

삼성 청년 SW 아카데미

Vue.js

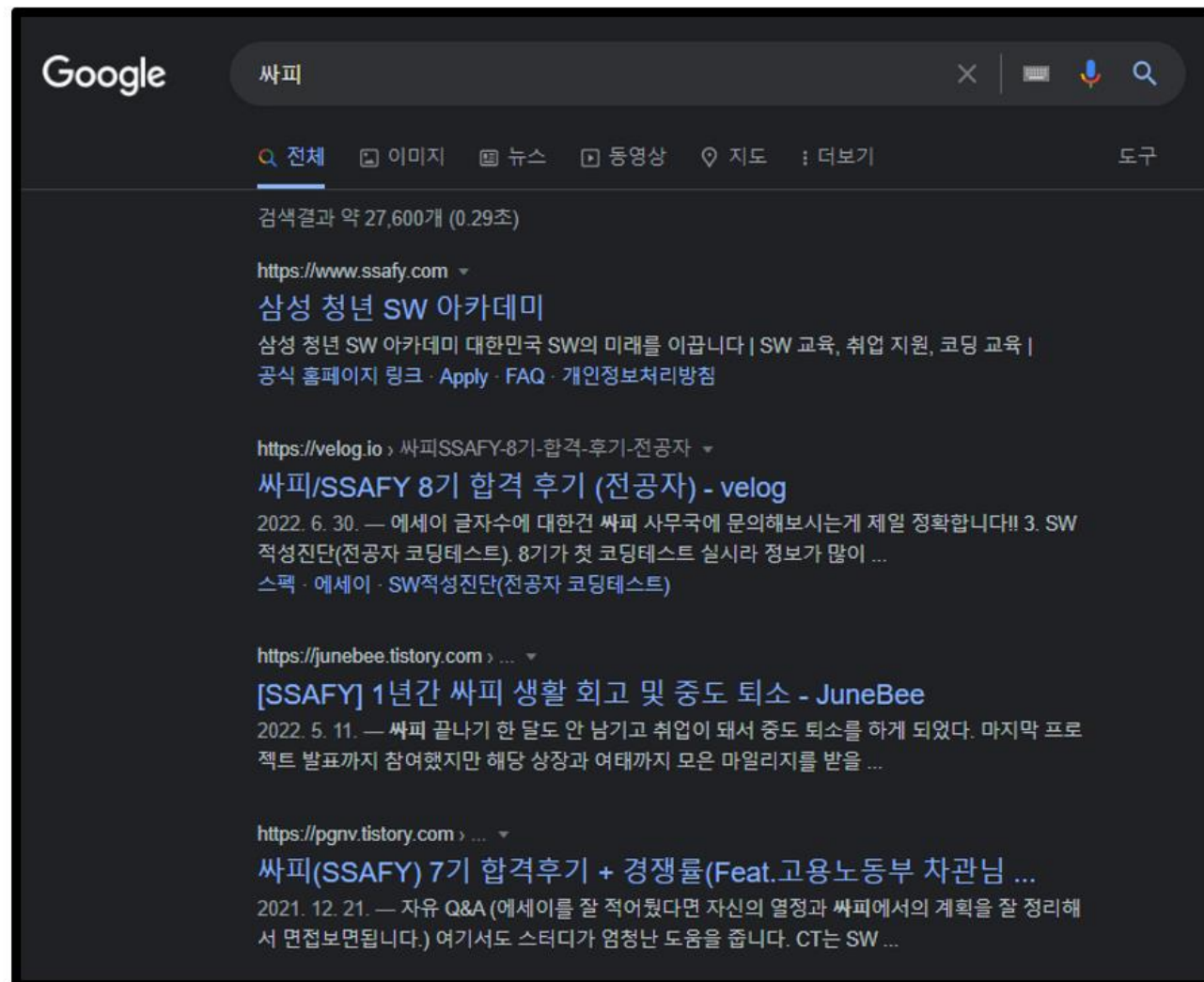
<알림>

본 강의는 삼성 청년 SW아카데미의 콘텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

컴포넌트

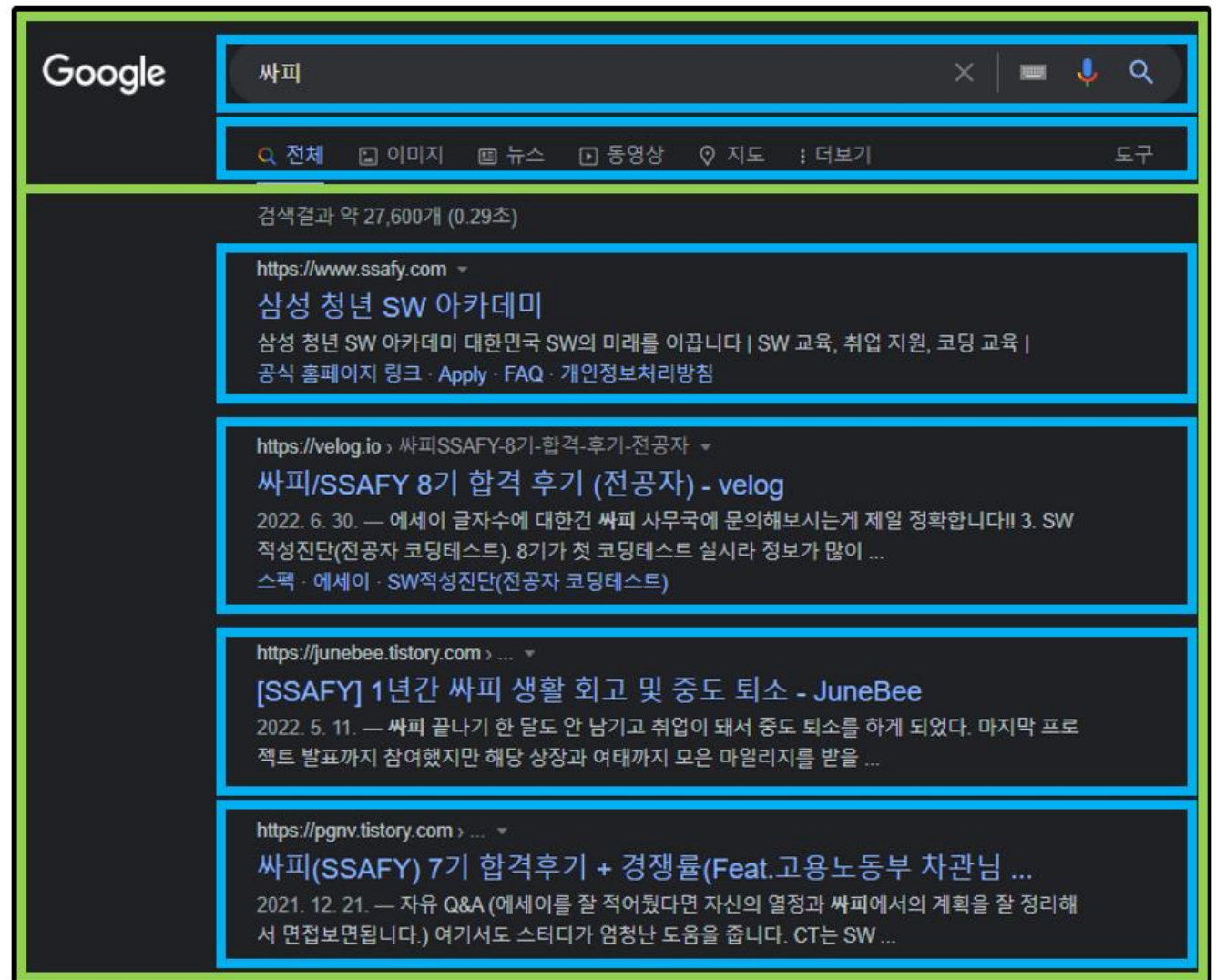
구글 검색화면의 예

- 하나의 페이지 - 하나의 HTML 문서
