

Ch01. Vue 시작

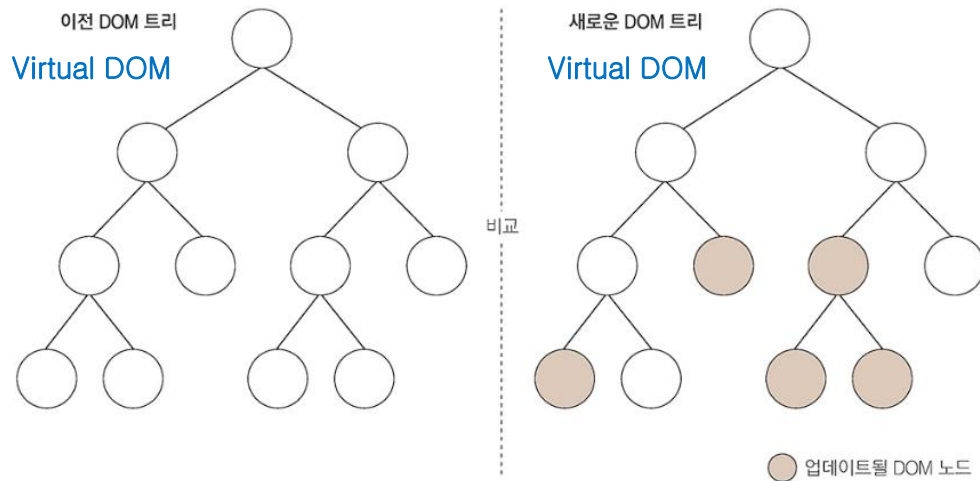
한국소프트웨어산업협회
신용권



❖ Vue 특징

➤ Front-End 뷰(View) 프레임워크:

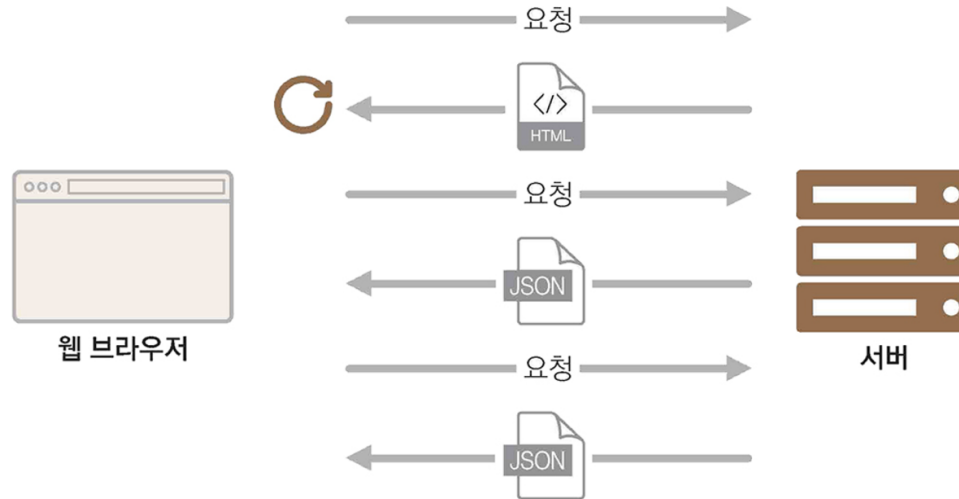
- SPA의 웹 페이지 화면 개발을 위한 프론트엔드(Front-end) 프레임워크
- 사용자 인터페이스를 컴포넌트화시킴으로 재 사용성 증가
- 전체 DOM을 변경하지 않고 이전 DOM과 차이가 나는 부분만 업데이트



1. 데이터를 업데이트하면 전체 UI를 Virtual DOM에 리렌더링합니다.
2. 이전 Virtual DOM에 있던 내용과 현재 내용을 비교합니다.
3. 바뀐 부분만 실제 DOM에 적용합니다.

- 지속적으로 데이터가 변화하는 대규모 애플리케이션에 유리
- 도큐먼트: <https://ko.vuejs.org/guide/introduction.html>

➤ 클라이언트 사이드 렌더링(CSR) – Single Page Application(SPA)



- **장점**
 - 네트워크 트래픽 감소
 - 서버 처리량 감소
 - 페이지 전환이 빠름
- **단점**
 - 자바스크립트 코드량 증가
- 첫 페이지를 제외하고 모두 AJAX 통신
- 동일 DOM 내부에서 라우팅

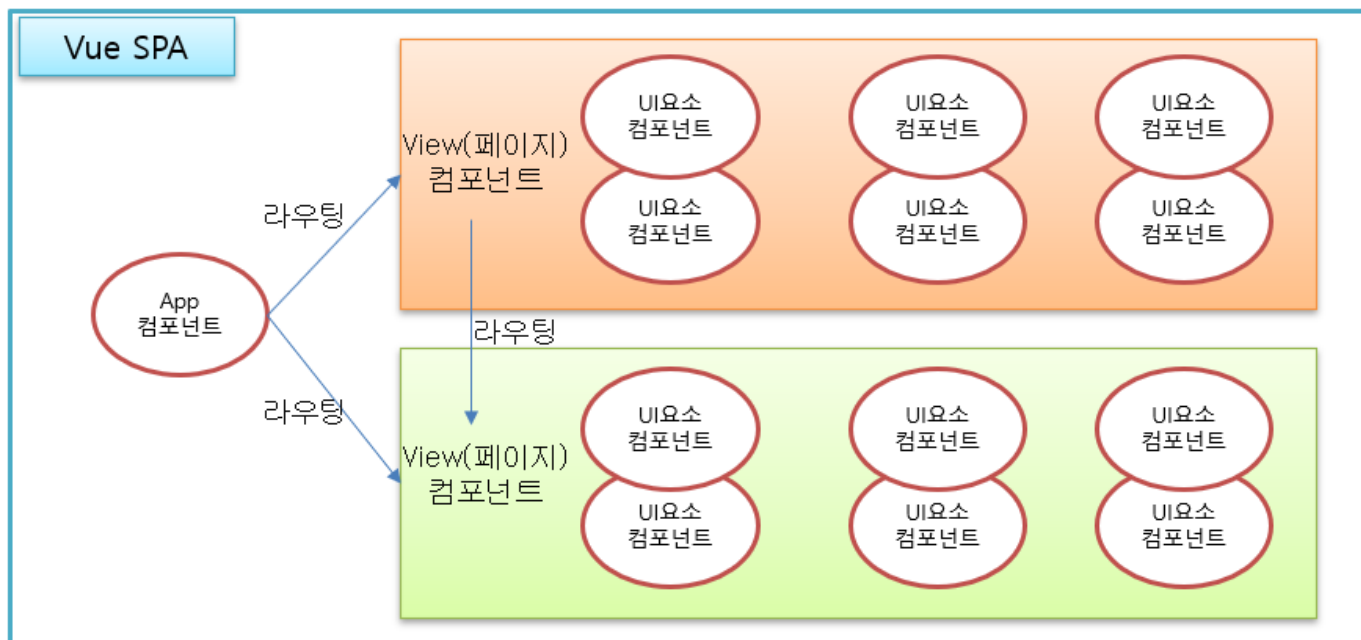
➤ 컴포넌트 개요

● Component 란

- 재사용 가능하고, 조립 가능한 독립적인 UI 객체
- HTML조각(<template>) + JavaScript(<script>) + CSS(<style>) 의 결합체

● View와 Component


- View: 화면 하나(페이지)를 말하며, View는 재사용이 어려운 컴포넌트 //views 폴더에 작성
- Component: 여러 화면(페이지) 안에 배치될 수 있는 재사용 UI 컴포넌트 //components 폴더에 작성
- 파일 확장명이 모두 **.vue** 로 작성되는 컴포넌트
- 싱글 파일 또는 폴더 형태로 작성가능



❖ 개발 환경 설정

➤ Node.js

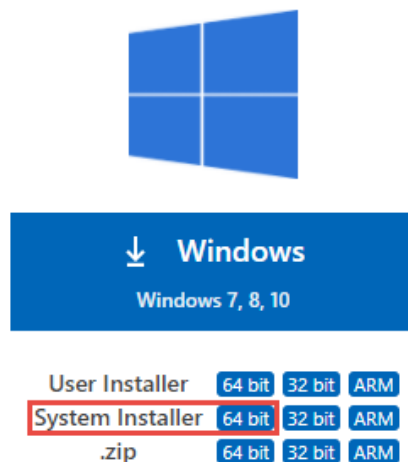
- Vue 프로젝트를 개발하는데 필요한 도구들이 Node.js를 사용하므로 설치 필요
- <https://nodejs.org/>

Node.js 다운로드 (LTS) 

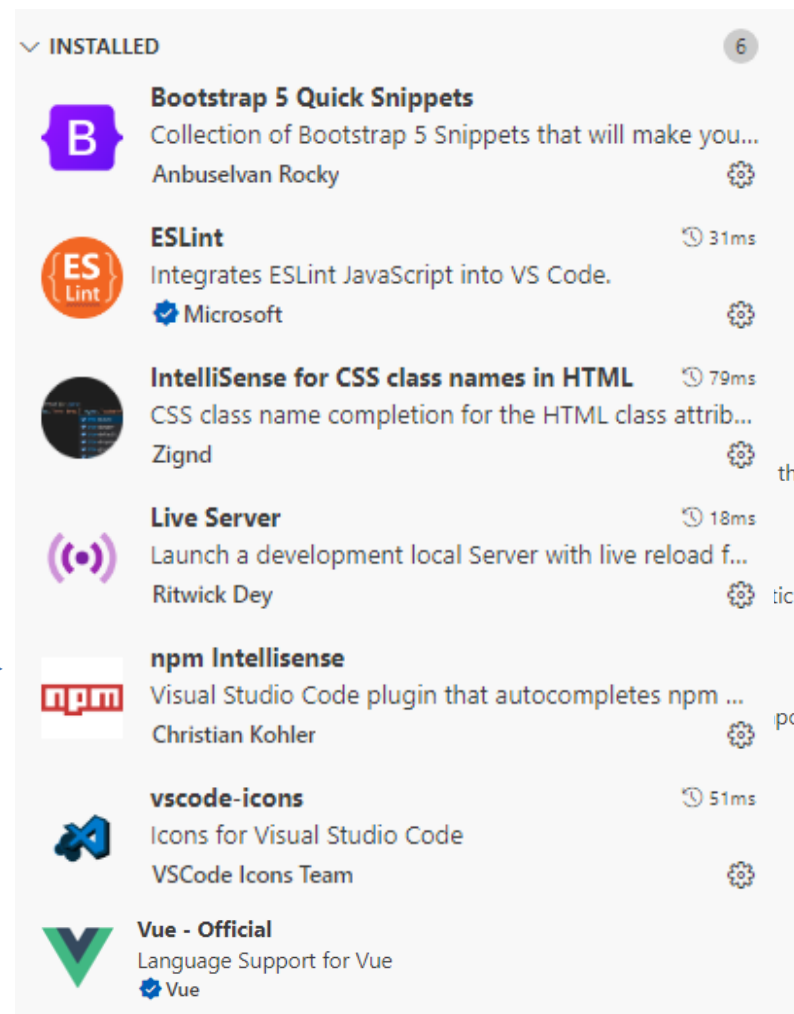
- 실행 파일: node-v22.14.0-x64.msi (기본 설치로 설치)
- 설치 확인: > node --version

❖ Visual Studio Code 설치

- <https://code.visualstudio.com/#alt-downloads>
- System Installer 64 bit 설치



- 추천되는 확장 설치 →
 - ESLint
 - 자바스크립트 문법 검사 도구
 - vscode-icons
 - 파일 유형별 아이콘 표시
 - Settings/Workbench/Appearance /Icon Theme/VSCode Icons



❖ Node.js 애플리케이션 폴더 및 package.json 생성

➤ 애플리케이션 폴더란

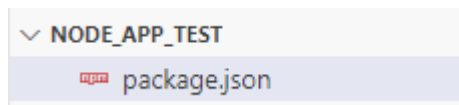
- package.json 파일이 포함된 폴더

➤ package.json 생성

- package.json: 애플리케이션 설정 및 외부 패키지들을 관리하는 파일
- 생성 방법: 애플리케이션 폴더 > **npm init**

[실습]

- node-app-test 폴더 생성, VSCode에서 폴더 열기
- node-app-test > npm init
 - 모두 Enter
- Windows PowerShell에서 스크립트를 실행할 수 없다는 메시지가 출력될 경우
 - 관리자 권한으로 Windows PowerShell 또는 VScode 실행
 - > **Set-ExecutionPolicy RemoteSigned** ← 외부 서명된 파일의 실행 정책 변경
 - > **Get-ExecutionPolicy** ← 실행 정책 변경 확인
 - » RemoteSigned
- package.json 확인



```
{
  "name": "exam10_npm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

➤ scripts 실행

- node-app-test/**app.js** 파일 생성

- app.js

```
console.log("app running...");
```

- package.json 수정

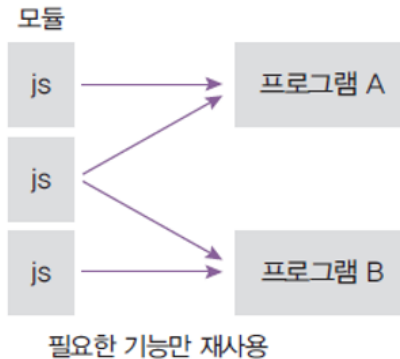
```
{  
  "name": "exam10_npm",  
  "version": "1.0.0",  
  "description": "",  
  "main": "app.js",  
  "scripts": {  
    "start": "node app.js"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

- 실행 방법: **npm run start**
- scripts 속성명이 start, test인 경우는 run 생략가능: **npm start**

❖ 모듈 이해

➤ 모듈(Module)이란

- 변수, 함수, 클래스, 인스턴스의 집합으로 독립된 파일(~.js)로 만든 것
- 모듈로 만들면 여러 프로그램에서 재사용 가능



➤ 모듈의 종류

- CommonJs 모듈
 - Node.js 기본
 - module.exports 및 exports 를 이용한 모듈 작성
 - require() 함수를 이용해서 모듈 가져옴
- EcmaScript 모듈: EcmaScript 2015(ES6)에서 지원
 - Vue 프로젝트에서 사용하는 모듈
 - export default와 export를 이용한 모듈 작성
 - import문을 이용해서 모듈을 가져옴

➤ ES6 모듈 시스템

- **export default** 또는 **export**로 내보낼 객체 또는 함수 지정
- **import**문으로 모듈 가져옴,
 - 여러 번 import 하더라도 모듈은 한번만 실행
 - 따라서 리턴된 객체 또는 함수는 동일

[실습]

- **module-test** 폴더 생성,
- **module-test>npm init** <-- **package.json** 파일 생성을 위해
- **CommonJS** 대신 **ES6 모듈 시스템** 사용하도록 설정 변경
 - **package.json**

```
{
  "name": "es6",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "type": "module"
}
```

- **module-test** 폴더에서 모듈 파일 작성

- 작성 방법

- » `export const 변수명 = ...;`
 - » `export function 함수명 = { ... };`
 - » `export class 클래스명 { ... };`

} 여러 개 작성 가능
 - » `export default 변수명;`
 - » `export default 함수명;`
 - » `export default 클래스명;`
 - » `export default { ... };`

} 한개만 작성 가능 가능

- **moduleA.js**

```
const moduleA_var1 = "moduleA_var1_value";

const moduleA_fun1 = function() {
  console.log("moduleA_fun1() 실행");
};

export default {
  moduleA_var1,
  moduleA_fun1
};

/*export default {
  moduleA_var1:moduleA_var1,
  moduleA_fun1:moduleA_fun1
};*/
```

– moduleB.js

```
export const moduleB_var1 = "moduleB_var1_value";

export const moduleB_fun1 = function() {
  console.log("moduleB_fun1() 실행");
}

const moduleB_default = function() {
  console.log("moduleB_default() 실행");
}

export default moduleB_default;
```

● module-test 폴더에서 실행 파일 작성

– app.js

```
import moduleA from "./moduleA.js";
console.log(moduleA);
console.log(moduleA.moduleA_var1);
moduleA.moduleA_fun1();
console.log("");

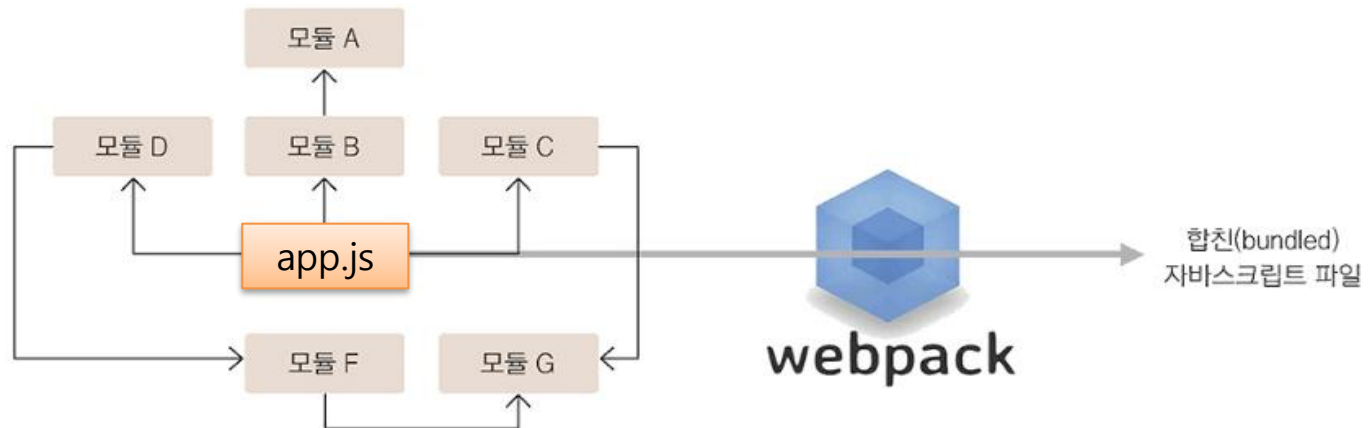
import moduleB_default, {moduleB_var1, moduleB_fun1} from "./moduleB.js";
moduleB_default();
console.log(moduleB_var1);
moduleB_fun1();
```

```
PS E:\vue\workspace-vscode\vue2-2\module-test\es6> node app.js
{
  moduleA_var1: 'moduleA_var1_value',
  moduleA_fun1: [Function: moduleA_fun1]
}
moduleA_var1_value
moduleA_fun1() 실행

moduleB_default() 실행
moduleB_var1_value
moduleB_fun1() 실행
```

