 보이지만,

실제로 페이지가 넘어간 것은 아니고, 화면에 보이는 컴포넌트만 교체된다.

## router 프로젝트 생성

프로젝트를 생성할 적당한 폴더로 이동하자.

|  |
| --- |
| cd /c/PJ/frontend |

아래 명령을 실행해서 프로젝트를 생성하자.

|  |
| --- |
| vue create vue\_app3 |

## 포트 번호 변경

컴퓨터에 오라클 DBMS나 톰캣이 설치되어 있다면, 8080 포트가 이미 사용중일 수 있다.

vue 프로젝트가 사용할 포트 번호를 8082로 변경하는 방법은 다음과 같다.

### package.json

|  |
| --- |
| ... 생략 ...    "scripts": {      "serve": "vue-cli-service serve --port 8082",      "build": "vue-cli-service build",      "lint": "vue-cli-service lint"    },  ... 생략 ... |

package.json 파일을 열고 위의 내용을 찾아서,

노란색으로 칠한 부분을 추가하자.

### 프로젝트 폴더 열기

Visual Studio Code에서 프로젝트 폴더를 열자

# Vue Router

## router 설치

Vue 앱 프로젝트에 Vue Router 라이브러리를 설치하자.

|  |
| --- |
| vue add router |

프로젝트 폴더에서 위 명령을 실행하자.

|  |
| --- |
|  |

Y 입력하고 엔터

위의 명령으로 router 라이브러리를 프로젝트에 설치하면,

router 관련 소스코드도 자동으로 생성된다.

생성된 소스코드를 실행해보자.

|  |
| --- |
| npm run serve |

Graphical user interface, application

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Home 링크, About 링크를 클릭하면, 해당 화면으로 넘어간다.

웹 페이지가 넘어가는 것은 아니고, 화면을 차지하는 컴포넌트만 교체된다.

## 생성된 코드

### src/main.js

|  |
| --- |
| import { createApp } from 'vue'  import App from './App.vue'  import router from './router'  createApp(App).use(router).mount('#app') |

import router from './router'

router 객체를 import 한다.

createApp(App).use(router).mount('#app')

라우터 서비스 객체를 앱에 추가한다.

### src/App.vue

|  |
| --- |
| <template>    <nav>      <router-link to="/">Home</router-link> |      <router-link to="/about">About</router-link>    </nav>    <router-view/>  </template>  <style>  #app {    font-family: Avenir, Helvetica, Arial, sans-serif;    -webkit-font-smoothing: antialiased;    -moz-osx-font-smoothing: grayscale;    text-align: center;    color: #2c3e50;  }  nav {    padding: 30px;  }  nav a {    font-weight: bold;    color: #2c3e50;  }  nav a.router-link-exact-active {    color: #42b983;  }  </style> |

<router-link to="/">Home</router-link> |

<router-link to="/about">About</router-link>

화면을 전환하기 위한 링크를 출력한다.

Home 링크를 클릭하면 "/" 주소에 등록된 컴포넌트가 현재 화면에 출력된다.

About 링크를 클릭하면 "/about" 주소에 등록된 컴포넌트가 현재 화면에 출력된다.

즉 위 링크들 중의 하나를 클릭하면, 해당 컴포넌트 화면으로 넘어간다.

<router-view/>

현재 화면을 차지하는 컴포넌트가 이 부분에 출력된다.

### src/views/HomeView.vue

|  |
| --- |
| <template>    <div class="home">      <img alt="Vue logo" src="../assets/logo.png">      <HelloWorld msg="Welcome to Your Vue.js App"/>    </div>  </template>  <script>  // @ is an alias to /src  import HelloWorld from '@/components/HelloWorld.vue'  export default {    name: 'HomeView',    components: {      HelloWorld    }  }  </script> |

Home 화면을 구현한 컴포넌트이다.

'@/components/HelloWorld.vue'

여기서 @ 부분은 프로젝트 폴더 아래 src 폴더를 뜻한다

### src/views/AboutView.vue

|  |
| --- |
| <template>    <div class="about">      <h1>This is an about page</h1>    </div>  </template> |

About 화면을 구현한 컴포넌트이다.