- 만약, 하나의 HTML 을 나눌 수 있다면?
 - 가독성 및 생산성 향상
 - 유지보수 원활



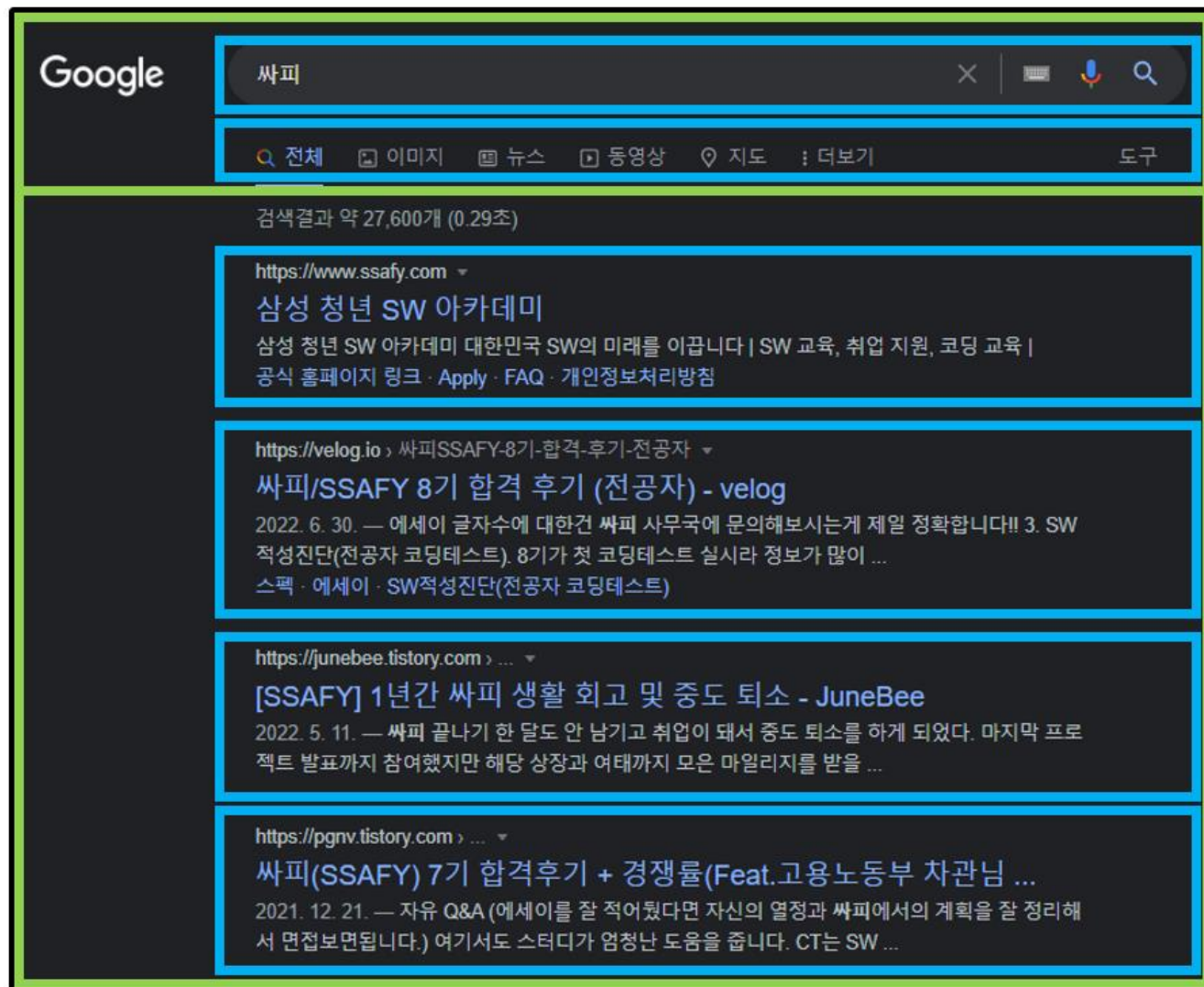
컴포넌트 단위로 나눔

- component: 부품
- 다음과 같이 나눈다고 가정
- Header
 - Search - 검색 창
 - Navigation - 메뉴 바
- Main
 - EachResult - 검색 결과
- 컴포넌트 안에 컴포넌트가 있을 경우,