● module-test>npm start 또는 node app.js

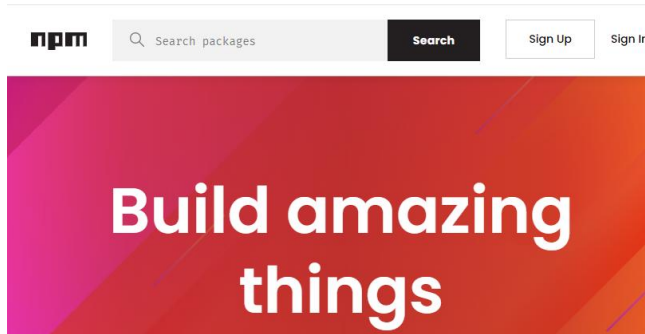
- Vue는 webpack을 이용해서 ES6 모듈 파일들을 가져와서 하나의 전체 파일 또는 몇개의 파일들로 합침(번들링)



❖ 외부 패키지 설치

➤ npm(Node Package Manager) 명령어

- npm 서버에서 외부 패키지(모듈)을 다운로드해서 설치
- npm 서버에서 패키지 검색(<https://www.npmjs.com/>)할 수 있음



- node.js를 설치하면 npm 명령어도 같이 설치: (확인: `npm -v`)

➤ 외부 패키지 설치 및 삭제

- npm-test 폴더 생성, npm-test 폴더에서 VSCode 열기
npm init

- 프로젝트별 설치(node_modules에 설치)
npm install --save 패키지명[@버전]
npm install 패키지명[@버전] //npm@5부터 --save는 생략가능
npm i 패키지명

```
npm-test> npm install jquery
```

```
package.json
"dependencies": {
  "패키지명" : "버전"
}
```

- 프로젝트별 개발용 패키지 설치(배포시에는 포함되지 않고, 개발 중에서만 사용)
npm install --save-dev 패키지명
npm install -D 패키지명

```
npm-test> npm install -D bootstrap
```

```
package.json
"devDependencies": {
  "패키지명" : "버전"
}
```

- 전역(프로젝트 공용) 패키지 설치
npm install --global 패키지명
npm install -g 패키지명

[참고: 저장위치]

```
npm-test> npm install -g @vue/cli
```

- windows: C:/Users/사용자/AppData/Roaming/npm/node_modules
- mac: /usr/local/lib/node_modules

- 깃허브에 위치한 패키지 설치
npm install 저장소주소

- package.json에 기술된 모든 패키지 설치

npm install

- 전역 패키지를 제외한 package.json에 기술된 모든 패키지를 자동 설치
- Git 사용시 node_modules 폴더는 .gitignore에 포함시켜 제외시킴
- package.json 은 협업시 패키지 버전을 통일화 시켜줌

```
[node_modules 폴더 삭제후 실행]  
npm_test> npm install
```

- 패키지 삭제

npm uninstall 패키지명

npm uninstall -g 패키지명

```
npm_test> npm uninstall jquery  
npm_test> npm uninstall bootstrap  
(npm_test\node_modules\... 해당 폴더 삭제됨)
```

```
npm_test> npm uninstall -g @vue/cli  
(C:\Users\사용자\AppData\Roaming\npm\node_modules\@vue\cli 폴더 삭제됨)
```

- 전역 패키지로 설치하지 않고, 사용하는 방법

npm install -D 패키지

- package.json에 포함되므로 npm install로 자동 재설치가 됨
- 보통 전역 패키지에는 명령어들이 포함되어 있음
- npx(Npm Package Runner)로 명령어를 실행
 - » npm@5.2.0 버전부터 새로 추가된 도구
 - » npx 명령어

```
npm_test> npm install -D @vue/cli  
npm_test> npx vue create front-end
```

- 기타 명령어

npm search 검색어

//npmjs.com에서 패키지 검색과 동일

npm info 패키지명

//npmjs.com에서 패키지 세부 정보

```
npm_test> npm search jquery  
npm_test> npm info jquery
```


➤ 패키지 버전 이해

- Major(주 버전).Minor(부 버전).Patch(업데이트수 버전)
- Major는 하위 버전과 호환되지 않은 수정 사항이 생겼을 때 올림
- Minor는 하위 버전과 호환되는 수정 사항이 생겼을 때 올림
- Patch는 기능에 버그를 해결했을 때 올림



➤ package.json 파일의 버전 기술 방법

- 1.1.1: `npm install` 시 동일한 버전만 설치
- ^1.1.1: `npm install` 시 minor 버전은 업데이트 가능함(1.2.0가 설치될 수 있음)
- ~1.1.1: `npm install` 시 patch 버전은 업데이트 가능함(1.1.2가 설치될 수 있음)
- >=, <=, >, <는 이상, 이하, 초과, 미만으로 업데이트 가능함
- @latest는 가상 최신 버전으로 업데이트 가능함
- @next로 가장 최신 배포판으로 업데이트 가능함(불안정함)
- 알파/베타/RC 버전까지 표시할 경우(1.1.1-alpha.0, 2.0.0-beta.1, 2.0.0-rc.0)

➤ package-lock.json 파일

- npm으로 패키지를 설치할 때 생성되는 파일
- node_modules에 설치된 패키지의 정확한 버전과 의존 관계가 기술되어 있음
- package-lock.json 이 존재할 때에는 `npm install` 실행시
 - package.json 을 사용하여 node_modules 를 생성하지않고
 - package-lock.json 을 사용하여 node-modules 를 생성하기 때문에
 - 협업시 node_modules에 패키지들은 모두 동일한 버전으로 설치됨

❖ Vue 프로젝트 생성 및 실행

➤ Vue CLI 설치

- Vue.js 개발을 위한 표준 도구 제공
- <https://cli.vuejs.org/guide/installation.html>
- 설치:
 - > npm install -g @vue/cli
 - > vue --version 또는 vue -V ← 설치된 Vue CLI 버전 확인

[참고]

- 업데이트: npm update -g @vue/cli
- 삭제: npm uninstall -g @vue/cli

[참고] Windows PowerShell에서 스크립트를 실행할 수 없다는 메시지가 출력될 경우

- 1) 관리자 권한으로 Windows PowerShell 또는 VScode 실행
- 2) > **Set-ExecutionPolicy RemoteSigned** ← 외부 서명된 파일의 실행 정책 변경
- 3) > **Get-ExecutionPolicy** ← 실행 정책 변경 확인
RemoteSigned

➤ Vue 새 프로젝트명 생성(VSCode에서 빠름)

● VSCode Terminal:

.../projects>**vue create front-end-vue**

? Please pick a preset:

Default ([Vue 3] babel, eslint)

Default ([Vue 2] babel, eslint)

> **Manually select features**

[Enter]

? Check the features needed for your project

(*) Babel

() TypeScript

() Progressive Web App (PWA) Support

(*) **Router**

() Vuex

() CSS Pre-processors

(*) Linter / Formatter

() Unit Testing

() E2E Testing

[Enter]

? Choose a version of Vue.js

3.x

2.x

[Enter]

? Use history mode for router? (Y/n)

[Enter]

? Pick a linter / formatter config:

> **ESLint with error prevention only**

ESLint + Airbnb config

ESLint + Standard config

ESLint + Prettier

[Enter]

? Pick additional lint features:

> (*) **Lint on save**

() Lint and fix on commit

[Enter]

? Where do you prefer placing config for Babel, ESLint, etc.?

> **In dedicated config files**

In package.json

[Enter]

? Save this as a preset for future projects? (y/N)

[Enter]

➤ Vue 프로젝트명 실행

- VSCode: front-end-vue 폴더 열기

- 개발 모드로 실행

npm run serve [-- --port=8080]

```
DONE Compiled successfully in 3581ms

App running at: Ctrl + 마우스 클릭
- Local: http://localhost:8080/
- Network: http://192.168.0.3:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```



- 프로덕션 모드로 실행

npm run build

serve dist //[참고] npm install -g serve

```
Serving! Ctrl + 마우스 클릭

- Local: http://localhost:3000
- Network: http://192.168.0.3:3000

Copied local address to clipboard!
```

[vue cli의 포트 변경]
기본 포트: 8080

vue.config.js 파일 수정

```
const { defineConfig } = ...
module.exports = defineConfig({
  transpileDependencies: true,
  devServer: {
    port: 8090
  }
})
```

[참고]

프로젝트 경로상에 &가 있을 경우 에러 발생

Error: Cannot find module 'C:\@vue\cli-service\bin\vue-cli-service.js'

❖ 자동 생성된 코드 이해

➤ public/index.html

- SPA의 최초 DOM을 생성하기 위한 시작 HTML

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">    Edge 엔진으로 렌더링하도록 설정
    <meta http-equiv="X-UA-Compatible" content="IE=edge">    Viewport를 브라우저 폭 전체를 차지하도록 하고 초기 확대 레벨은 없음
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">    브라우저 탭에 보이는 아이콘
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>    프로젝트 폴더명이 기본 값이므로 원하는 제목으로 변경할 수 있음
  <body>
    <noscript>
      <strong>
        We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly
        without JavaScript enabled. Please enable it to continue.
      </strong>
    </noscript>    브라우저가 자바스크립트를 지원하지 않을 경우 나오는 문구
    <div id="app">
      <!--
        여기에 App.vue 컴포넌트의 내용이 들어감
        main.js 파일 참고
      -->
    </div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

➤ src/main.js

- App.vue 컴포넌트를 생성
- 뷰(view) 페이지 이동을 위한 라우터 적용
- public/index.html의 <div id= "add" >에 App.vue 렌더링 내용을 삽입(마운트)

```
import { createApp } from 'vue'  
import App from './App.vue'  
import router from './router'
```

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

```
<!DOCTYPE html>  
<html lang="">  
  <head>  
    ...  
  </head>  
  <body>  
    ...  
    <div id="app">  
      <!--  
        여기에 App.vue 컴포넌트의 렌더링 내용이 들어감  
        main.js 파일 참고  
      -->  
    </div>  
    <!-- built files will be auto injected -->  
  </body>  
</html>
```

index.html

➤ src/App.vue

```
<template>
```

- App 컴포넌트의 렌더링 내용 정의

```
<nav>
```

```
<router-link to="/">Home</router-link> |  
<router-link to="/about">About</router-link>
```

- <router-link>를 클릭할 경우 to 경로와 매핑되는 뷰(view) 컴포넌트가 렌더링됨
- to 경로와 컴포넌트 매핑은 src/router/index.js에서 정의되어 있음

```
</nav>
```

```
<router-view/>
```

- <router-link>를 클릭했을 때 뷰(view) 컴포넌트의 렌더링 내용이 들어가는 태그

```
</template>
```

```
<style>
```

```
/* 전역 CSS */
```

```
#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>
```

➤ src/router/index.js

● 라우팅(페이지 이동) 설정

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

라우트
정의
(
경로와
컴포넌트
매핑
정의
)

```
const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/about',
    name: 'about',
    // route level code-splitting(라우터 레벨 코드 스플리팅)
    // 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')
  }
]
```

라우터
생성

```
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})
```

```
export default router
```

<router-link to="/">와 매핑되는 컴포넌트 지정

<router-link to="/about">와 매핑되는 컴포넌트 지정

개발시 브라우저 갱신시 히스토리 정보를 이용해서 라우팅하도록 설정

디폴트 BASE_URL 값: /

➤ src/views/HomeView.vue

● 뷰(view) 컴포넌트인 HomeView 정의

```
<template>
  <div class="home">
    
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>
```

HelloView의 렌더링 내용

HelloWorld 컴포넌트의 렌더링 내용을 삽입

```
<script>
  // @ is an alias to /src
  import HelloWorld from '@components/HelloWorld.vue'

  export default {
    name: 'HomeView',
    components: {
      HelloWorld
    }
  }
</script>
```

HelloWorld 컴포넌트 가져오기

컴포넌트 이름 및 템플릿 내부에 사용되는 컴포넌트를 지정해서 내보내기 (Options API 방식)

[참고: Options API 방식과 Composition API 방식의 차이점]
<https://ko.vuejs.org/guide/introduction.html#api-styles>



다음 Composition API 권장 방식으로 변경 가능
(<https://ko.vuejs.org/api/sfc-script-setup>)

```
<script setup>
  // @ is an alias to /src
  import HelloWorld from '@components/HelloWorld.vue'
</script>
```

➤ src/components/HelloWorld.vue

- UI 요소 컴포넌트인 HelloWorld 정의
- 뷰(view) 컴포넌트인 HomeView 내부에 UI 요소로 사용됨

```
<template>
  <div class="hello">
    <h1> {{ msg }} </h1>
    <p>
      For a guide and recipes on how to configure / customize this project,<br>
      check out the
      <a href="https://cli.vuejs.org" target="_blank" rel="noopener">vue-cli documentation</a>.
    </p>
    ...
  </div>
</template>
```

HelloWorld 컴포넌트의 렌더링 내용

msg 값을 바인딩하는 표현식

```
<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>
```

컴포넌트 이름 지정

<HelloWorld msg="...">로 전달되는 msg를 props에 등록
=> <template>에서 {{msg}}와 같이 바인딩할 수 있음

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
  h3 {
    margin: 40px 0 0;
  }
  ...
</style>
```

HelloWorld 컴포넌트 내에서만 사용하도록 범위 제한

➤ src/views/AboutView.vue

- 뷰(view) 컴포넌트인 AboutView 정의

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

AboutView 컴포넌트의 렌더링 내용

➤ 설정 파일

● .browserslistrc

- 지원 또는 지원하지 않는 브라우저 정보 설정 파일
- <https://github.com/browserslist/browserslist>

```
> 1%           //전세계 1% 이상 점유율을 가진 브라우저만 지원
last 2 versions //최근 2개 버전의 브라우저만 지원
not dead       //지원이 중단되지 않은 브라우저만 지원
not ie 11      //ie 11은 지원하지 않음
```

● .eslintrc.js

- EcmaScript의 문법을 검사하기 위한 설정 파일
- <https://eslint.org/>

● babel.config.js

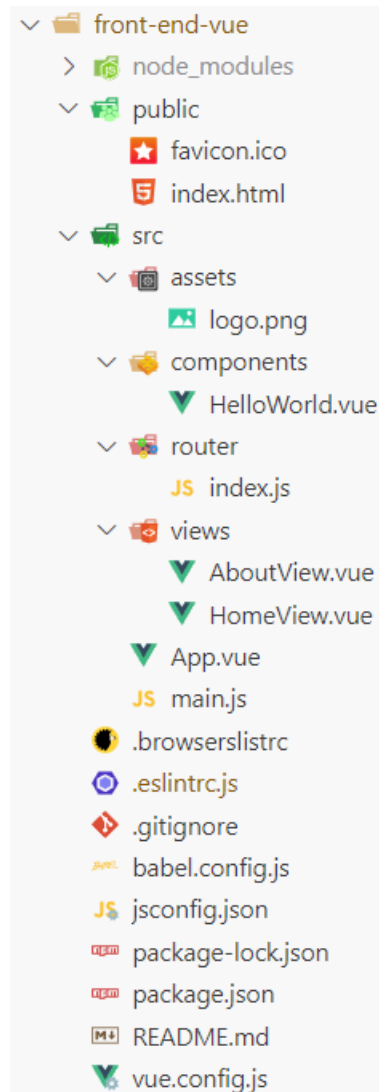
- Babel 설정 파일
- Babel
 - » 크로스 브라우징을 위해
 - » ES6+ 이상의 JavaScript를 하위 버전의 JavaScript 문법으로 변환하는 역할
- <https://babeljs.io/>

● jsconfig.json

- JavaScript 프로젝트 설정 파일로 VSCode에서 사용함
- 프로젝트 소스에 포함 또는 제외되는 파일 설정, 컴파일 옵션 설정
- 이 파일이 있으면 해당 폴더는 JavaScript 프로젝트 루트임을 나타냄
- <https://code.visualstudio.com/docs/languages/jsconfig>

● vue.config.js

- vue 설정 파일
- <https://cli.vuejs.org/config/>



❖ 프로젝트 재구성

➤ 설정 파일 수정 및 생성

- .eslintrc.js 파일 수정: 사용하지 않는 변수 에러 표시 제거

```
module.exports = {
  root: true,
  env: {
    node: true
  },
  'extends': [
    'plugin:vue/vue3-essential',
    'eslint:recommended'
  ],
  parserOptions: {
    parser: '@babel/eslint-parser'
  },
  rules: {
    'no-console': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
    'no-debugger': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
    'no-unused-vars': 'off'
  }
}
```

➤ bootstrap 모듈 설치

- npm install [bootstrap@5.3.3](#)
- npm install [@popperjs/core](#)
- src/main.js 수정

```
import { createApp } from 'vue'  
import App from './App.vue'  
import router from './router'
```

```
import 'bootstrap/dist/css/bootstrap.min.css';  
import 'bootstrap/dist/js/bootstrap.min.js';
```

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

➤ 코드 수정

● src/components/HelloWorld.vue

- 컴포넌트 영역 표시를 위해 템플릿 코드 추가
- Composition API 스타일을 사용하도록 변경

```
<template>
  <div class="card">
    <div class="card-header">
      HelloWorld
    </div>
    <div class="card-body">

      <div class="hello">
        <h1>{{ msg }}</h1>
        ...(기존 내용)
      </div>

    </div>
  </div>
</template>
```

```
<script setup>
//import { defineProps } from "vue";
const props = defineProps(['msg'])
</script>
```