### src/router/index.js

|  |
| --- |
| import { createRouter, createWebHistory } from 'vue-router'  import HomeView from '../views/HomeView.vue'  const 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(/\* webpackChunkName: "about" \*/ '../views/AboutView.vue')    }  ]  const router = createRouter({    history: createWebHistory(process.env.BASE\_URL),    routes  })  export default router |

router 객체를 생성하는 코드이다.

(여기서 생성된 router 객체는 main.js에 import 된다.)

### HomeView 컴포넌트 등록

import HomeView from '../views/HomeView.vue'

Home 화면을 구현한 HomeView 컴포넌트를 import 한다.

{

  path: '/',

  name: 'home',

 component: HomeView

}

'/' 주소에 HomeView 컴포넌트를 등록한다.

<router-link to="/"> 링크가 클릭되면, HomeView 컴포넌트가 <router-view/> 부분에 출력된다.

### AboutView 컴포넌트 등록

AboutView 컴포넌트 등록 코드는 HomeView 컴포넌트 등록 코드에 비해 복잡하다.

lazy loading 방식으로 등록했기 때문이다.

{

    path: '/about',

    name: 'about',

    component: () => import('../views/AboutView.vue')

}

'/about' 주소에 AboutView 컴포넌트를 등록한다.

<router-link to="/about"> 링크가 클릭되면, AboutView 컴포넌트가 <router-view/> 부분에 출력된다.

() => import('../views/AboutView.vue')

위와 같이 구현하면, '/about' 주소 링크가 처음으로 클릭된 경우에,

AboutView 컴포넌트를 생성한다. (lazy loading)

'../views/AboutView.vue'

이 소스코드가 들어있는 파일은 src/router 폴더에 들어있다. (src/router/index.js)

../views/AboutView.vue 경로는,

src/router 폴더를 기준으로 ../views 폴더에 있는 AboutView.vue 파일을 의미한다.

.. 부분은 부모 폴더를 의미한다.

따라서 src/router 폴더를 기준으로 ../views 폴더는 src/view 폴더이다.

즉 src/view/AboutView.vue 폴더를 의미한다.

## AboutView 컴포넌트 등록 수정

lazy loading 방식으로 등록하지 않을 거라면, 다음과 같이 간단하게 등록할 수 있다.

### src/router/index.js

|  |
| --- |
| import { createRouter, createWebHistory } from 'vue-router'  import HomeView from '../views/HomeView.vue'  import AboutView from '../views/AboutView.vue'  const routes = [    {      path: '/',      name: 'home',      component: HomeView    },    {      path: '/about',      name: 'about',      component: AboutView    }  ]  const router = createRouter({    history: createWebHistory(process.env.BASE\_URL),    routes  })  export default router |

lazy load 기능을 사용하지 않으면 위와 같이 간단하게 구현할 수 있다.

# TodoView 구현

## src/views/TodoView.vue 생성

|  |
| --- |
| <template>    <div>      <h1>할일</h1>    </div>  </template>  <script>  export default {    name: 'TodoView'  }  </script> |

Todo 화면을 구현한 컴포넌트이다.

## src/router/index.js 수정

|  |
| --- |
| import { createRouter, createWebHistory } from 'vue-router'  import HomeView from '../views/HomeView.vue'  import AboutView from '../views/AboutView.vue'  import TodoView from '../views/TodoView.vue'  const routes = [    {      path: '/',      name: 'home',      component: HomeView    },    {      path: '/about',      name: 'about',      component: AboutView    },    {      path: '/todo',      name: 'todo',      component: TodoView    }  ]  const router = createRouter({    history: createWebHistory(process.env.BASE\_URL),    routes  })  export default router |

## src/App.vue 수정

|  |
| --- |
| <template>    <nav>      <router-link to="/">Home</router-link> |      <router-link to="/about">About</router-link> |     <router-link to="/todo">Todo</router-link>    </nav>    <router-view/>  </template>  <style>  #app {    font-family: Avenir, Helvetica, Arial, sans-serif;    -webkit-font-smoothing: antialiased;    -moz-osx-font-smoothing: grayscale;    text-align: center;    color: #2c3e50;  }  nav {    padding: 30px;  }  nav a {    font-weight: bold;    color: #2c3e50;  }  nav a.router-link-exact-active {    color: #42b983;  }  </style> |

## 실행

|  |
| --- |
| npm run serve |

Graphical user interface, text, application, email

Description automatically generated

# 연습문제

공지사항 화면을 구현하자. (NoticeView.vue)