부모 컴포넌트, 자식 컴포넌트로 구.
- 이 중, EachResult 는 v-for 를 사용하



컴포넌트 방식 개발

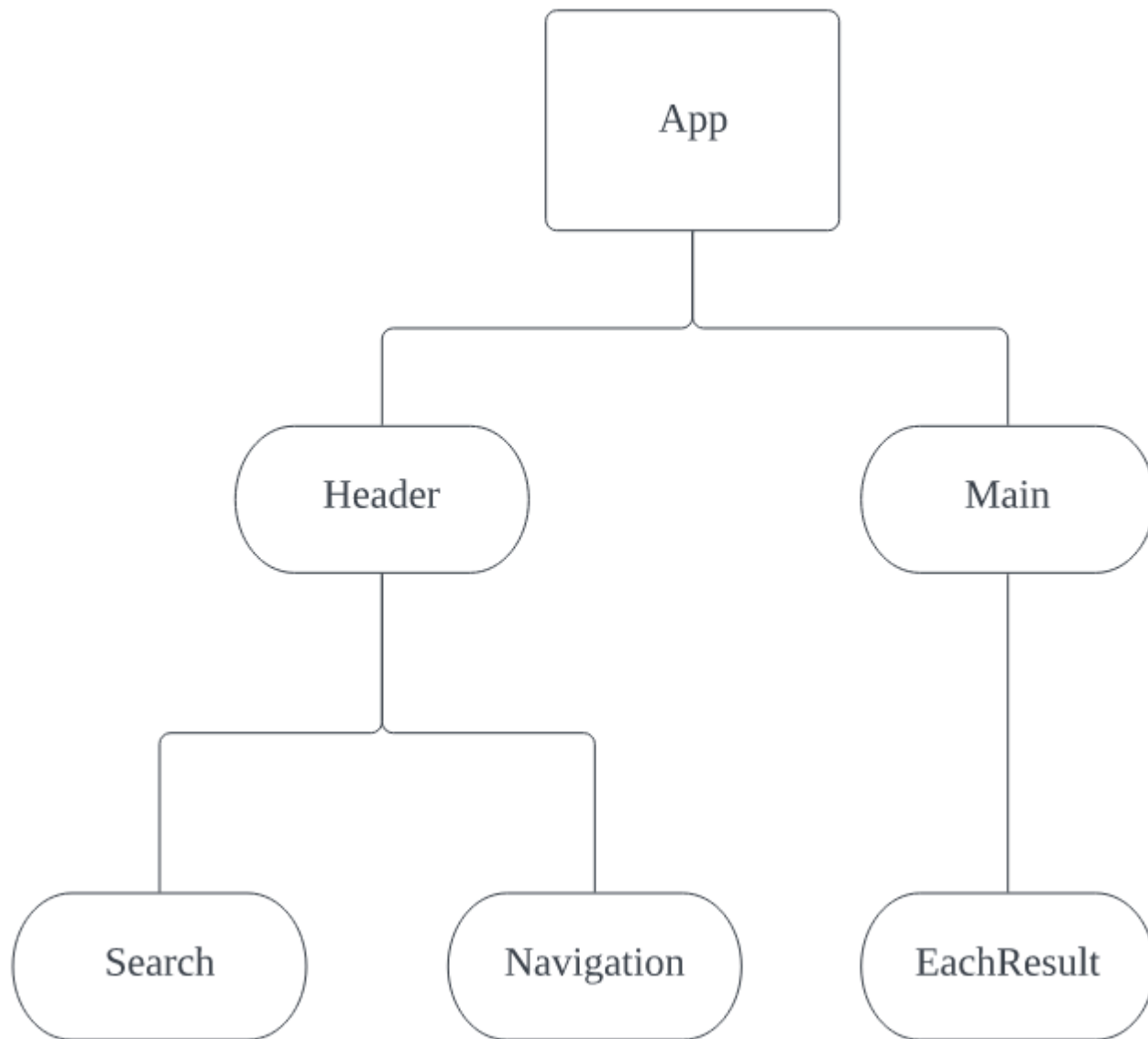
- 수십 개의 컴포넌트를 각각 개발
- 하나의 HTML 문서로 결합
- 각각의 컴포넌트는 필요에 따라 다른 컴포넌트에 재사용
- Vue CLI 사용해 개발시
 - 컴포넌트는 각각의 파일 단위 구분
 - .vue로 구분한다.