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
...(기존 내용)
</style>
```

API Styles

<https://ko.vuejs.org/guide/introduction.html#api-styles>
Vue 컴포넌트는 옵션(Options) API와
컴포지션(Composition) API 두 가지 스타일로 작성할 수
있다.

Options ☐ Composition ☒

[Options API]

빌드 도구를 사용하지 않거나 점진적 향상과 같이
복잡성이 낮은 시나리오에서 주로 Vue를 사용할
계획이라면 Options API를 사용

[Composition API]

Vue로 전체 애플리케이션을 빌드하려는 경우
Composition API + Single-File Components 를 사용

[.eslintrc.js 파일 수정]
import 문없이 defineProps() 함수를 사용할 수 있도록 다음과 같이 설정
...
env: {
 node: true,
 'vue/setup-compiler-macros': true
}
...

- **src/views/HomeView.vue**
 - 컴포넌트 영역 표시를 위해 템플릿 코드 추가
 - Composition API 스타일을 사용하도록 변경

```
<template>
  <div class="card">
    <div class="card-header">
      HomeView
    </div>
    <div class="card-body">
      <div class="home">
        
        <HelloWorld msg="Welcome to Your Vue.js App"/>
      </div>
    </div>
  </div>
</template>
```

```
<script setup>
// @ is an alias to /src
import HelloWorld from '@components/HelloWorld.vue'
</script>
```

```
<style scoped>
.home {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
}
</style>
```

src/App.vue에서 복사해서 사용

```
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
}
```



- **src/views/AboutView.vue**
 - 컴포넌트 영역 표시를 위해 템플릿 코드 추가

```
<template>
  <div class="card">
    <div class="card-header">
      AboutView
    </div>
    <div class="card-body">

      <div class="about">
        This is an about page
      </div>

    </div>
  </div>
</template>
```

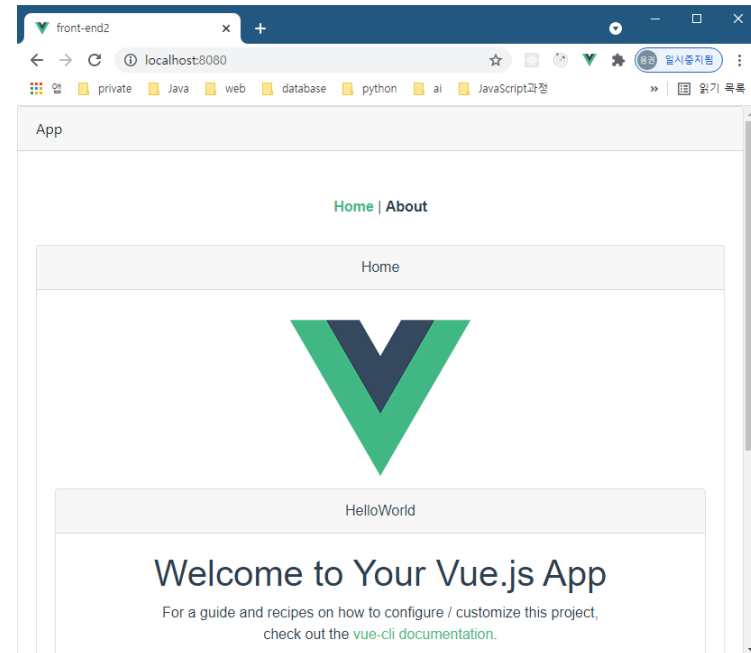
- **src/views/App.vue**
 - 컴포넌트 영역 표시를 위해 템플릿 코드 추가

```
<template>
  <div class="card">
    <div class="card-header">
      App
    </div>
    <div class="card-body">

      <nav>
        <router-link to="/">Home</router-link> |
        <router-link to="/about">About</router-link>
      </nav>
      <router-view/>

    </div>
  </div>
</template>

<style>
  ...
</style>
```



➤ 헤더바 및 메뉴바 만들기

- 헤더바: `src/components/AppHeader.vue`

```
<template>
  <nav class="navbar justify-content-between bg-dark text-white">
    <div class="ms-2">
      <RouterLink to="/" class="navbar-brand">
        
        <span class="ms-2 text-white">Vue.js</span>
      </RouterLink>
    </div>

    <div class="me-2">
      <RouterLink to="/" class="btn btn-success btn-sm">로그인</RouterLink>
    </div>
  </nav>
</template>

<script setup>
</script>

<style scoped>
</style>
```

- 메뉴바: `src/components/AppMenu.vue`
 - 파일 복사해서 붙여넣기

```
<template>
  <div class="accordion" id="accordionExample">
    <div class="accordion-item">
      <h2 class="accordion-header">
        <button class="accordion-button fw-bold" type="button" data-bs-toggle="collapse"
          data-bs-target="#collapseOne" aria-expanded="true" aria-controls="collapseOne">
          Ch01. Vue 시작
        </button>
      </h2>
      <div id="collapseOne" class="accordion-collapse collapse show"
        data-bs-parent="#accordionExample">
        <div class="accordion-body">
          <ul class="nav nav-underline flex-column">
            <li class="nav-item">
              <RouterLink to="/" class="nav-link">Home</RouterLink>
              <RouterLink to="/about" class="nav-link">About</RouterLink>
            </li>
          </ul>
        </div>
      </div>
    </div>
  </div>
  ...
</div>
</template>
```

- `src/main.js`

```
...
// [Vue Router warn]이 출력되지 않도록 설정
const originalWarn = console.warn;
console.warn = function (msg, ...args) {
  if (msg.includes('[Vue Router warn]')) { return; }
  originalWarn(msg, ...args);
};
```

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

- **헤더바 및 메뉴바 추가: src/App.vue :**

```
<template>
  <div className="d-flex flex-column vh-100">
    <AppHeader />
    <div className="flex-grow-1 container-fluid">
      <div className="row">

        <div className="col-md-4 col-lg-3 p-2">
          <AppMenu />
        </div>

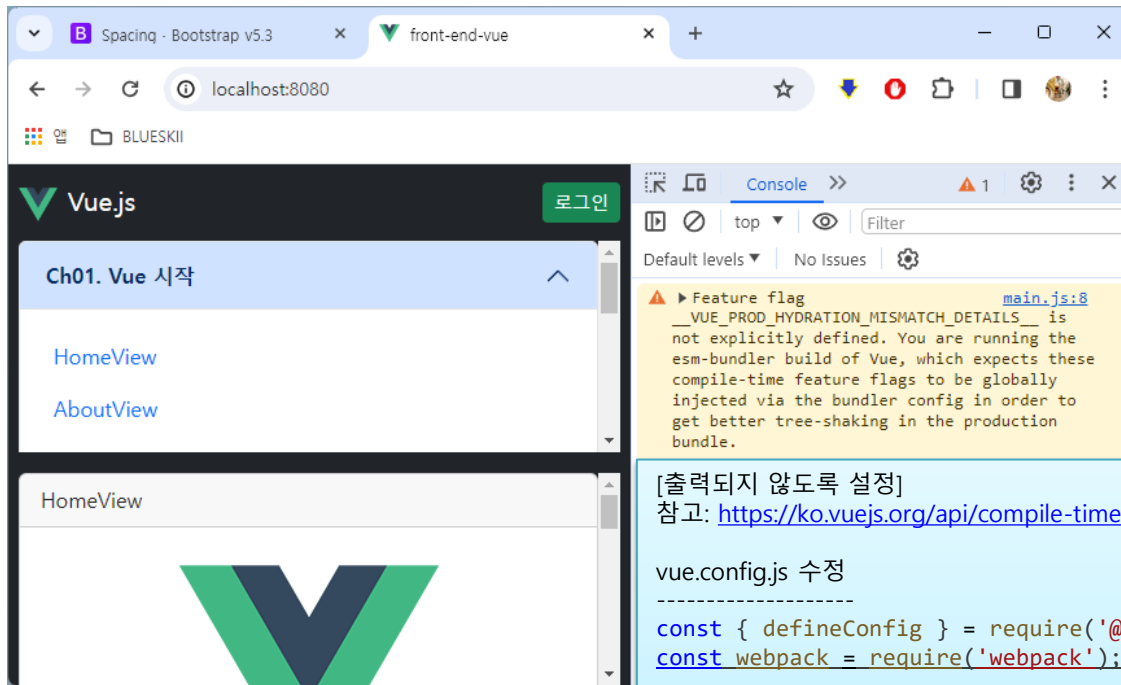
        <div className="col-md-8 col-lg-9 p-2">
          <router-view/>
        </div>

      </div>
    </div>
  </div>
</template>

<script setup>
import AppHeader from "@/components/AppHeader.vue";
import AppMenu from "@/components/AppMenu.vue";
</script>

<style>
</style>
```

- npm run serve



– npm run serve 재실행

[출력되지 않도록 설정]

참고: <https://ko.vuejs.org/api/compile-time-flags>

vue.config.js 수정

```
const { defineConfig } = require('@vue/cli-service')
const webpack = require('webpack');

module.exports = defineConfig({
  transpileDependencies: true,
  configureWebpack: {
    plugins: [
      new webpack.DefinePlugin({
        __VUE_OPTIONS_API__: 'true',
        __VUE_PROD_DEVTOOLS__: 'false',
        __VUE_PROD_HYDRATION_MISMATCH_DETAILS__: 'false'
      })
    ]
  }
})
```

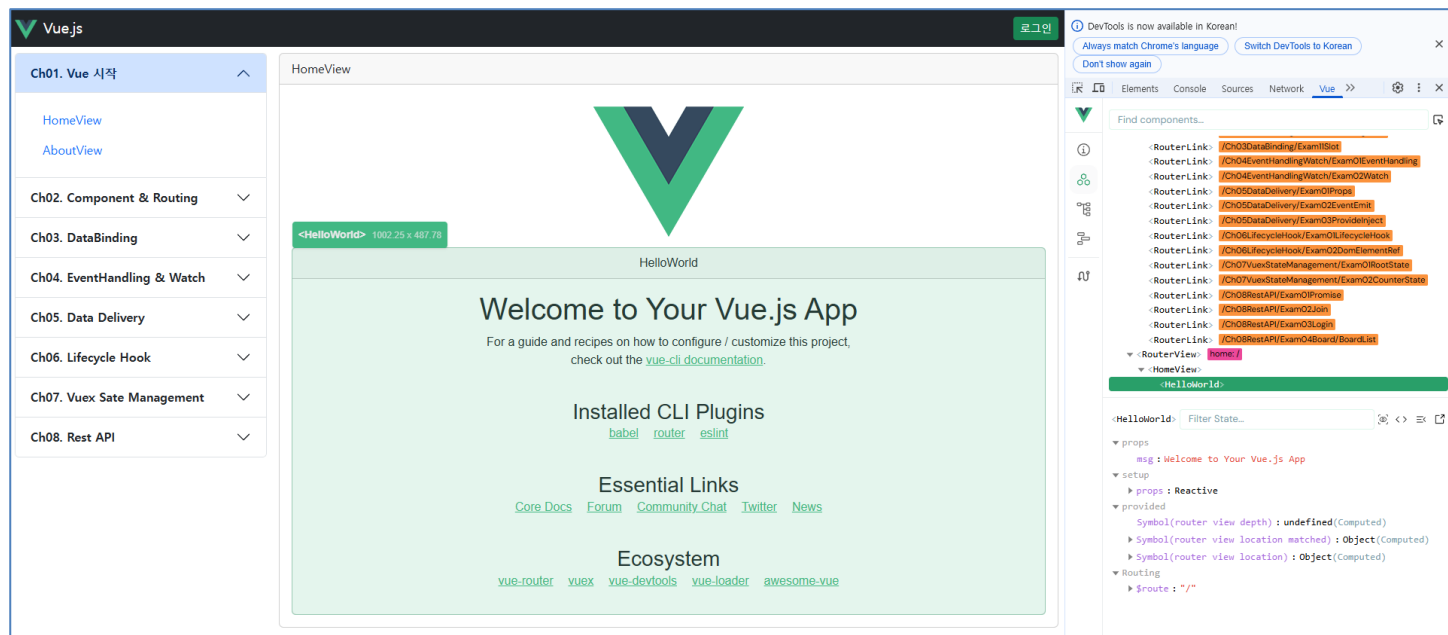
npm run serve 로 재시작

- **Vue.js Devtools 크롬 확장 프로그램 설치**
 - **Vue.js 애플리케이션 디버깅을 위한 Chrome 확장 프로그램**



— 설치후 Chrome 재실행

[개발 모드로 실행했을 때만 작동]



Ch02. Component & Routing

한국소프트웨어산업협회
신용권



❖ Vue 컴포넌트 이해

➤ Vue 컴포넌트 종류

- 화면 컴포넌트(src/views/...vue): 하나의 화면을 정의하는 컴포넌트
- 공용 컴포넌트(src/components/...vue): 화면마다 포함되는 공유 컴포넌트

➤ Vue 싱글 파일 컴포넌트(Single-File Component, SFC) 구조

```
<template>
  <!--
  Vue2: template의 루트 태그는 하나만 작성해야 함
  Vue3: 상관없음
  -->
  <div>
    <!-- 화면에 표시할 내용을 태그로 작성하는 영역 -->
  </div>
</template>

<script setup>
  // Compositon API 스타일로 자바스크립트를 작성 영역
  // 뷰 모델 정의, 외부 컴포넌트 가져오기, 이벤트 처리, 컴포넌트 상태 처리
</script>

<style scoped>
  /* 컴포넌트 CSS 스타일 작성 */
  /* scoped 없으면 전체 컴포넌트에서 사용 가능한 전역 Style이 됨 */
  /* scoped 있으면 현재 컴포넌트에서만 사용 가능한 지역 Style이 됨*/
</style>
```

❖ 화면 컴포넌트

➤ 화면 컴포넌트 생성 방법

- src/views 폴더에서 두가지 형태로 작성 가능
- 컴포넌트이름.vue 싱글 파일 컴포넌트로 생성
- 컴포넌트이름/index.vue 폴더 컴포넌트로 생성(멀티 파일로 구성될 경우)

➤ 싱글 파일 컴포넌트 생성

- Exam01SingleFileComponent.vue

```
<template>
  <div class="card">
    <div class="card-header">Exam01SingleFileComponent</div>
    <div class="card-body">
      
    </div>
  </div>
</template>

<script setup>
  console.log("Exam01SingleFileComponent 입니다.");
</script>

<style scoped>
</style>
```

[컴포넌트 이름 작성시 주의사항]

- 두가지 이상의 단어(multi-word)가 결합된 이름을 사용해야함
- 단일 이름을 사용할 경우 eslint에 의해 에러 표시됨

- 이 기능을 끄기위해 다음과 같이 설정할 수 있음

```
-----
.eslintrc.js
```

```
rules: {
```

```
  ...,
  'vue/multi-word-component-names': 'off'
}
```

➤ 폴더 컴포넌트 생성

● Exam02FolderComponent/index.vue

```
<template>
  <div class="card">
    <div class="card-header">Exam02FolderComponent</div>
    <div class="card-body">
      
    </div>
  </div>
</template>

<script setup>
console.log("Exam02FolderComponent 입니다.");
</script>

<style scoped>
</style>
```

❖ 라우팅(routing)

➤ 라우팅이란

- URL 주소에 따라 페이지가 전환되는 것
- Vue는 URL 주소에 따라 화면 컴포넌트인 View가 교체

➤ Vue Router

- Vue에서 라우팅 기능을 제공하는 공식 라이브러리
- <https://router.vuejs.kr/>
- Router v4.x: Vue3
- Router v3.x: Vue2

➤ 설치

- 프로젝트 생성시 체크하지 않았을 경우 다음 명령어를 실행해서 프로젝트에 추가할 수 있음
➤ **vue add router**

➤ 라우팅 모듈 이해

● route 정의 및 router 인스턴스 생성

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

라우트
정의

```
const routes = [                                라우트 정의 배열 생성
  {
    path: '/',                                  라우트 URL
    name: 'home',                              라우트 이름(생략 가능)
    component: HomeView                       보여줄 View 컴포넌트
  },
  {
    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')
  }
]
```

라우터
인스턴스 생성
및
내보내기

```
const router = createRouter({                  라우터 인스턴스 생성
  history: createWebHistory(process.env.BASE_URL),
  routes                                     디폴트 BASE_URL 값: /
})                                           라우트 정의 배열 지정 (routes : routes)
```

```
export default router                        라우터를 기본 내보내기
```

URL 전환시 history 객체 이용(URL에 #이 안붙음)
매개값: 포트번호 다음에 고정적으로 붙는 경로
process.env.변수명: 시스템 환경변수 값을 읽음
참고: <https://router.vuejs.kr/guide/essentials/history-mode.html>

➤ 라우트 등록

- **src/router/Ch02ComponentRouting.js**

```
const routes = [  
  {  
    path: '/Ch02ComponentRouting/Exam01SingleFileComponent',  
    name: 'Exam01SingleFileComponent',  
    component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */  
                           '@views/Ch02ComponentRouting/Exam01SingleFileComponent'),  
  },  
  {  
    path: '/Ch02ComponentRouting/Exam02FolderComponent',  
    component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */  
                           '@views/Ch02ComponentRouting/Exam02FolderComponent'),  
  },  
];  
  
export default routes;
```

- **src/router/index.js**

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
import Ch02ComponentRouting from "../Ch02ComponentRouting";

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')
  },
  ...Ch02ComponentRouting
]

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})

export default router
```

➤ 메뉴 컴포넌트(src/components/AppMenu.vue)

```
<div class="accordion-item">
  <h2 class="accordion-header">
    <button class="accordion-button collapsed fw-bold" type="button" data-bs-toggle="collapse"
      data-bs-target="#collapse2" aria-expanded="false" aria-controls="collapse2">
      Ch02. Component & Routing
    </button>
  </h2>
  <div id="collapse2" class="accordion-collapse collapse" data-bs-parent="#accordionExample">
    <div class="accordion-body">
      <ul class="nav flex-column">
        <li class="nav-item">
          <RouterLink to="/Ch02ComponentRouting/Exam01SingleFileComponent" class="nav-link">
            Exam01SingleFileComponent
          </RouterLink>
        </li>
        <li class="nav-item">
          <RouterLink to="/Ch02ComponentRouting/Exam02FolderComponent" class="nav-link">
            Exam02FolderComponent
          </RouterLink>
        </li>
      </ul>
    </div>
  </div>
</div>
```


❖ UI 컴포넌트

➤ 공용 UI 컴포넌트

- 여러 화면에 포함되는 공용 컴포넌트
- 작성 위치: `src/components` 폴더
 - 싱글 파일 컴포넌트: `컴포넌트이름.vue`
 - 폴더 컴포넌트(멀티 파일로 구성될 경우): `컴포넌트이름/index.vue`
- 실습
 - `src/components/Ch02ComponentRouting/UIComponentA.vue`

```
<template>
  <div class="card">
    <div class="card-header">UIComponentA</div>
    <div class="card-body"></div>
  </div>
</template>

<script setup>
</script>

<style scoped>
</style>
```

– **src/components/Ch02ComponentRouting/UIComponentB.vue**

```
<template>
  <div class="card">
    <div class="card-header">UIComponentB</div>
    <div class="card-body"></div>
  </div>
</template>

<script setup>
</script>

<style scoped>
</style>
```

– **src/components/Ch02ComponentRouting/UIComponentC.vue**

```
<template>
  <div class="card">
    <div class="card-header">UIComponentC</div>
    <div class="card-body"></div>
  </div>
</template>

<script setup>
</script>

<style scoped>
</style>
```

➤ 로컬 UI 컴포넌트

- 특정 화면에서만 사용되는 컴포넌트
- 작성 위치: src/views/컴포넌트 폴더
 - 컴포넌트이름.vue **싱글 파일 컴포넌트로 생성**
 - 컴포넌트이름/index.vue **폴더 컴포넌트로 생성(멀티 파일로 구성될 경우)**
- 실습
 - Exam03UseUIComponent/UIComponentD.vue

```
<template>
  <div class="card">
    <div class="card-header">UIComponentD</div>
    <div class="card-body"></div>
  </div>
</template>

<script setup>
</script>

<style scoped>
</style>
```

➤ 공용 및 로컬 컴포넌트 사용

● Exam03UseUIComponent/index.vue

```
<template>
  <div class="card">
    <div class="card-header">Exam03UseUIComponent</div>
    <div class="card-body">
      <UIComponentA/>
      <UIComponentB class="mt-3"/>
      <UIComponentC class="mt-3"/>
      <UIComponentD class="mt-3"/>
    </div>
  </div>
</template>

<script setup>
import UIComponentA from "@components/Ch02ComponentRouting/UIComponentA.vue";
import UIComponentB from "@components/Ch02ComponentRouting/UIComponentB.vue";
import UIComponentC from "@components/Ch02ComponentRouting/UIComponentC.vue";
import UIComponentD from "../UIComponentD.vue";
</script>

<style scoped>
</style>
```

● 컴포넌트의 class 속성

- 컴포넌트의 class 속성은 해당 컴포넌트의 <template>의 루트 엘리먼트의 class로 자동 추가됨
- <UIComponentB class="mt-3">

➤ 라우트 등록 및 메뉴 컴포넌트

● 라우트 등록(src/router/Ch02ComponentRouting.js)

```
{  
  path: "/Ch02ComponentRouting/Exam03UseUIComponent",  
  component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */  
    "../views/Ch02ComponentRouting/Exam03UseUIComponent"),  
},
```

● 메뉴 컴포넌트(src/components/AppMenu.vue)

```
<RouterLink to="/Ch02ComponentRouting/Exam03UseUIComponent" class="nav-link">  
  Exam03UseUIComponent  
</RouterLink>
```

❖ 뷰 이동

➤ 태그 방식

- <RouterLink> 또는 <router-link> 태그 이용
 - <RouterLink to="정적라우트URL">링크문자열</RouterLink>
 - <RouterLink v-bind:to="백틱바인딩문자열 또는 { ... }">링크문자열</RouterLink>
 - <RouterLink :to="백틱바인딩문자열 또는 { ... }">링크문자열</RouterLink>
- Exam04ViewNavigation/index.vue

```
<RouterLink to="/Ch02ComponentRouting/Exam01SingleFileComponent">
```

```
<RouterLink v-bind:to="`/Ch02ComponentRouting` + `/Exam01SingleFileComponent`">
```

```
<RouterLink :to="`/Ch02ComponentRouting` + `/Exam01SingleFileComponent`">
```

```
<RouterLink :to="{path: '/Ch02ComponentRouting/Exam01SingleFileComponent'}">
```

```
<RouterLink :to="{name: 'Exam01SingleFileComponent'}">
```

Component & Routing

➤ 프로그래밍 방식

- <https://router.vuejs.kr/guide/essentials/navigation.html>

- Composition API 방식에서 Router 인스턴스 얻기

```
import { useRouter } from 'vue-router';
const router = useRouter();
```

- 화면 이동 및 콜백 처리

- **router.push()** 메소드는 promise를 리턴하므로 비동기로 처리
- 따라서 성공 여부시 추가 코드를 실행할 수 있음

<pre>.then(() => { ... }) .catch(() => { ... })</pre>	<pre>.then(() => { ... }, () => { ... })</pre>	<pre>async function() { try { await router.push(...); } catch(error) { ... } }</pre>
---	--	--
- vue-router는 경로가 매칭되지 않을 경우에도 에러를 발생시키지 않기 때문에 성공 콜백만 작성


```
router.push(`/Ch02ComponentRouting/${path}`)
  .then(() => {
    console.log("이동 완료");
  });
```

- Exam04ViewNavigation/index.vue

```
<button v-on:click="goUrl('Exam01SingleFileComponent')">Exam01SingleFileComponent</button>
<button @click="goUrl('Exam02FolderComponent')">Exam02FolderComponent</button>
```

```
<script setup>
import {useRouter} from 'vue-router'
const router = useRouter();
function goUrl(path) {
  router.push("/Ch02ComponentRouting/" + path)
  .then(() => {
    console.log("이동 완료");
  });
}
</script>
```

❖ 중첩된 라우트

➤ 컴포넌트 대치

- 서버 URL에 따라 화면 내부의 컴포넌트를 교체해서 보여줌
 - `http://localhost:8080/Ch02ComponentRouting/Exam05NestedRoute/UIComponentA`
 - `http://localhost:8080/Ch02ComponentRouting/Exam05NestedRoute/UIComponentB`

- 컴포넌트가 대치될 위치에 자리 표시자 작성

`<!-- 중첩된 라우트의 컴포넌트가 보여질 위치 -->`
`<RouterView/>` 또는 `<router-view></router-view>`

- 템플릿

```
                                /Ch02ComponentRouting/Exam05NestedRoute/UIComponentA
<RouterLink to="UIComponentA">UIComponentA</RouterLink>
<RouterLink to="UIComponentB">UIComponentB</RouterLink>
<button @click="changeNestedComponent()">UIComponentC</button>

<script setup>
import { useRouter } from 'vue-router';
const router = useRouter();

function changeNestedComponent() {
  router.push("UIComponentC");
}
</script>
```

.../Exam05NestedRoute/UIComponentA

[Exam05NestedRoute/index.vue]

```
<router-view>
  UIComponentA
</router-view>
```

.../Exam05NestedRoute/UIComponentB

[Exam05NestedRoute/index.vue]

```
<router-view>
  UIComponentB
</router-view>
```


- 라우트 등록
 - src/router/Ch02ComponentRouting.js

```
{
  path: '/Ch02ComponentRouting/Exam05NestedRoute',
  component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */
    '@/views/Ch02ComponentRouting/Exam05NestedRoute'),
  // /Ch02ComponentRouting/Exam05NestedRoute로 요청했을 경우 리다이렉트
  redirect: '/Ch02ComponentRouting/Exam05NestedRoute/UIComponentA',
  children: [
    {
      path: "UIComponentA", // /Ch02ComponentRouting/Exam05NestedRoute/UIComponentA
      component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */
        '@/components/Ch02ComponentRouting/UIComponentA.vue')
    },
    {
      path: "UIComponentB", // /Ch02ComponentRouting/Exam05NestedRoute/UIComponentB
      component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */
        '@/components/Ch02ComponentRouting/UIComponentB.vue')
    },
    {
      path: "UIComponentC", // /Ch02ComponentRouting/Exam05NestedRoute/UIComponentC
      component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */
        '@/components/Ch02ComponentRouting/UIComponentC.vue')
    }
  ]
},
```

❖ URL 데이터 전달

➤ Path Variables로 전달

● 템플릿

```
<RouterLink to="Exam06UrlDataReceive/notice/blue">
<RouterLink v-bind:to="` Exam06UrlDataReceive/${kind}/${color}`">
<RouterLink :to="{path: ` Exam06UrlDataReceive/${kind}/${color}`}">
<RouterLink :to="{name: 'Exam06UrlDataReceive', params: {kind: kind, color: color}}">
<button @click="goUrl1()">
```

● 프로그램

```
import { useRouter } from 'vue-router';
const router = useRouter();

let kind = "freeboard";
let color = "blue";

function goUrl1() {
  router.push(` Exam06UrlDataReceive/${kind}/${color}`);
}
```

● 라우트 등록(src/router/Ch02ComponentRouting.js)

```
{
  path: '/Ch02ComponentRouting/Exam06UrlDataReceive/:kind?/:color?',
  component: () => import(/* webpackChunkName: "Ch02ComponentRouting" */
    '@views/Ch02ComponentRouting/Exam06UrlDataReceive')
}
```

?를 붙이지 않으면 Path Variables 값을 반드시 제공해야 함

➤ Query String으로 전달

● 템플릿

```
<RouterLink to="Exam06UrlDataReceive?kind=notice&color=blue">  
<RouterLink :to="`Exam06UrlDataReceive?kind=${kind}&color=${color}`">  
<button @click="goUrl2()">
```

● 프로그램

```
import { useRouter } from 'vue-router';  
const router = useRouter();  
  
let kind = "freeboard";  
let color = "blue";  
  
function goUrl2() {  
  router.push(`Exam06UrlDataReceive?kind=${kind}&color=${color}`);  
}
```

➤ 데이터 받기

- **src/views/Ch02ComponentRouting/Exam06DataReceive/index.vue**

```
<template>
  <div class="card">
    <div class="card-header">Exam06UrlDataReceive</div>
    <div class="card-body">
      <h5>경로변수로 전달된 데이터</h5>
      <p>전달된 게시판 종류: {{ route.params.kind }}</p>
      <p>전달된 색상 이름: {{ route.params.color }}</p>

      <h5 class="mt-5">쿼리스트링으로 전달된 데이터</h5>
      <p>전달된 게시판 종류: {{ route.query.kind }}</p>
      <p>전달된 색상 이름: {{ route.query.color }}</p>
    </div>
  </div>
</template>

<script setup>
import { useRoute } from 'vue-router';
const route = useRoute();

console.log("route.params.kind:", route.params.kind);
console.log("route.params.color:", route.params.color);
console.log("route.query.kind:", route.query.kind);
console.log("route.query.color:", route.query.color);
</script>

<style scoped>
</style>
```

Ch03. Data Binding

한국소프트웨어산업협회
신용권



Data Binding

❖ 데이터 바인딩(Data Binding)

➤ 데이터 바인딩

- 단방향(one-way): 데이터가 변경되면 UI 요소 내용 변경
- 양방향(two-way): 데이터 변경 \leftrightarrow UI 요소의 변경, 주로 데이터와 폼 양식간의 바인딩

➤ 표현식(mustaches)을 이용한 바인딩

- `{{ 변수명 }}` //{{ number }}
- `{{ 삼항연산식 }}` //{{ (number>0)? "YES" : "NO" }}
- `{{ 하나의 값을 산출하는 연산식 }}` //{{ number + 1 }}
- `{{ 하나의 값을 리턴하는 메소드호출 }}` //{{ getNumber() }}
- `{{ computed속성명 }}` //{{ getMethod }}

➤ V-디렉티브(directives)를 이용한 바인딩

- v-html = "변수명" //데이터 -> 태그 내용으로 바인딩
- v-bind:속성명 = "변수명" //데이터 -> 속성값으로 바인딩
- v-model = "변수명" //데이터 <-> 폼 양식과의 양방향 바인딩
- v-if = "변수명 | true 또는 false를 산출하는 연산식" //요소포함여부
- v-show = "변수명 | true 또는 false를 산출하는 연산식" //요소보기여부
- v-for = "(item, index) in array" v-bind:key = "item.key|index" //요소반복

Data Binding

❖ 표현식을 이용한 바인딩

➤ {{ ... }}

- 단 하나의 값을 갖는 변수, 연산식, 리턴값 있는 함수 호출이 올 수 있음
- 속성 값으로 표현식을 사용할 수 없음(v-디렉티브 이용)

```
<p>번호: {{ no }} </p>
<p>이름: {{ name + " - " + company }} </p>
<p>설명: {{ detail.info }} </p>
<p>상태: {{ detail.sale ? "판매" : "품절" }} </p>
<p>가격: {{ getPrice() }} </p>
```

```
<script setup>
  let no = 1;
  let name = "미니백";
  let company = "클레인";

  let detail = {
    info: "시그너츠 Cecyle Lock 마그네틱 클로저가 특징입니다.",
    sale: false
  };

  let price = 300000;
  function getPrice() {
    return price;
  }
</script>
```

Data Binding

❖ 상태 데이터 바인딩

➤ 일반 데이터

- 일반 데이터는 값이 변경되어도 컴포넌트가 리렌더링되지 않기 때문에 재 바인딩안됨

➤ 상태 데이터

- ref() 함수의 리턴값을 대입한 변수
- 상태 데이터는 값이 변경되면 컴포넌트가 리렌더링되어 재 바인딩됨
- ref() 함수의 매개값의 타입: 기본(string, number, Boolean), 객체, 배열 가능
 - 매개값이 객체일 경우 객체 자체를 교체할 수도 있고, 내부 속성도 변경 가능

```
<script setup>
  import { ref } from "vue";

  let no = ref(1);
  let name = ref("미니백");
  let company = ref("클레인");
  let detail = ref({
    info: "시그너츠 ...특징입니다.",
    sale: false
  });

  let price = ref(300000);
  function getPrice() {
    return price.value;
  }

```

```
function changeData() {
  no.value++;
  name.value = "빨간 미니백";
  company.value = "구찌";

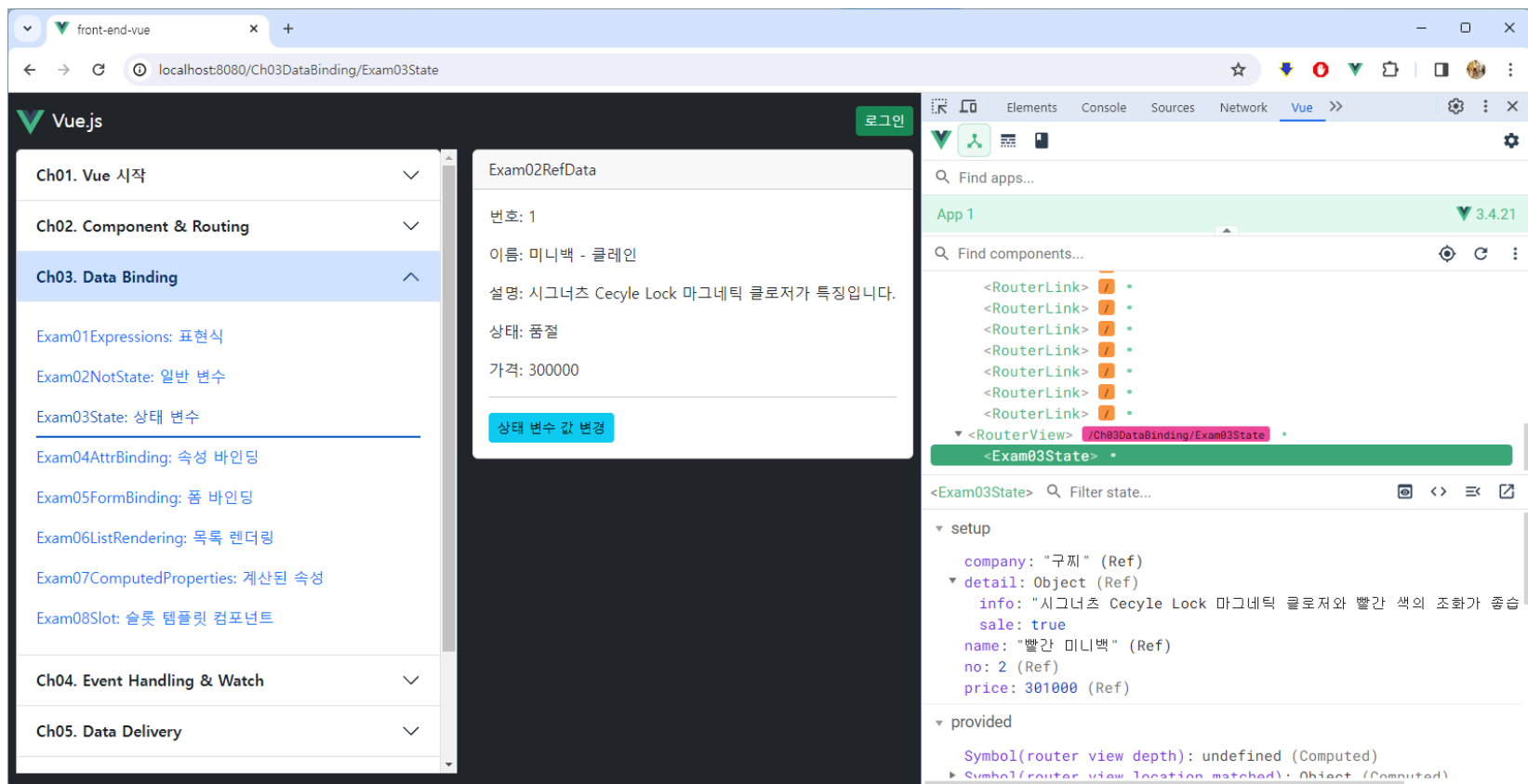
  //객체의 속성값 변경
  detail.value.info = "시그너츠 ...좋습니다."
  detail.value.sale = !detail.value.sale;

  //객체 자체를 교체
  /*detail.value = {
    info: "시그너츠 ...좋습니다.",
    sale: !detail.value.sale
  };*/

  price.value += 1000;
}
</script>
```


Data Binding

➤ Vue Devtools 에서 상태 데이터 변환 확인



Data Binding

❖ Computed 함수 바인딩

➤ `const computedFun = computed(() => { //데이터 가공... });`

- 데이터를 가공한 새로운 값을 생성해서 바인딩할 목적으로 사용
- 정의 형태는 **리턴값있는 함수**이지만, 사용은 상태 데이터처럼 사용
- 데이터가 변경되지 않으면 캐싱된 이전 리턴 값을 바인딩하므로 함수를 재실행하지 않음
- 데이터가 변경되면 자동으로 함수가 재실행되고 새로운 리턴 값으로 새로 캐싱함
- 일반 함수 호출은 컴포넌트가 리렌더링될 때마다 실행한다는 점에서 차이가 있음

```

<p>name: {{name}} </p>
<p>ssn: {{ssn1}}-{{ssn2.charAt(0) + "*****"}} </p>
<p>ssn: {{getSsn()}} </p>
<p>ssn: {{computedSsn}} </p>

```

```

<script setup>
import { ref, computed } from "vue";

//반응형 속성 선언
const name = ref("홍길동");
const ssn1 = ref("880515");
const ssn2 = ref("1110345");

```

```

//리렌더링될 때마다 실행
function getSsn() {
  console.log("getSsn() 실행");
  const ssn = ssn1.value + "-" +
    ssn2.value.charAt(0) + "*****";
  return ssn;
}

//리렌더링되더라도 ssn1과 ssn2가 변경이 없으면 실행하지 않음
const computedSsn = computed(() => {
  console.log("computedSsn() 실행");
  const ssn = ssn1.value + "-" +
    ssn2.value.charAt(0) + "*****";
  return ssn;
});
</script>

```

Data Binding

❖ 디렉티브를 이용한 바인딩

➤ v-bind:속성= “...” | :속성= “...”