트리 형식으로 표현 가능

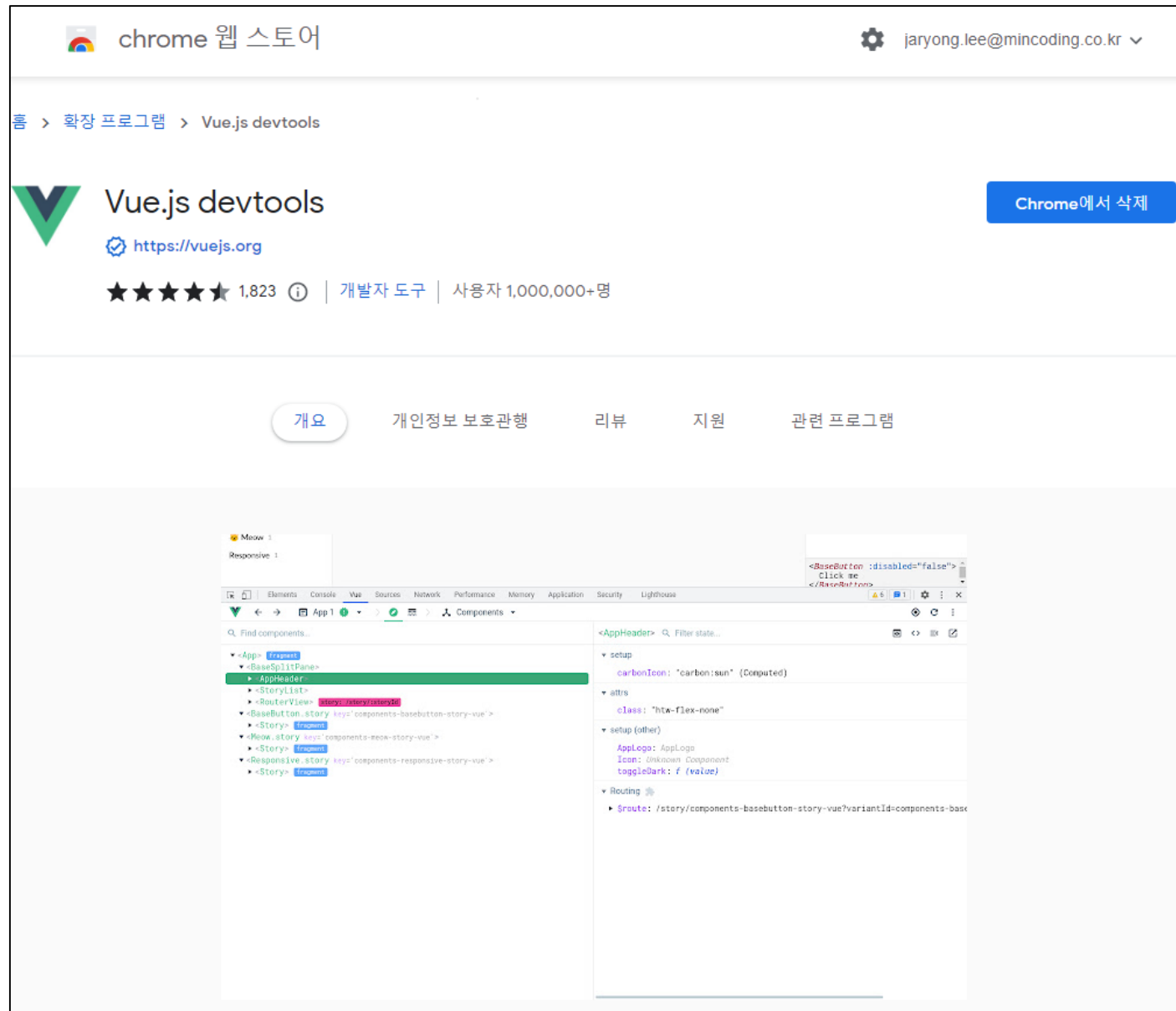
- App - 루트 컴포넌트
 - Header, Main : App 의 자식 컴포넌트