- 속성값 바인딩은 표현식({{ ... }})을 사용할 수 없음
- 변수 | 객체({ ... }) | 배열([...]) | 연산식 | 매개변수화된 문자열(` ... \${변수명} ... `)

```
<div class="mb-4">
  <h5>:속성="변수"</h5>
  <a :href="vueHome" class="me-3">뷰 홈페이지</a>
  <RouterLink :to="appHome">홈 페이지</RouterLink>
</div>
```

```
<div class="mb-4">
  <h5>:속성="연산식" | `매개변수화된 문자열`</h5>
  
  
</div>
```

```
<div class="mb-4">
  <h5>:class="변수 | 객체 | 배열"</h5>
  <div class="mb-3 fw-bold">아름다운 풍경</div>
  <div class="mb-3" :class="className1">아름다운 풍경</div>
  <div class="mb-3" :class="[className1, className2]">아름다운 풍경</div>
  <div class="mb-3" :class="{ 'fw-bold':isBold, 'text-danger':isRed}">아름다운 풍경</div>
</div>
```

```
<div class="mb-4">
  <h5>:style="변수 | 객체 | 배열"</h5>
  <div style="margin-bottom: 10px; font-weight: bold;">아름다운 풍경</div>
  <div style="margin-bottom: 10px;" :style="[style1, style2]">아름다운 풍경</div>
  <div style="margin-bottom: 10px;" :style="{ 'font-weight':fontWeight, 'color':textColor}">아름다운 풍경</div>
</div>
```

Data Binding

➤ v-html= “변수”

- 내부 HTML이 변수에 저장되어 있을 경우 사용

```
<template>
  <div class="card">
    <div class="card-header">Exam06InnerHTMLBinding</div>
    <div class="card-body">
      <div v-html="innerHTML"></div>
      <hr/>
      <button @click="changeData" class="btn btn-info btn-sm">내용 변경</button>
    </div>
  </div>
</template>

<script setup>
import { ref } from 'vue';

let innerHtml = ref("");

let toggle = ref(true);
function changeData() {
  if(toggle.value) {
    innerHtml.value = `
      <div>
        <h5>풍경1</h5>
        
      </div>
    `;
  } else {
    innerHtml.value = `
      <div>
        <h5>풍경2</h5>
        
      </div>
    `;
  }
  toggle.value = !toggle.value;
}
</script>
```

Data Binding

➤ v-if= “변수” , v-show= “변수”

- 보임 여부를 결정
- 변수는 Boolean 타입의 값을 가짐
- v-if는 DOM 추가 여부 결정
- v-show는 DOM에 추가하되 보여줄지 여부를 결정

```
<template>
  <div class="card">
    <div class="card-header">Exam07IfShowBinding</div>
    <div class="card-body">
      <div id="div1" v-if="isPhoto" class="mb-2">
        
      </div>

      <div id="div2" v-show="isPhoto">
        
      </div>

      <hr/>

      <button @click="changeData" class="btn btn-info btn-sm">상태 변수값 변경</button>
    </div>
  </div>
</template>

<script setup>
import {ref} from "vue";

let isPhoto = ref(true);

function changeData() {
  isPhoto.value = !isPhoto.value;
}
</script>

<style scoped>
</style>
```

Data Binding

➤ 반복 바인딩

- <태그 v-for= “(item, index) in 배열” :key= “항목식별값|index” > ... </태그>
- <태그 v-for= “(value, name, index) in 객체” :key= “항목식별값|index” > ... </태그>
- <태그 v-for= “n in 10” :key= “n” > ... </태그>
n은 1부터 시작

```
<div>
  <h6>[범위 반복]</h6>
  <span v-for="n in 3" :key="n" class="mr-2">
    
  </span>
</div>
```

```
<div class="mt-3">
  <h6>[배열 항목 반복]</h6>
  <span v-for="(photo, index) in photos" :key="index" class="mr-2">
    
  </span>
</div>
```

```
<tr v-for="(board) in boards" :key="board.bno">
  <td>{{board.bno}}</td>
  <td>{{board.btitle}}</td>
  <td>{{board.bwriter}}</td>
  <td>{{board.bdate}}</td>
</tr>
```

```
<div class="mt-3">
  <h6>[객체 속성 반복]</h6>
  <div v-for="(value, name, index) in board" :key="index">
    <p>({{index}}) {{name}}: {{value}}</p>
  </div>
</div>
```

```
<script setup>
  import { ref } from "vue";

  const photos = ref(["photo1.jpg", "photo2.jpg", "photo3.jpg"]);

  const boards = ref([
    {bno:1, btitle:"제목1", bwriter:"글쓴이1", bdate:"2021-08-07"},
    {bno:2, btitle:"제목2", bwriter:"글쓴이2", bdate:"2021-08-08"},
    {bno:3, btitle:"제목3", bwriter:"글쓴이3", bdate:"2021-08-09"}
  ]);

  const board = ref({
    bno:4,
    btitle: "제목4",
    bcontent: "내용4",
    bwriter: "글쓴이4",
    bdate: "2021-08-10"
  });
</script>
```

Data Binding

➤ Form 바인딩

- **v-on:submit.prevent** 속성: 자동 제출 기능 중지
- **v-model** 속성: 상태 데이터와 양방향 바인딩

```
<form v-on:submit.prevent="handleSubmit">
  <input type="text" v-model="product.name">
  <input type="text" v-model="product.company">
  <input type="number" v-model="product.price">
  <input type="text" v-model="product.info">

  <select class="form-control" v-model="product.madein">
    <option value="한국">한국</option>
    <option value="미국">미국</option>
    <option value="독일">독일</option>
  </select>

  <input type="checkbox" value="black" v-model="product.colors">
  <input type="checkbox" value="red" v-model="product.colors">
  <input type="checkbox" value="yellow" v-model="product.colors">

  <input type="checkbox" value="black" value="true" v-model="product.sale1">
  <input type="checkbox" v-model="product.sale2" true-value="yes" false-value="no">

  <input class="form-check-input" type="radio" value="man" v-model="product.sex">
  <input class="form-check-input" type="radio" value="woman" v-model="product.sex">

  <input type="submit" value="등록" class="btn btn-danger btn-sm me-2" :disabled="!checkData"/>
  <!-- Vue에서 리셋 버튼은 양식을 초기화하지 않음 -->
  <!-- <input type="reset" value="재작성" class="btn btn-info btn-sm"/> -->
  <button type="button" class="btn btn-info btn-sm" @click="handleReset">재작성</button>
</form>
```

```
<script setup>
import { ref, computed, toRaw } from "vue";

let product = ref({
  name: "",
  company: "",
  price: 0,
  colors: ['yellow'],
  info: "",
  madein: "한국",
  sale1: true,
  sale2: "no",
  sex: "woman"
});

const checkData = computed(() => {
  var result = product.value.name !== "" &&
    product.value.company !== "";
  return result;
});
```

Data Binding

```
<script setup>
import { computed, ref } from 'vue';

let product = ref({
  name: "셔츠",
  company: "지오다노",
  price: 20000,
  info: "통풍이 잘되어 시원해요",
  madein: "미국",
  colors: ["black", "yellow"],
  sale1: true,
  sale2: "yes",
  sex: "woman"
});

function handleSubmit() {
  console.log(product.value);
  //반응성이 제거된 일반 자바스크립트 객체로 변환
  console.log(structuredClone(product.value));
}

const checkData = computed(() => {
  var result = product.value.name !== "" && product.value.company !== "";
  return result;
});

function handleReset() {
  product.value = {
    name: "셔츠",
    company: "지오다노",
    price: 20000,
    info: "통풍이 잘되어 시원해요",
    madein: "미국",
    colors: ["black", "yellow"],
    sale1: true,
    sale2: "yes",
    sex: "woman"
  };
}
</script>
```


Ch04. Event Handling & Watch

한국소프트웨어산업협회
신용권



Event Handling & Watch

❖ 이벤트 처리 및 감시

➤ 요소 이벤트 처리

- **v-on:이벤트** = “실행문 | 메소드 | 메소드(arg1, ..., \$event)”
 @이벤트 메소드 참조 메소드 호출 DOM 이벤트 참조

```
<button v-on:click= “handleBtnOk” >OK</button>
```

```
<button @click= “handleBtnCancel($event)” >Cancel</button>
```

```
function handleBtnOk() { ... },  
function handleBtnCancel(event) { ... }
```

event.preventDefault() 효과

- 수식어(Modifiers)

- <form @submit.prevent=“handleSubmit”>...</form>
- <a @click.prevent=“handleClick”>...
- <div @click.ctrl=“doSomething”></div>
- <input @keyup.enter=“handleSubmit”>...</input>
- <input @keyup.alt.67=“handleClear”>...</input>

//form의 기본 submit 기능 취소

//a의 기본 이동 기능 취소

//Ctrl + Click 할 경우

//Enter키를 누른 경우

//Alt + C 를 누른 경우

Event Handling & Watch

➤ 상태 데이터 변경 감시

● 상태 데이터

```
import { ref, watch } from 'vue';

//상태 생성
let userId = ref("");

let product = ref({
  name: "제네시스",
  company: "현대",
  price: 80000000
});
```

● 감시 상태가 값일 경우

```
watch(userId, (newUserId, oldUserId) => {
  console.group("값 상태 감시");
  console.log("newUserId: " + newUserId);
  console.log("oldUserId: " + oldUserId);
  console.groupEnd();
});
```

● 감시 상태가 객체일 경우

- 기본적으로 객체 참조가 변경되었을 경우에만 감시
 - » new와 old의 참조 객체는 다름

```
watch(product, (newProduct, oldProduct) => {
  console.group("객체 참조 변경 감시");
  console.log("newProduct:", structuredClone(newProduct));
  console.log("oldProduct:", structuredClone(oldProduct));
  console.groupEnd();
});
```

Event Handling & Watch

- 속성까지 감시하려면 {deep: true} 옵션 추가
 - » 속성만 변경되었을 경우: new와 old의 참조 객체는 같음

```
watch(product, (newProduct, oldProduct) => {  
  console.group("객체 참조 뿐만 아니라 속성까지도 변경 감시");  
  console.log("newProduct:", structuredClone(newProduct));  
  console.log("oldProduct:", structuredClone(oldProduct));  
  console.groupEnd();  
}, { deep: true });
```

- 특정 속성만 변경 감시

```
watch(() => product.value.name, (newName, oldName) => {  
  console.group("product 객체의 name 속성 변경 감시")  
  console.log("newName:", newName);  
  console.log("oldName:", oldName);  
  console.groupEnd();  
});
```

- 감시 상태가 복수개인 경우

```
watch([userId, product], ([newUserId, newProduct], [oldUserId, oldProduct]) => {  
  console.group("userId와 product 멀티 변경 감시")  
  console.log("new:", [newUserId, JSON.parse(JSON.stringify(newProduct))]);  
  console.log("old:", [oldUserId, JSON.parse(JSON.stringify(oldProduct))]);  
  console.groupEnd();  
}, { deep: true });
```

- 감시 상태가 복수개인 경우

```
watch([userId, product], ([newUserId, newProduct], [oldUserId, oldProduct]) => {  
  console.group("userId와 product 멀티 변경 감시")  
  console.log("new:", [newUserId, JSON.parse(JSON.stringify(newProduct))]);  
  console.log("old:", [oldUserId, JSON.parse(JSON.stringify(oldProduct))]);  
  console.groupEnd();  
}, { deep: true });
```

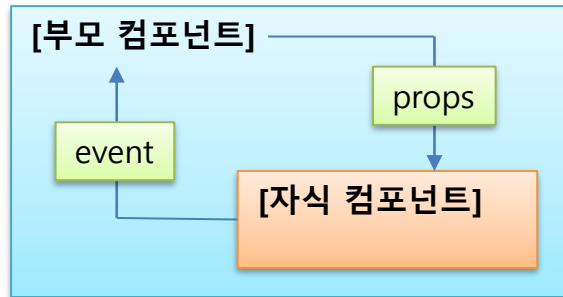
Ch05. 컴포넌트간 데이터 전달

한국소프트웨어산업협회
신용권



❖ 컴포넌트간 데이터 전달

- 부모 컴포넌트와 자식 컴포넌트간의 데이터 전달
 - 부모는 자식에게 props로 데이터 전달
 - 자식은 자신에게 일어난 일을 부모에게 event로 알림
 - props는 아래로, event는 위로

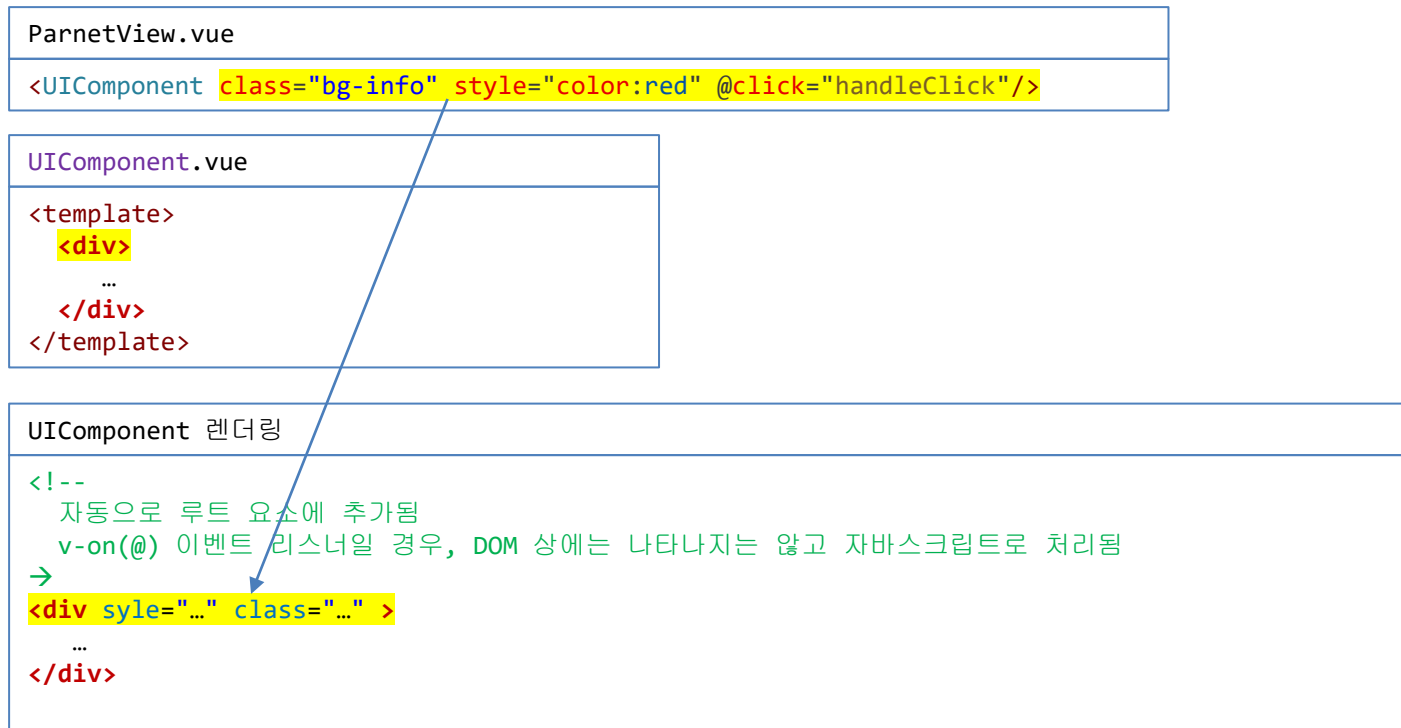


- 부모는 자식에서 발생한 이벤트를 처리
 - 부모 데이터 갱신 작업
 - 필요하면 props로 다시 자식으로 데이터 전달

❖ Fallthrough

➤ 컴포넌트의 속성 전달

- <https://vuejs.org/guide/components/attrs.html#fallthrough-attributes>
- 컴포넌트에 작성된 모든 속성은 기본적으로
- <template> 루트 요소의 속성으로 자동으로 추가됨



Data Binding

❖ defineProps

- `defineProps(["속성명", "속성명", ...])`로 Fallthrough에서 제외
- 데이터 전달 방향: 부모 -> 자식
 - 태그 속성 이름이 kebab-case 일 경우에는 camelCase로 변경

[Parent 컴포넌트]

```
<child productNo="5" product-kind="bag"></child>
```

[Child 컴포넌트]

```
import { defineProps } from 'vue';
```

```
const props = defineProps([ "productNo", "productKind" ]);
```

[템플릿]

```
props.productNo
```

```
$props.productKind
```

[자바스크립트]

```
props.productNo
```

- 숫자, 객체, 배열, 함수를 전달하고자 할 경우

- props로 전달되는 숫자는 기본적으로 문자열로 전달됨

```
<child attr="1"></child> // "1" 로 전달
```

- JavaScript 표현식으로 평가되도록 `v-bind(:)`를 사용해서 전달

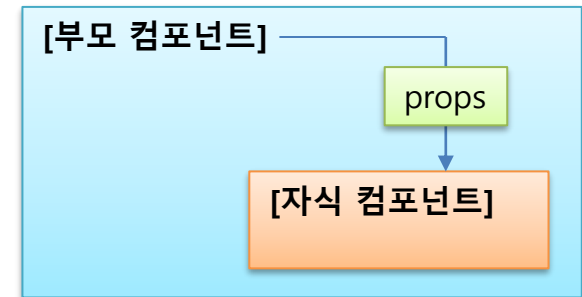
```
<child v-bind:attr="1"></child> //1 로 전달
```

```
<child :attr="{...}"></child>
```

```
<child :attr="[...]"></child>
```

```
<child :attr="() => {...}"></child>
```

```
<child v-bind={...}></child> //객체의 속성이름으로 개별적으로 분해해서 전달
```



컴포넌트간 데이터 전달

❖ defineEmits

- `defineEmits(["이벤트명", "이벤트명", ...])`로 Fallthrough에서 제외
- 데이터 전달 방향: 부모 ← 자식

- 부모 컴포넌트가 이벤트명과 핸들러 전달

[Parent 컴포넌트]

```
<child v-on:이벤트명 = "핸들러" ></child>
```

```
<child @이벤트명 = "핸들러" ></child>
```

- 자식 컴포넌트는 받은 이벤트명으로 이벤트 발생

[Child 컴포넌트]

```
import { defineEmits } from "vue";
```

```
const emit = defineEmits(["이벤트명", "이벤트명", ...]);
```

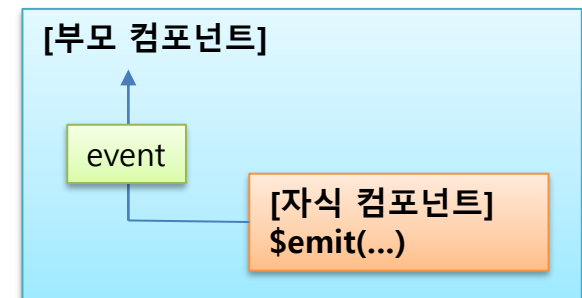
[템플릿]

```
<button @click="emit("이벤트명", 전달데이터, ...)">
```

```
<button @click="$emit("이벤트명", 전달데이터, ...)">
```

[자바스크립트]

```
emit("이벤트명", 전달데이터, ...);
```



Data Binding

❖ useAttrs

- Fallthrough 비활성화후, useAttrs를 이용해서 개별 속성 접근
 - defineProps()과 defineEmits() 대체 방법으로 사용

```
<UIComponent class="bg-info" style="color:red"
  :attr1="1" :attr2="{name:'홍길동', age:25}" :attr3="['블랙', '레드']" :attr4="()" => '리턴값'
  @click="handleClick"
  @customEvent="handleCustomEvent"/>
```

```
<script setup>
import { defineOptions, useAttrs } from "vue";
```

```
defineOptions({
  //Fallthrough를 비활성화함
  inheritAttrs: false
});
```

```
const attrs = useAttrs();
```

```
//속성값 읽기
console.group("UIComponent");
console.log("class: ", attrs.class);
console.log("style: ", attrs.style);
console.log("attr1: ", attrs.attr1, typeof(attrs.attr1));
console.log("attr2: ", attrs.attr2);
console.log("attr3: ", attrs.attr3);
console.log("attr4: ", attrs.attr4());
console.groupEnd();
```

```
function handle1() {
  attrs.onClick();
}
```

```
function handle2() {
  attrs.onCustomEvent("data1", "data2");
}
</script>
```

```
<p :class="$attrs.class">배경색 적용</p>
<hr/>
```

```
<p :style="$attrs.style">글자색 적용</p>
<hr/>
```

```
<p>attr1: {{ $attrs.attr1 }}</p>
<hr/>
```

```
<p>attr2: name={{ $attrs.attr2.name}}, age={{ $attrs.attr2.age}}</p>
<hr/>
```

```
<p>attr3:
  <span v-for="(item, index) in $attrs.attr3" :key="index">
    {{item}},
  </span>
</p>
<hr/>
```

```
<p>attr4: {{ $attrs.attr4()}}</p>
<hr/>
```

```
<button class="btn btn-danger btn-sm me-2"
  @click="$attrs.onClick">$attrs.onClick 이용</button>
<button class="btn btn-danger btn-sm me-2"
  @click="$attrs.onCustomEvent('data1', 'data2')">
  $attrs.onCustomEvent 이용</button>
```

Data Binding

❖ Slot

➤ 슬롯 용도

- 컴포넌트들이 UI 구조가 동일하고 내용만 다른 경우가 있음
- 동일한 부분을 템플릿 컴포넌트로 작성하고 내용이 변경되는 부분에 `<slot>`으로 표기
- 각 컴포넌트는 템플릿 컴포넌트를 배치할 때 `<slot>`에 들어갈 내용만 작성

➤ 템플릿 컴포넌트 작성

- 컴포넌트마다 바뀌는 내용 부분은 `<slot>`으로 지정

```
<div class="modal-header">
  <h5 class="modal-title">
    <!-- slot -->
    <slot name="header">제목</slot>
  </h5>
  <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
</div>
<div class="modal-body">
  <!-- slot -->
  <slot name="body"><p>다이얼로그 내용</p></slot>
</div>
<div class="modal-footer">
  <slot name="footer">
    <!-- slot -->
    <button type="button" class="btn btn-info btn-sm" data-bs-dismiss="modal">확인</button>
    <button type="button" class="btn btn-secondary btn-sm" data-bs-dismiss="modal">닫기</button>
  </slot>
</div>
```

개별컴포넌트에서
대체할 부분

➤ 템플릿 컴포넌트 배치

- 템플릿의 <slot> 자리에 들어갈 내용 정의

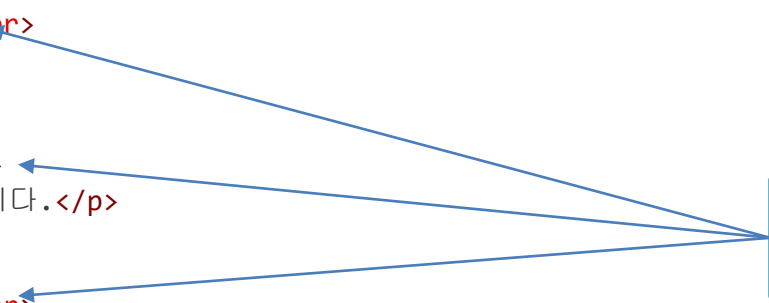
```
<template>
  <DialogTemplate>
    <template v-slot:header>
      알림
    </template>

    <template v-slot:body>
      <p>DialogB 내용입니다.</p>
    </template>

    <template v-slot:footer>
      <button class="btn btn-warning btn-sm" @click="emit('close')">확인</button>
    </template>
  </DialogTemplate>
</template>

<script setup>
import DialogTemplate from "../DialogTemplate.vue";
import { defineEmits } from "vue";

const emit = defineEmits(["close"]);
</script>
```



템플릿 컴포넌트의
슬롯에 끼워넣을
내용

- 템플릿의 <slot> 자리에 들어갈 내용 정의

```
<template>
  <DialogTemplate>
    <template v-slot:header>
      로그인
    </template>

    <template v-slot:body>
      <div class="mb-3">
        <label for="inputEmail" class="form-label">이메일</label>
        <input type="text" class="form-control" id="inputEmail" v-model="inputData.email">
      </div>
      <div class="mb-3">
        <label for="inputPassword" class="form-label">비밀번호</label>
        <input type="password" class="form-control" id="inputPassword" v-model="inputData.password">
      </div>
    </template>

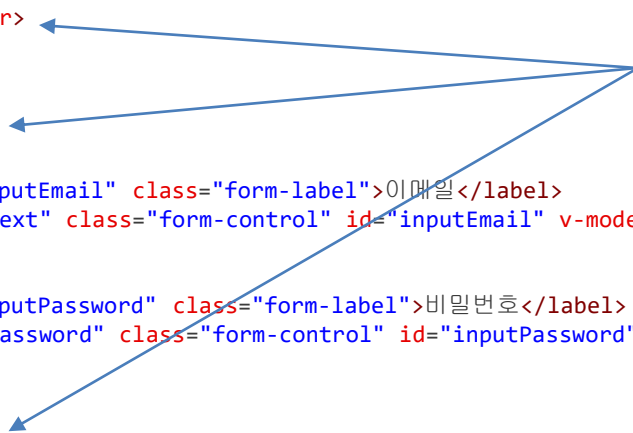
    <template v-slot:footer>
      <button class="btn btn-warning btn-sm me-2" @click="handleLogin">로그인</button>
      <button type="button" class="btn btn-secondary btn-sm" data-bs-dismiss="modal">닫기</button>
    </template>
  </DialogTemplate>
</template>

<script setup>
import { defineEmits, ref } from "vue";
import DialogTemplate from "../DialogTemplate.vue";

const emit = defineEmits(["close"]);

const inputData = ref({ email: "", password: "" });

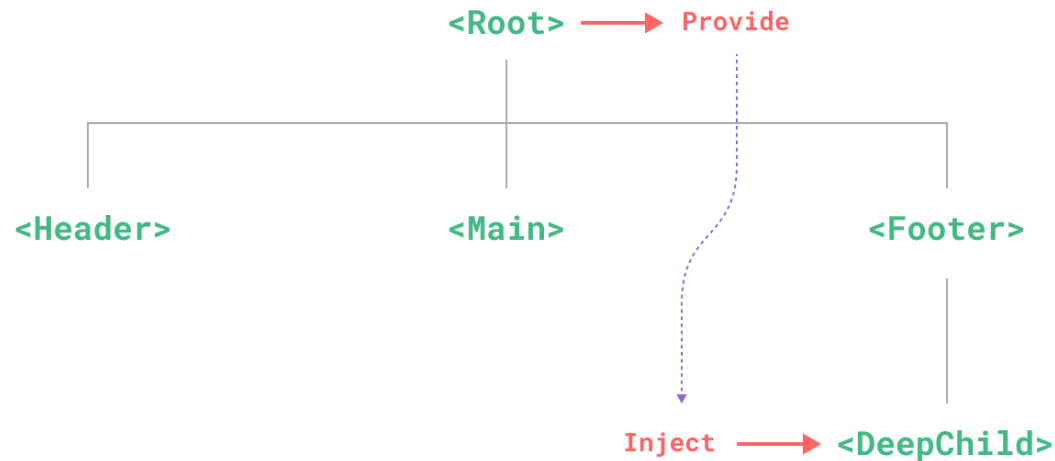
function handleLogin() {
  console.log(structuredClone(inputData.value));
  emit('close');
}
</script>
```



템플릿 컴포넌트의 슬롯에 끼워넣을 내용

❖ Provide / Inject

- Provide: 조상이 모든 자손 컴포넌트에게 데이터를 제공
- Inject: 자손은 이용하고자하는 데이터를 주입(변경 가능)



- 자손 컴포넌트마다 props으로 데이터 전달하는 것을 없애줌
- 자손 컴포넌트에서 데이터 변경이 가능

- Provide

```
<script setup>
import { ref, provide } from "vue";

//상태 생성
const name = ref("삼성전자");
const stock = ref({
  ticker: "005930",
  currentPrice: 150000,
  tradingVolume: 1250000
});

//하위 컴포넌트로 데이터를 제공
provide("message", {name: name, stock: stock});

//데이터 변경 함수
function changeData() {
  stock.value.currentPrice = 200000;
  stock.value.tradingVolume = 1350000;
}
</script>
```


- Inject

```
<template>
  <div class="card">
    <div class="card-header">UIComponentB</div>
    <div class="card-body">
      <div>
        <p>종목: {{ message.name.value }}</p>
        <p>현재가: {{ message.stock.value.currentPrice }}</p>
        <p>거래량: {{ message.stock.value.tradingVolume }}</p>
        <button @click="changeData">데이터 변경</button>
      </div>
    </div>
  </div>
</template>

<script setup>
import { inject } from "vue";

const message = inject("message");

//데이터 변경 함수
function changeData() {
  message.stock.value.currentPrice = 250000;
  message.stock.value.tradingVolume = 1700000;
}
</script>
```

Ch06. Lifecycle Hook

한국소프트웨어산업협회
신용권



Lifecycle Hooks

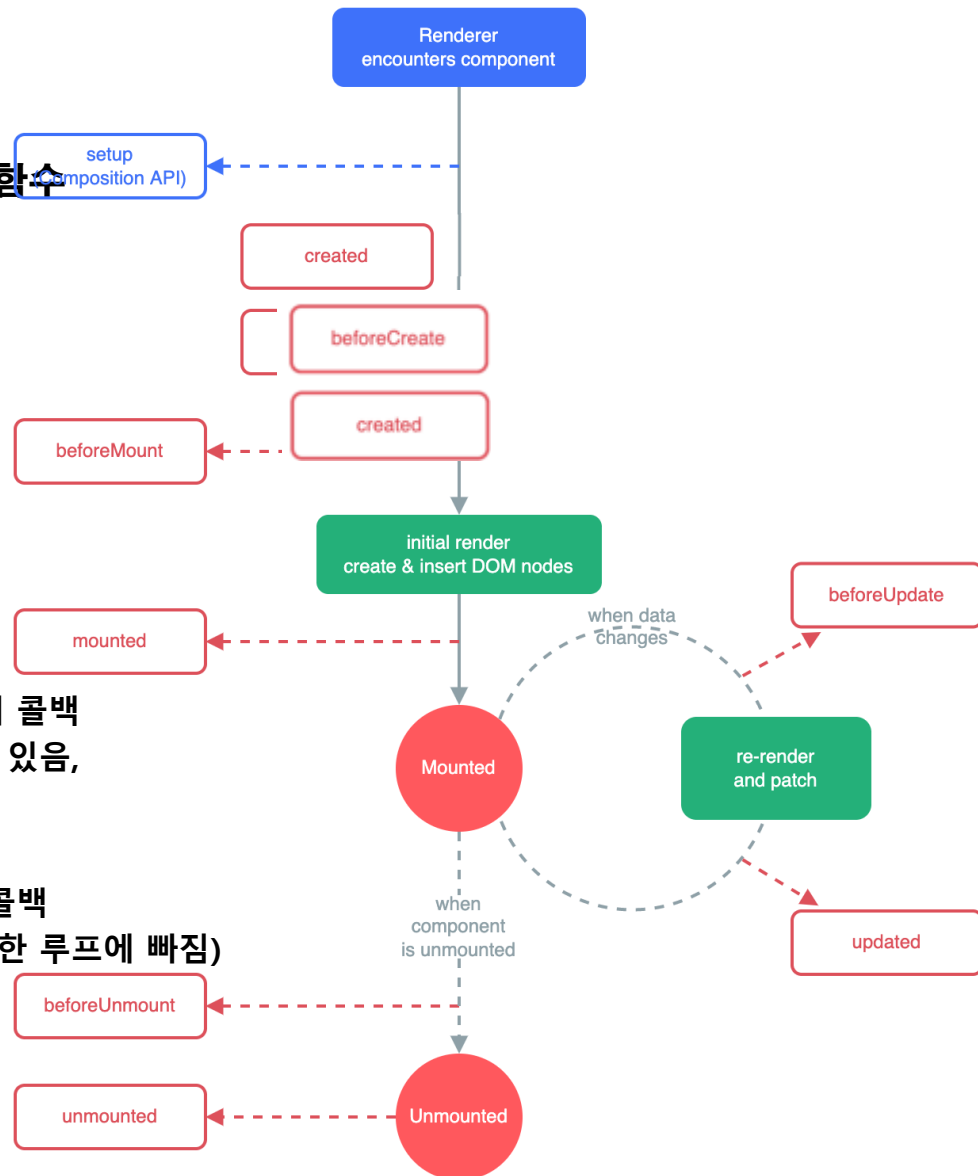
❖ Lifecycle Hook

➤ Lifecycle Hook 이란

- 컴포넌트의 상태에 따라 콜백되는 함수
- Lifecycle Hook를 재정의해서 사용자 로직을 포함시킬 수 있음

➤ Lifecycle Hook 종류

- **onBeforeMount()**
 - DOM 생성 직전에 콜백
- **onMounted()**
 - DOM을 생성한 후 콜백
- **onBeforeUpdate()**
 - 컴포넌트의 데이터가 변경되기 전에 콜백
 - 변경 예정인 새 데이터에 접근할 수 있음, 단 읽기만하고 변경해서는 안됨
- **onUpdated()**
 - 컴포넌트의 상태가 변경되고 나서 콜백
 - 다시 **상태 변경을 하지 말아야됨** (무한 루프에 빠짐)
- **onBeforeUnmount()**
 - 컴포넌트가 제거되기 직전에 실행
- **onUnmounted()**
 - 컴포넌트가 제거된 후 실행