직접 구현해보자



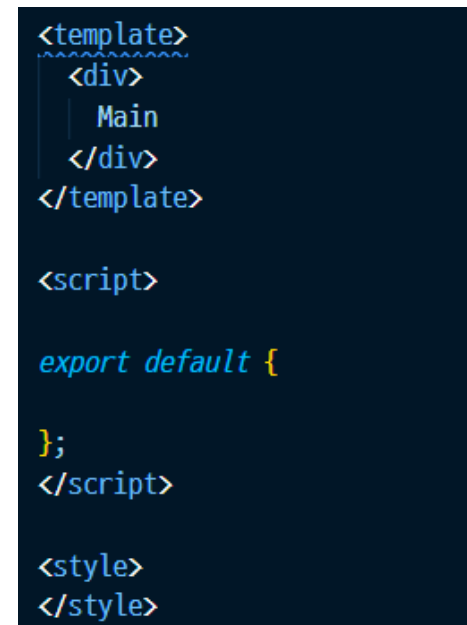
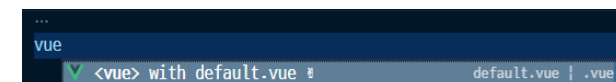
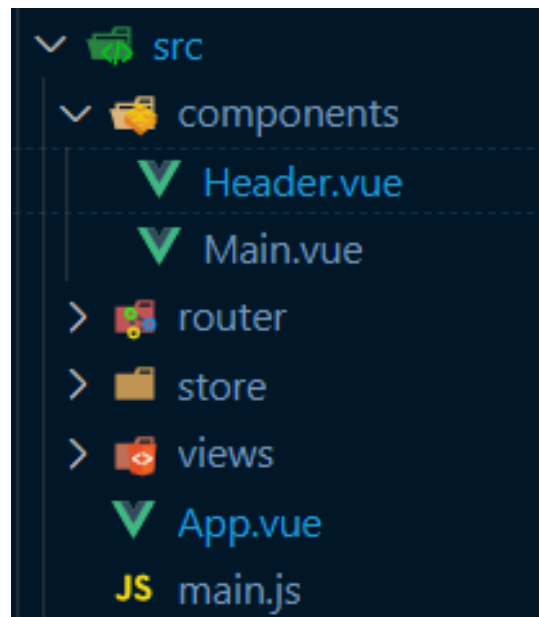
크롬 확장프로그램 설치

- 크롬 웹 스토어 접속
- Vue.js devtools 설치



컴포넌트 구현

- components 폴더에 각 componet생성
- .vue 파일에서 vue 라고 입력 후
Ctrl + Space(자동완성)
실행시 기본 템플릿이 생성



Navigation.vue

컴포넌트 생성시 주의사항

- 컴포넌트들은 모두 하나의 루트 엘리먼트를 가지는 형태가 되어야한다.
- 그렇지 않을 경우 에러가 발생

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

<script>
  export default {
}
</script>

<style>
</style>
```

```
you, 2 authors (you and
<template>
  <div>
    TEST
  </div>
  <div>
    ERROR
  </div>
</template>