➤ DOM 요소 참조

- 컴포넌트가 mounted된 이후만 가능
- 방법1: document.querySelector(...) 이용
- 방법2: 상태와 ref 속성을 이용

```
<p>data: {{ data }}</p>
<input id="input1" ref="input1Ref" type="text" style="line-height: 25px;"/>
<button class="btn btn-info btn-sm ms-2" @click="setData">입력 양식값 변경</button>

<script setup>
import { ref } from "vue";

//상태 생성
const data = ref("summer");
const input1Ref = ref(null);

function setData() {
  //방법1
  //const input1 = document.querySelector("#input1");
  //data.value = input1.value;

  //방법2
  data.value = input1Ref.value.value;
}
</script>
```

Ch07. Global State

한국소프트웨어산업협회
신용권



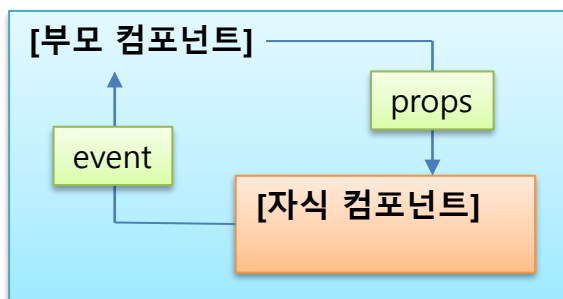
❖ 전역 상태(Global State)

➤ 전역 상태란

- 애플리케이션이 실행할 동안 지속성을 가지고, 컴포넌트간에 공유되는 데이터
- 컴포넌트가 생성할 때마다(라우팅마다) 선언되는 일회성 데이터는 전역 상태가 아님

➤ 전역 상태의 필요성

- Props, EventEmit 은 중첩된 컴포넌트가 많을 수록 데이터 흐름을 파악하기 어렵고, 코드가 복잡해짐



➤ Vuex

- Vue.js의 전역 상태 관리 기능을 제공하는 라이브러리
- 컴포넌트간의 공유 데이터를 중앙 집중식으로 관리
- 중앙 저장소 Store에서 데이터를 계층적으로 관리
- <https://vuex.vuejs.org/kr/>
 - V4.x: Vue 3 용
 - V3.x: Vue 2 용

❖ Vuex 설치

➤ 설치 방법

- 프로젝트 생성시 추가

? Check the features needed for your project

(*) Babel

() TypeScript

() Progressive Web App (PWA) Support

(*) Router

> (*) Vuex

() CSS Pre-processors

(*) Linter / Formatter

() Unit Testing

() E2E Testing

[Enter]

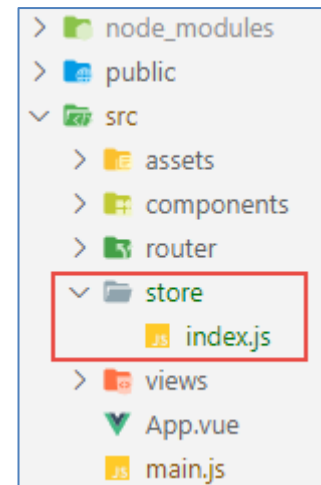
- 기존 프로젝트 추가

vue add vuex

➤ 추가된 내용

- src/store 폴더 생성

- main.js에 store 관련 코드 추가



```
import { createApp } from "vue";  
import App from "./App.vue";  
import router from "./router";  
import store from './store'
```

```
createApp(App)  
  .use(store)  
  .use(router)  
  .mount("#app");
```

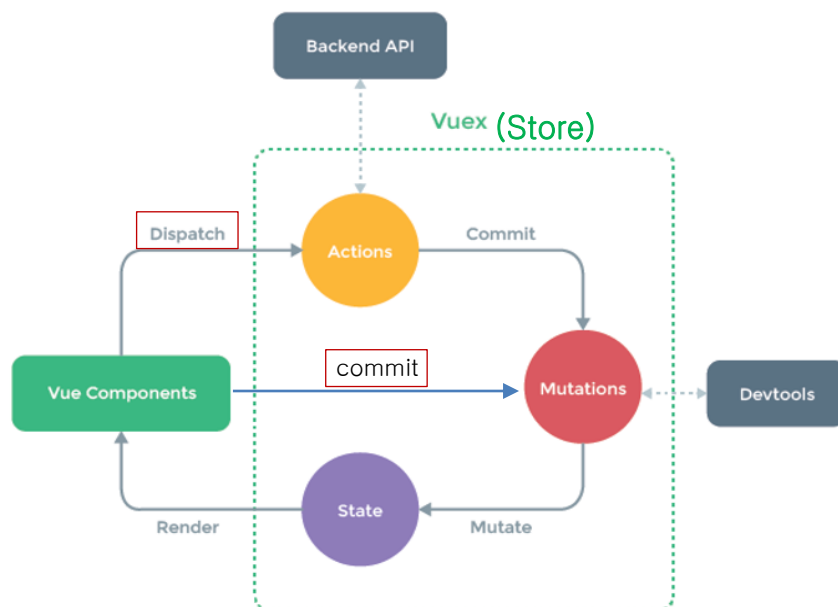
❖ Store 개념

➤ Store 란

- Store는 상태를 저장하고 있는 저장소
- Store에 저장된 상태가 변경되면 상태를 이용하는 컴포넌트는 리렌더링됨

➤ Store의 상태 변경 방법

- 상태를 직접 변경할 수 없음
- 방법1: 상태 추적이 가능하도록 변화(mutation)을 commit (동기)
- 방법2: 상태를 변경할 수 있는 액션(Action)을 dispatch (비동기)



❖ Store 루트모듈

➤ Store 루트 모듈(src/store/index.js)의 구성 이해

```
import { createStore } from 'vuex'

//Store 생성 및 내보내기
export default createStore({
  //루트 상태 정의
  state: {
  },

  //루트 상태 값을 읽는 메소드(Getter) 정의
  getters: {
  },

  //루트 상태 값을 변화시키는 메소드(Setter) 정의(동기 방식)
  mutations: {
  },

  //비동기 작업을 실행하고 결과에 따라 상태 값을 변화시키는 메소드 정의
  actions: {
  },

  //루트 하위 상태 모듈 추가
  modules: {
  }
})
```

➤ Store 루트 모듈 수정

● 상태 이름 정의

```
//Root State 정의
state: {
  user: "",
},
```

● 상태 값을 읽는 Getter 정의

```
//Root State 값을 읽는 메소드(Getter) 정의
getters: {
  getUser(state, getters, rootState, rootGetters) {
    /*
    console.group("RootState Getters/getUser");
    console.log("state:", state);           // root state 객체
    console.log("getters:", getters);        // root getters 객체
    console.log("rootState:", rootState);     // root stat 객체
    console.log("rootGetters:", rootGetters); // root getters 객체
    console.groupEnd();
    */
    return state.user;
  },
},
```

- **Mutation(Setter) 정의**
 - 동기방식으로 상태 값을 변경시키는 메소드

```
//Root State 값을 변경시키는 메소드(Setter) 정의 (동기 방식)
mutations: {
  setUser(state, payload) {
    /*
      console.group("RootState Mutations/setUser");
      console.log("state:", state);      // root state 객체
      console.log("payload:", payload); // 전달된 데이터
      console.groupEnd();
    */
    state.user = payload
  },
},
```

● Action 정의

- 작업을 수행하고 상태를 변화시키는 메소드
- 주로 Restful API로 비동기 요청하고, 응답을 받은 후에 상태 변이를 시키는 것을 목적으로 함.

//RootState를 변경하는 메소드 정의(비동기 방식)

```
actions: {  
  //payload: {mid:"xxxxx"}  
  loginAction(context, payload) {  
    new Promise((resolve, reject) => {  
      //서버와 통신 작업(?초 소요)  
      //로그인 성공  
      resolve({result:"success", mid:payload.mid});  
      //로그인 실패  
      //reject({result:"fail", reason:"password is wrong"});  
    })  
    .then((data) => {  
      console.log("로그인 성공");  
      context.commit("setUser", data.mid);  
    })  
    .catch((error) => {  
      console.log("로그인 실패");  
    });  
  },  
}
```

[매개변수 설명]

```
context: {  
  state,      // root state 객체  
  rootState, // root state 객체  
  getters,    // root getters 객체  
  rootGetters // root getters 객체  
  commit,    // store.commit 함수  
  dispatch,  // store.dispatch 함수  
},
```

payload: 전달된 데이터

➤ 컴포넌트에서 루트 상태 읽기

```
<h6>[RootState 읽기]</h6>
<p>user 단방향 바인딩: {{ $store.state.user }}</p>
<p>user 단방향 바인딩: {{ $store.getters.getUser }}</p>
<p>user 단방향 바인딩: {{ getUser() }}</p>
<p>user 단방향 바인딩: {{ computedUser }}</p>
<p>user 양방향 바인딩: <input type="text" v-model="$store.state.user"/></p>

<h6>[RootState 변경]</h6>
<button class="btn btn-info btn-sm me-2" @click="changeByMutation">user 변경(Mutation 동기 방식)</button>
<button class="btn btn-info btn-sm" @click="changeByAction">user 변경(Action 비동기 방식)</button>

<script setup>
import { useStore } from "vuex";
import { computed } from "vue";

const store = useStore();

function getUser() {
  //return store.state.user;
  return store.getters.getUser;
}

const computedUser = computed(() => {
  //return store.state.user;
  return store.getters.getUser;
});

function changeByMutation() {
  new Promise((resolve, reject) => {
    //서버와 통신 작업(?초 소요)
    //로그인 성공
    resolve({result:"success", mid:"user1"});
    //로그인 실패
    //reject({result:"fail", reason:"password is wrong"});
  })
  .then((data) => {
    console.log("로그인 성공");
    store.commit("setUser", data.mid);
  })
  .catch((error) => {
    console.log("로그인 실패");
  });
}

function changeByAction() {
  store.dispatch("loginAction", {mid:"user1"});
}
</script>
```

Global State

❖ 하위 상태 모듈

➤ 하위 상태 모듈 필요성

- 루트 상태 모듈에 상태가 많아 지면 복잡성 증가
- 기능별로 하위 상태 모듈로 분리하여 유지 보수 편리성 제공

➤ 하위 상태 모듈 파일 생성(src/store/counter.js)

```
export default {  
  //모듈의 이름을 접두사로 사용  
  namespaced: true,  
  //상태 정의  
  state: {  
    count: 0  
  },  
  //상태값을 읽는 메소드 정의  
  getters: {  
    getCount(state, getters, rootState, rootGetters) {  
      return state.count;  
    }  
  },  
  //상태값을 변경하는 메소드 정의(동기 방식)  
  mutations: {  
    setCount(state, payload) {  
      state.count += payload;  
    }  
  },  
  //상태값을 변경하는 메소드 정의(비동기 방식)  
  actions: {  
    addCount(context, payload) {  
      ...  
    }  
  }  
};
```

* 기본적으로 각각의 상태 모듈이 Store에 추가되면 선언된 action, mutation, getter는 Store의 전역 네임스페이스에 추가된다. 따라서 상태 모듈간의 action, mutation, getter 이름이 중복될 수 있는데, 이 경우 여러 모듈이 동일한 mutation 혹은 action에 반응할 수 있다.

ex) counter 상태 모듈의 "setCount"로 정의된 mutation 실행할 경우
store.commit("setCount", value);

* **namespaced: true**는 모듈의 이름을 루트 하위 네임스페이스로 설정한다. 이 설정은 특정 상태 모듈의 mutation, action, getter를 선택할 때 좋다.

ex) namespaced: true일 경우
store.commit("counter/setCount", value);

➤ 루트 하위 상태 모듈로 등록

- **src/store/index.js**

```
import { createStore } from 'vuex'

//counter 하위 상태 모듈 가져오기
import counter from "./counter";

//Store 생성 및 내보내기
export default createStore({
  ...

  //루트 하위 상태 모듈 추가
  modules: {
    counter
  }
})
```

- 액션 정의(actions) - 비동기

- 비동기 작업을 수행하고 상태를 변화시키는 메소드
- 주로 Restful API로 비동기 요청하고, 응답을 받은 후에 상태 변이를 시키는 것을 목적으로 함.

```
actions: {
  addCount(context, payload) {
    /*
    console.group("Counter Actions/addCount");
    console.log("context:", context);
    console.log("payload:", payload); //payload: 10
    console.groupEnd();
    */

    const work = async () => {
      try {
        const result = await new Promise((resolve, reject) => {
          //?초 소요
          //성공
          resolve({result:"success", amount:payload});
          //실패
          //reject({result:"fail", reason:"password is wrong"});
        });
        context.commit("setCount", result.amount);
      } catch(error) {
        console.log(error.reason);
      }
    };
    work();
  }
}
```

```
context: {
  state,      // 모듈 state 객체
  rootState, // root state 객체
  getters,    // 모듈 getters 객체
  rootGetters // root getters 객체
  commit,     // store.commit 함수
  dispatch,   // store.dispatch 함수
}
payload: 전달된 데이터
```


➤ 컴포넌트에서 하위 상태 읽기 및 변경

```
<h6>[counter 상태 읽기]</h6>
<p>count 단방향 바인딩: {{ $store.state.counter.count }}</p>
<p>count 단방향 바인딩: {{ $store.getters["counter/getCount"] }}</p>
<p>count 단방향 바인딩: {{ getCount() }}</p>
<p>count 단방향 바인딩: {{ computedCount }}</p>
<p>count 양방향 바인딩: <input type="text" v-model.number="$store.state.counter.count" /></p>

<hr />
```

입력된 값을 숫자로 바인딩

<https://vuejs.org/guide/essentials/forms.html#modifiers>
type을 number로 변경하면 .number는 필요 없음

```
<h6>[counter 상태 변경]</h6>
<button @click="changeByMutation">count 변경(Mutation 동기 방식)</button>
<button @click="changeByAction">count 변경(Action 비동기 방식)</button>
```

```
<script setup>
import { useStore } from "vuex";
import { computed } from "vue";

const store = useStore();

function getCount() {
  return store.state.counter.count;
  //return store.getters["counter/getCount"];
}

const computedCount = computed(() => {
  return store.state.counter.count;
  //return store.getters["counter/getCount"];
});
```

```
function changeByMutation() {
  store.commit("counter/setCount", 10);
}

function changeByAction() {
  store.dispatch("counter/addCount", 10);
}
</script>
```

Ch08. REST API 연동

한국소프트웨어산업협회
신용권



❖ REST API

➤ REST의 개념

- **RE**presentational **S**tate **T**ransfer
- HTTP 요청 방식 + URL 조합으로 자원 요청
 - HTTP 요청 방식: 수행할 작업을 식별
 - URL: 작업할 데이터를 식별

CRUD	HTTP	URI
전체 리소스 조회	GET	/resources
특정 리소스 조회	GET	/resources/:id
리소스 생성	POST	/resources
리소스 전체 수정	PUT	/resources/:id
리소스 일부 수정	PATCH	/resources/:id
특정 리소스 삭제	DELETE	/resources/:id

➤ REST API

- REST 요청을 처리하는 API (서버)
- 요청 및 응답 본문의 데이터 포맷은 JSON 또는 XML 사용

➤ REST API 프로젝트 실행

- VS Code 확장 설치
 - Extension Pack for Java
 - Spring Boot Extension Pack
- VS Code에서 back-end-all-api 프로젝트 열기
- back-end-all-api 프로젝트 실행
 - Spring Boot Dashboard 클릭
 - APPS/back-end-all-api 선택
 - 오른쪽에 삼각형 아이콘 클릭
 - 기본 80 포트로 실행됨
- PostMain에서 테스트
 - 왼쪽 상단 [Import] 버튼 클릭
 - postman/back-end-all-api.postman_collection.json 파일을 끌어서 놓음
 - [Import] 클릭

❖ Axios

➤ Axios란

- <https://axios-http.com/kr/docs/intro>
- 자바스크립트 HTTP 클라이언트, 모든 통신은 비동기 Promise 기반
- Fetch() 함수와 비교

구분	fetch() 함수	Axios
기본 라이브러리	자바스크립트 내장	서드파티 라이브러
사용 편리성	설정이 필요함 (Headers, JSON 변환)	간단한 문법 및 자동
응답 JSON 변환	아니요 (response.json() 호출 필요)	자동 변환(response.data)
에러처리	response.ok 체크	자동 에러 처리
인터셉터 지원	없음	있음
기본 헤더 설정	자동 설정 안됨	기본 헤더 설정 가능
파일 업로드/다운로드 진행률	내장 지원 없음	onUploadProgress, onDownloadProgress 지원

➤ Axios 모듈 설치

- `npm install axios`

➤ Axios API

- 요청 방식의 이름을 따서 메소드로 만듦
- 참고: https://axios-http.com/kr/docs/api_intro

```
axios.request(config)
```

```
axios.get(url[, config])
```

```
axios.delete(url[, config])
```

```
axios.head(url[, config])
```

```
axios.options(url[, config])
```

```
axios.post(url[, data[, config]])
```

```
axios.put(url[, data[, config]])
```

```
axios.patch(url[, data[, config]])
```

- **config 매개값:** 요청을 만드는데 사용할 수 있는 옵션 객체
 - 참고: https://axios-http.com/kr/docs/req_config
- 메소드의 리턴값은 Promise 임

❖ Promise 비동기 작업 처리

➤ 처리 방법

- Promise를 직접 이용

```
asyncWork()  
  .then(result => {...})  
  .catch(error => {...});  
  .finally(() => {...});
```

```
//비동기 작업 정의  
function asyncWork() {  
  //3초후에 응답이 오는 Promise 생성  
  const promise = new Promise((resolve, reject) => {  
    setTimeout(() => {  
      //성공적으로 처리했을 경우  
      resolve({ message: "성공" });  
      //실패 처리할 경우  
      //reject({ message: "실패" });  
    }, 3000);  
  });  
  return promise;  
}
```