<script>
  export default {
}
</script>

<style>
</style>
```

컴포넌트 활용

- 가져오기

- import 컴포넌트명 from "상대경로"
- import 는 ES Module 문법으로서,
node.js 의 require 와 같은 의미

- 등록하기

- 컴포넌트를 사용하기 위해,
components 객체에 등록
- import 를 제외한 모든 객체는
export default 내부에 작성

- 붙이기

- 가져온 컴포넌트를 태그 사용법과 동일하게
- 화면에 위치하고 싶은 곳에 기입
- 이후 컴포넌트 간 부모 자식 관계가 성립

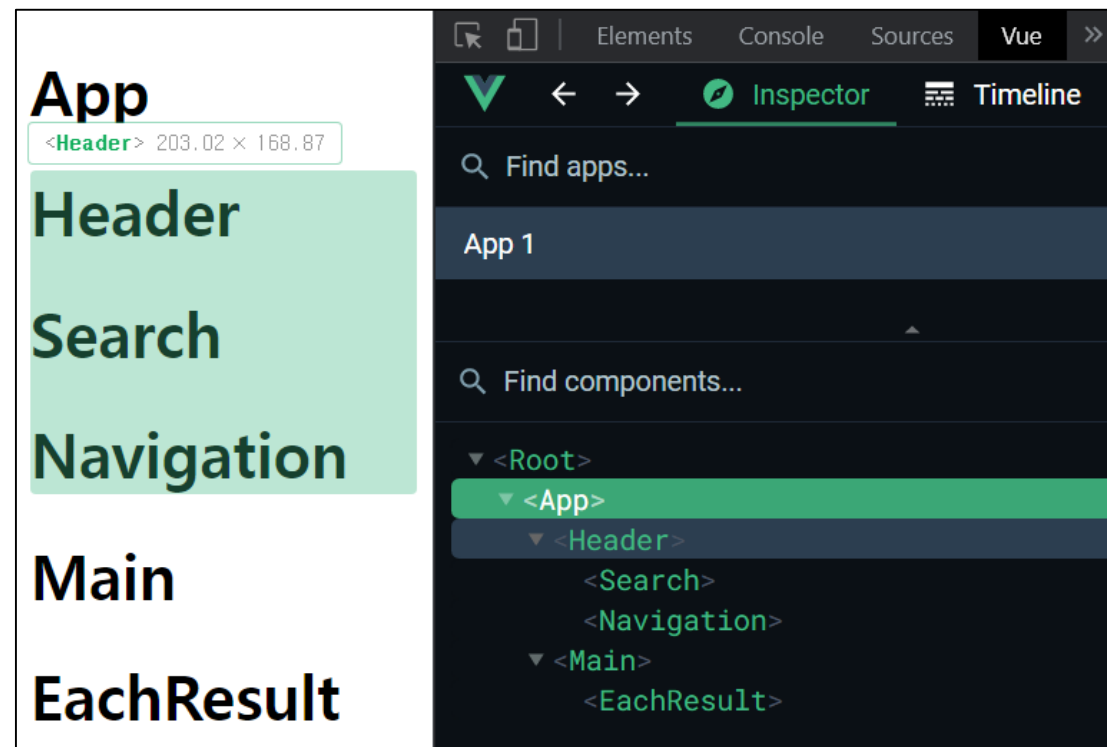
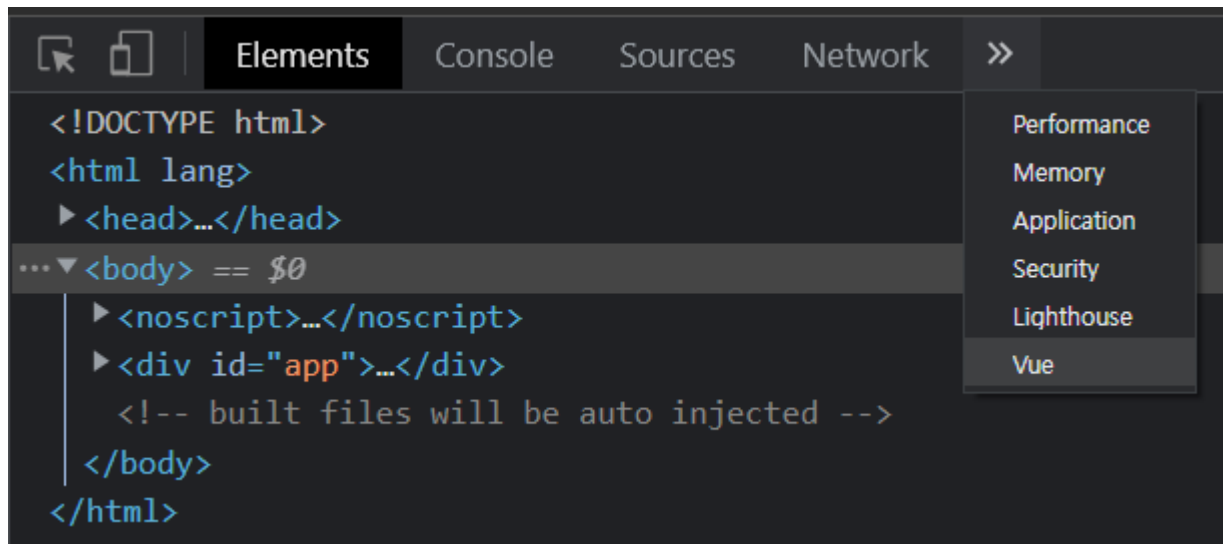
```
<template>
  <div>
    <h1>App</h1>
    <Header ></Header>
    <Main ></Main>
  </div>
</template>

<script>
import Header from "../components/Header";
import Main from "../components/Main";
export default {
  components: {
    Header,
    Main,
  },
};
</script>

<style>
</style>
```

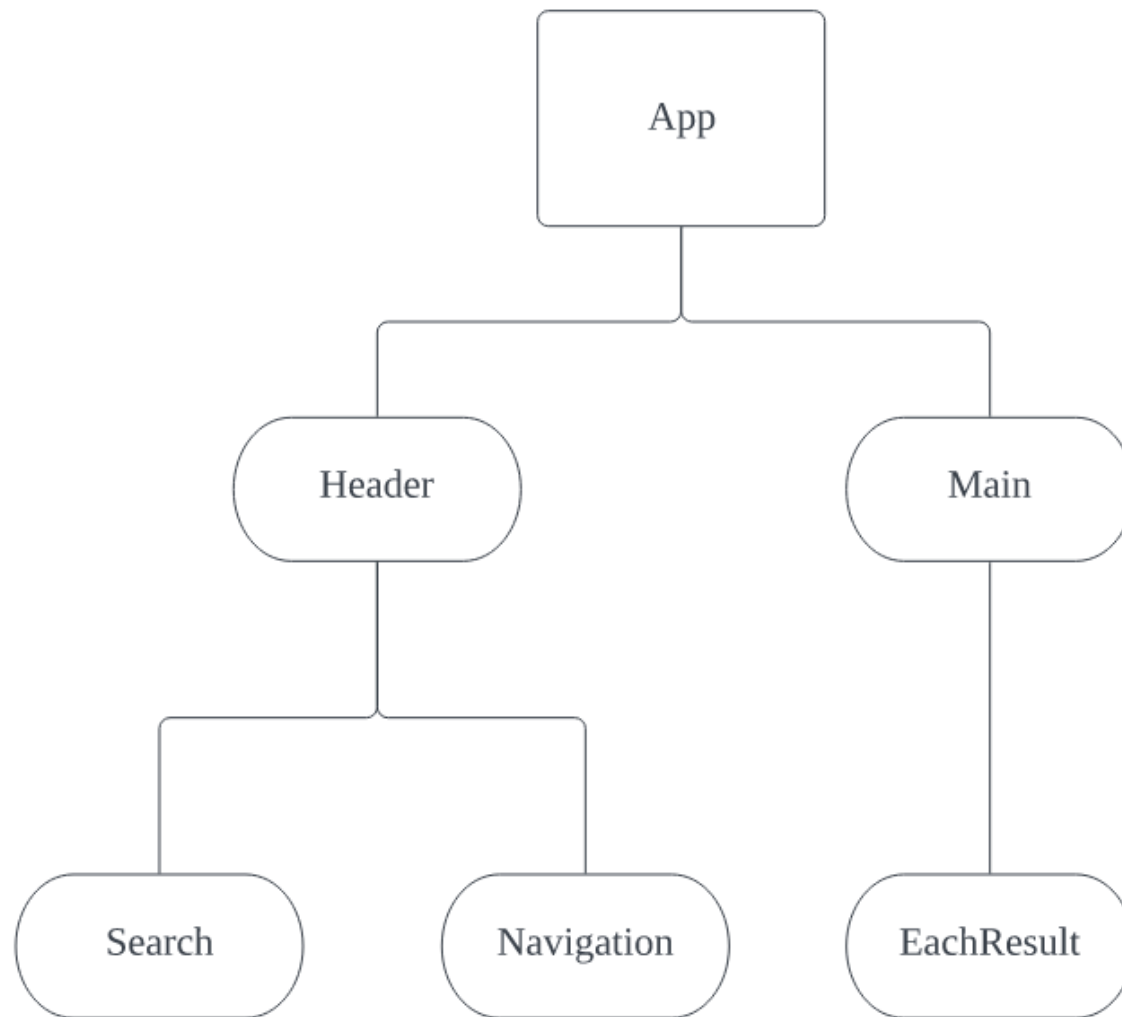
Vue.js devtools 사용

- 개발자 도구에서 >> 누르고 Vue 클릭



다음 컴포넌트 트리를 구현하기

- Header 의 자식 컴포넌트
 - Search, Navigation
- Main 의 자식 컴포넌트
 - EachResult
- 구현 후 Vue.js devtools 로 결과 확인



5장. Vue Router

챕터의 ^{• • •}포인트

- Vue Router
- SPA

Vue Router

Vue.js는 SPA

- 단 하나의 index.html만 가지고있는 어플리케이션이다.
 - 단 하나의 index.html만 가지고 있기 때문에 보여주고싶은 내용에 제약이 있을수 있다.
 - Vue Router를 통해 하나의 index.html 사용하지만 실제로 여러 주소를 활용하는것처럼 만들 수 있다.
 - 하나의 index.html만 존재하기때문에 라우터 이동간에도 깜빡임이 발생하지 않는다.

vue-router

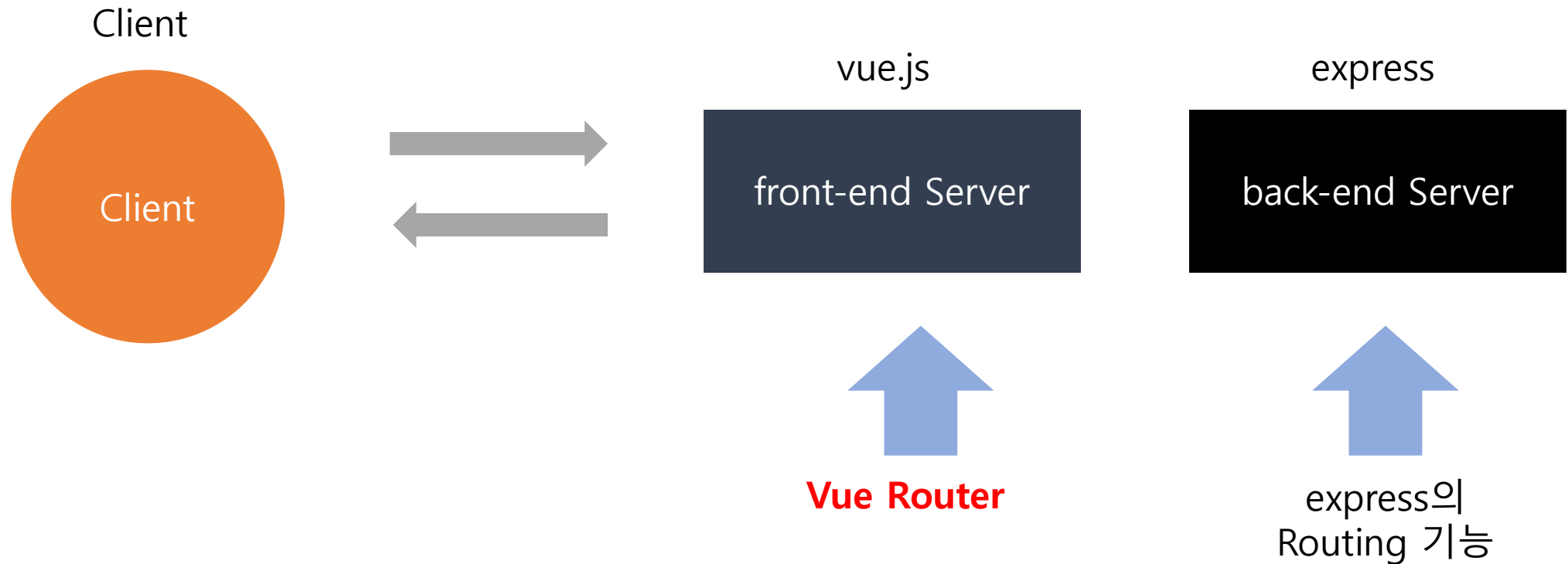