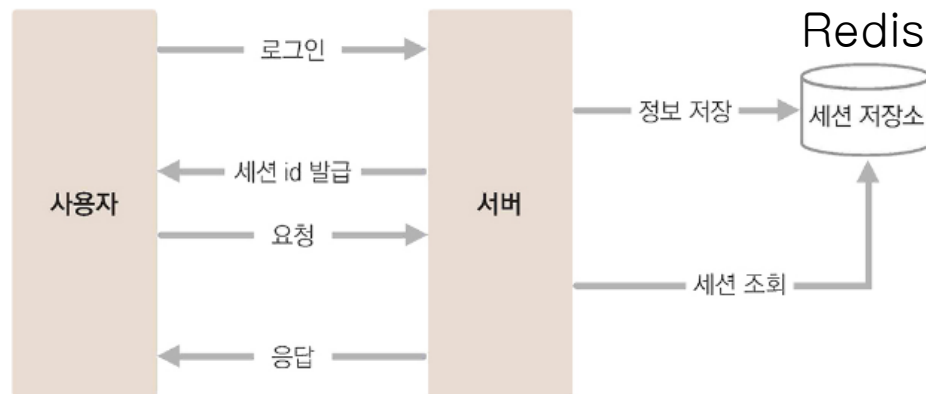
- async / await 이용

```
async () => {  
  try {  
    const result = await asyncWork();  
    ...  
  } catch(error) { ... }  
  finally { ... }  
}
```

❖ REST API 인증 시스템 이해

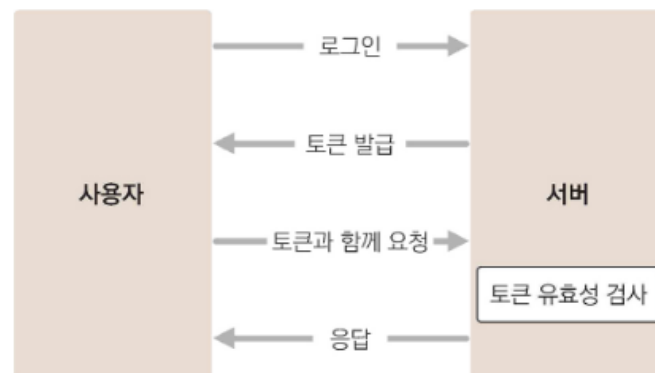
➤ 세션 기반 인증 시스템

- 서버에 인증 정보 저장
- 세션 id를 쿠키로 저장
- 요청시 쿠키로 세션 id 전달
- 멀티 도메인 세션 공유를 위한 세션 저장소 필요



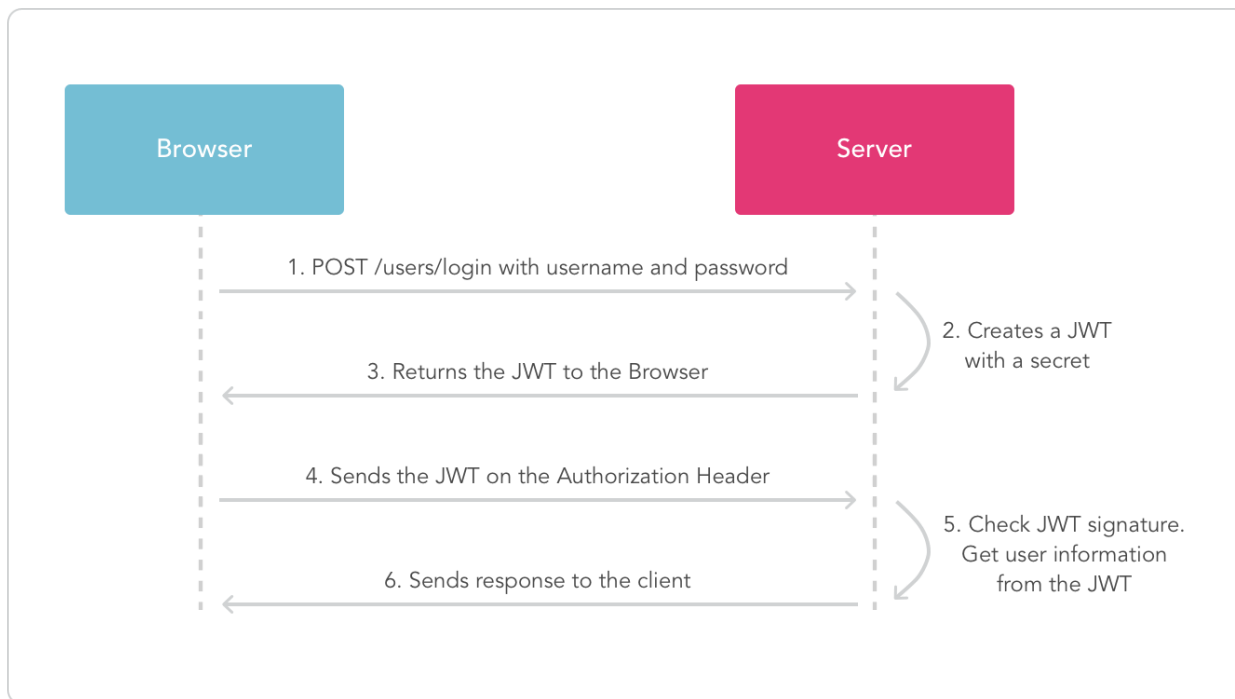
➤ 토큰 기반 인증 시스템

- 클라이언트에 토큰(인증 정보) 저장
- 토큰은 JWT를 주로 사용
- 멀티 도메인간 토큰 공유 쉬움
- 토큰 저장 방법
 - 쿠키(동일 도메인일 경우 편리)
 - 메모리/세션 스토리지(멀티 도메인에 유리)
- 토큰 전달 방법
 - 쿠키(동일 도메인일 경우 편리)
 - 요청 헤더(멀티 도메인에 유리)



➤ JWT(JsonWebToken) 인증

- 인증 받은 사용자에게 고유한 JWT를 발급
- 재요청시 Authorization 헤더에 토큰을 함께 보내어 인증된 사용자임을 밝힘



- 멀티 서버간 인증 정보 공유 쉬움: 통합 인증(Single Sign-On, SSO)에 유용
- MSA(MicroService Architecture) 인증 및 인가에 주로 사용
- 서버에 정보를 저장하지 않으므로 Stateless한 REST API에서 주로 사용

❖ JWT 인증 구현

➤ src/store/index.js 수정

● accessToken 전역 상태 추가

```
//Store 생성 및 내보내기
export default createStore({
  //RootState 정의
  state: {
    user: "",
    accessToken: ""
  },
  //RootState를 읽는 메소드 정의
  getters: {
    ...
    getAccessToken(state, getters, rootState, rootGetters) {
      return state.accessToken;
    }
  },
  //RootState를 변경하는 메소드 정의(동기 방식)
  mutations: {
    ...
    setAccessToken(state, payload) {
      state.accessToken = payload;
    }
  },
  //RootState를 변경하는 메소드 정의(비동기 방식)
  actions: {
    ...
  },
  //RootState 하위 상태 모듈 추가
  modules: {
    counter
  }
})
```

➤ Axios 기본 설정

● src/apis/axiosConfig.js

```
import axios from "axios";

//기본 경로 설정
axios.defaults.baseURL = "http://localhost";

//AccessToken을 받고나서 다음 요청시 전달할 수 있도록 요청 헤더에 추가
//로그인 성공했을 때 호출됨
function addAuthHeader(accessToken) {
  //common 객체에 동적 속성으로 Authorization을 추가
  axios.defaults.headers.common["Authorization"] = "Bearer " + accessToken;
  console.log(axios.defaults.headers.common);
}

//공통 요청 헤더에서 Authorization 헤더 삭제
//로그아웃시 호출됨
function removeAuthHeader() {
  //common 객체의 Authorization 속성을 삭제
  delete axios.defaults.headers.common["Authorization"];
  console.log(axios.defaults.headers.common);
}

//외부에서 사용할 수 있도록 내보내기
const obj = {
  addAuthHeader,
  removeAuthHeader
};
export default obj;
```

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9TjV...r7E20RMHrHDcEfxjoYZgeFONFh7HgQ
```

- **src/apis/memberAPI.js**

```
import axios from "axios";

//회원 가입
function join(member) {
  /*
    member = {
      mid: "user1",
      mname: "사용자1",
      mpassword: "Ab123",
      memail: "user1@naver.com"
    }
  */
  //POST: raw/json 방식으로 전달
  return axios.post("/member/join", member);
}

//로그인
function login(member) {
  /*
    member = {
      mid: "user",
      mpassword: "12345"
    }
  */
  //POST: raw/json 방식으로 전달
  return axios.post("/member/login", member);
}

const obj = {
  join,
  login
};
export default obj;
```

➤ src/store/index.js 수정

● action 함수 추가

```
//브라우저가 재실행될때 인증 정보를 전역상태로 복구
loadAuth(context, payload) {
  //user 전역 상태 설정
  context.commit("setUser", localStorage.getItem("user") || "");
  //accessToken 전역 상태 설정
  const accessToken = localStorage.getItem("accessToken") || "";
  context.commit("setAccessToken", accessToken);
  //Axios 요청 공통 헤더에 인증 정보 추가
  if(accessToken !== "") {
    axiosConfig.addAuthHeader(accessToken);
  }
},
```

```
//로그인 성공했을 때 인증 정보를 전역 상태 및 로컬 파일에 저장
saveAuth(context, payload) {
  /*
  payload = {
    mid: "user",
    accessToken: "xxxxx.yyyyy.zzzzz"
  }
  */
  //전역 상태값 변경
  context.commit("setUser", payload.mid);
  context.commit("setAccessToken", payload.accessToken);
  //로컬 스토리지에 저장
  localStorage.setItem("user", payload.mid);
  localStorage.setItem("accessToken", payload.accessToken);
  //Axios 요청 공통 헤더에 인증 정보 추가
  axiosConfig.addAuthHeader(payload.accessToken);
},
```

```
//로그아웃 할때 인증 정보를 모두 삭제
deleteAuth(context, payload) {
  //전역 상태값 변경
  context.commit("setUser", "");
  context.commit("setAccessToken", "");
  //로컬 파일에서 삭제
  localStorage.removeItem("user");
  localStorage.removeItem("accessToken");
  //Axios 요청 공통 헤더에 인증 정보 제거
  axiosConfig.removeAuthHeader();
}
```

➤ 브라우저 리프레쉬시 인증 정보 로딩을 위한 설정

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.min.js';
import store from './store'

// [Vue Router warn]이 출력되지 않도록 설정
const originalWarn = console.warn;
console.warn = function (msg, ...args) {
  if (msg.includes('[Vue Router warn]')) { return; }
  originalWarn(msg, ...args);
};

//브라우저 리프레쉬시 -----
//인증 정보를 store에 저장
store.dispatch("loadAuth");
//-----

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

➤ Exam03Login 컴포넌트

● template

```
<div v-if="$store.state.user === ''">
  <form @submit.prevent="handleLogin">
    <div class="input-group mb-2">
      <span class="input-group-text">아이디</span>
      <input type="text" class="form-control" v-model="member.mid">
    </div>

    <div class="input-group mb-3">
      <span class="input-group-text">비밀번호</span>
      <input type="password" class="form-control" v-model="member.mpassword">
    </div>

    <div class="d-flex justify-content-center">
      <input type="submit" value="로그인" class="btn btn-danger btn-sm me-2" />
      <!--Vue에서 리셋 버튼은 양식을 초기화하지 않음-->
      <!-- <input type="reset" value="재작성" class="btn btn-info btn-sm"/> -->
      <button type="button" class="btn btn-info btn-sm" @click="handleReset">재작성</button>
    </div>
  </form>
</div>

<div v-if="$store.state.user !== ''">
  <div class="d-flex justify-content-center">
    <button class="btn btn-success btn-sm" @click="handleLogout">로그아웃</button>
  </div>
</div>
```

● script

```
<script setup>
import { ref } from "vue";
import memberAPI from "@/apis/memberAPI";
import { useStore } from "vuex";

//store 객체 얻기
const store = useStore();

//상태 정의
const member = ref({
  mid: "",
  mpassword: ""
});

//로그인 버튼 클릭시 실행
async function handleLogin() {
  try {
    const data = structuredClone(member.value);
    const response = await memberAPI.login(data);
    if (response.data.result === "success") {
      const payload = {
        mid: response.data.mid,
        accessToken: response.data.accessToken
      };
      store.dispatch("saveAuth", payload);
    }
  } catch (error) {
    console.log(error);
  }
}
```

```
//재작성 버튼 클릭시 실행
function handleReset() {
  member.value.mid = "";
  member.value.mpassword = "";
}

//로그아웃 버튼 클릭시 실행
function handleLogout() {
  store.dispatch("deleteAuth");
}
</script>
```


➤ AppHeader 컴포넌트 수정

```
<template>
  <nav class="navbar justify-content-between bg-dark text-white">
    ...

    <!--
      <div class="me-2">
        <RouterLink to="/" class="btn btn-success btn-sm">로그인</RouterLink>
      </div>
    -->

    <div class="me-2">
      <div v-if="$store.state.user === ''">
        <RouterLink class="btn btn-success btn-sm" to="/Ch08RestAPI/Exam02Login">로그인</RouterLink>
      </div>
      <div v-if="$store.state.user !== ''">
        <span>UserID: {{ $store.state.user }}</span>
        <button class="btn btn-danger btn-sm ms-2" @click="handleLogout">로그아웃</button>
      </div>
    </div>
  </nav>
</template>

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

const store = useStore();
const router = useRouter();

function handleLogout() {
  store.dispatch("deleteAuth");
  router.push("/Ch08RestAPI/Exam02Login");
}
</script>
```

➤ Exam02Join 컴포넌트

● template

```
<form @submit.prevent="handleSubmit">
  <div class="input-group mb-2">
    <span class="input-group-text">아이디</span>
    <input type="text" class="form-control" v-model="member.mid">
  </div>

  <div class="input-group mb-2">
    <span class="input-group-text">이름</span>
    <input type="text" class="form-control" v-model="member.mname">
  </div>

  <div class="input-group mb-2">
    <span class="input-group-text">비밀번호</span>
    <input type="password" class="form-control" v-model="member.mpassword">
  </div>

  <div class="input-group mb-2">
    <span class="input-group-text">이메일</span>
    <input type="email" class="form-control" v-model="member.memail">
  </div>

  <div>
    <input type="submit" value="가입" class="btn btn-danger btn-sm me-2"/>
    <!--Vue에서 리셋 버튼은 양식을 초기화하지 않음-->
    <!-- <input type="reset" value="재작성" class="btn btn-info btn-sm"/> -->
    <button type="button" class="btn btn-info btn-sm" @click="handleReset">재작성</button>
  </div>
</form>
```

- script

```
import ...

//라우터 객체 얻기
const router = useRouter();

//상태 정의
const member = ref({
  mid: "",
  mname: "",
  mpassword: "",
  memail: ""
});

//가입 버튼 클릭시 실행
async function handleSubmit() {
  try {
    //Axios를 이용해서 Back-End로 회원 가입 요청
    const data = JSON.parse(JSON.stringify(member.value));
    const response = await memberAPI.join(data);
    console.log(response.data);
    //홈 페이지로 이동
    router.push("/");
    //이전 페이지로 이동
    //router.back();
  } catch(error) {
    console.log(error);
  }
}

//재작성 버튼 클릭시 실행
function handleReset() {
  ...
}
```

❖ CRUD 게시판 구현

➤ 실습

❖ 배포

➤ 라우터의 history 모드 해결 방법

- src/router/index.js에서 히스토리 모드로 설정은 개발시에만 적용됨

```
const router = createRouter({  
  history: createWebHistory(),  
  ...  
})
```

참고: <https://router.vuejs.org/guide/essentials/history-mode.html>

- dist 폴더의 빌드 결과물을 배포해서 실행할 때, 첫페이지가 아니라 다른 페이지에서
 - 브라우저 새로고침하면 404 Not Found 오류 발생
-
- 원인
 - history 모드로 사용할 경우, 기본적으로 브라우저는 서버에 해당 URL을 요청하게 됨.
 - 서버는 해당 경로의 페이지를 찾지 못해 404 오류를 반환함.
 - 이는 라우트를 클라이언트 사이드에서 관리하기 때문.

- 해결 방법(서버별 설정)

- NginX 설정(추천)
 - » 빌드 후에는 dist 폴더 내용을 nginx의 html 폴더에 복사
 - » nginx.conf 파일에서 다음과 같이 수정해야 함

```
location / {  
  #root html;  
  #index index.html index.htm;  
  try_files $uri $uri/ /index.html;  
}
```

****try_files****는 요청된 파일 또는 경로를 처리하기 위해 시도할 파일을 지정

1. \$uri: 요청된 URI에 해당하는 파일이 있는지 확인
예: /about.html.
2. \$uri/: 요청된 URI가 디렉토리인 경우, 해당 디렉토리에 대한 파일을 확인.
예: /about/
3. /index.html: 위의 조건이 충족되지 않을 경우,
/index.html 파일로 요청을 전달

➤ NGINX 설치

- <https://nginx.org/en/download.html>

Stable version

[CHANGES-1.26](#) [nginx-1.26.3](#) .pgp [nginx/Windows-1.26.3](#) .pgp

- nginx-1.26.3.zip 압축 풀기
- 프로젝트 빌드: npm run build
- nginx-1.26.3/html/에 프로젝트 dist 폴더에 있는 내용을 복사/붙여넣기
- nginx-1.26.3/conf/nginx.conf 파일 수정

listen 8090; #Rest API가 80을 사용하기 때문

```
location / {  
    #root html;  
    #index index.html index.htm;  
    try_files $uri $uri/ /index.html;  
}
```

- nginx-1.26.3/nginx.exe 실행
- 브라우저 테스트: <http://localhost:8090>
- nginx 종료: 백그라운드로 실행되므로 작업 관리자에서 프로세스 종료

수고 하셨습니다.