- vue.js 공식 라이브러리
- 라우트에 컴포넌트를 매핑
- 어떤 URL 에서 해당 컴포넌트를 렌더링할지 결정

vue-router



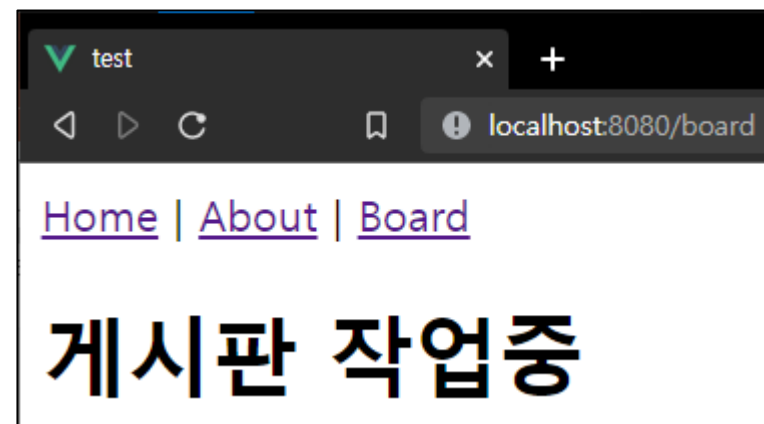
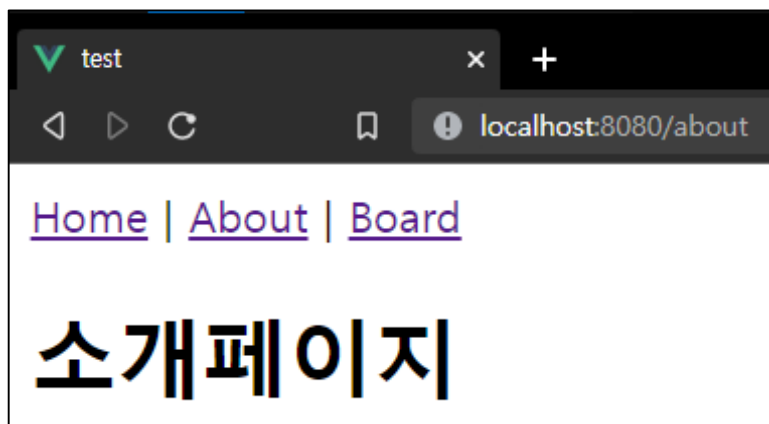
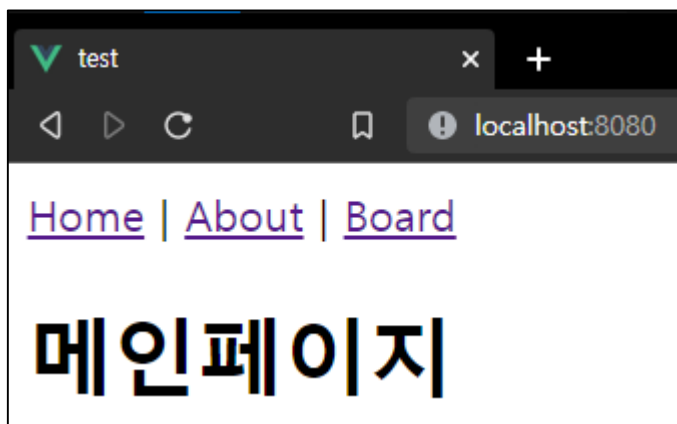
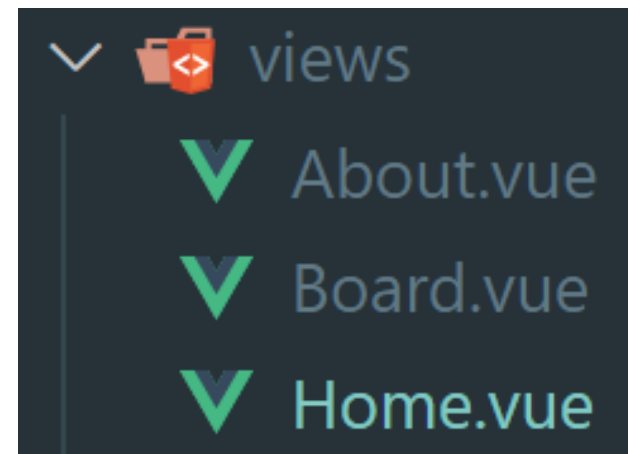
두가지 Routing

- 특정 라우터에 도착하면 원하는 부분을 보여준다는 것에서는 동일하다고 할수있다.
 - express Routing : Rest API 제작시,
 - Vue Routing** : URL 마다 다른 Content를 보여주고 싶을 때



링크 클릭 시 각각의 페이지로 전환

- 메인
 - / 도착시 Main.vue를 보여준다.
- 소개
 - /about 도착시 About.vue를 보여준다
- 게시판
 - /board 도착시 Board.vue를 보여준다
- 기본 .vue 템플릿으로 세 가지 파일 작성



App.vue 에 다음과 같이 작성

`<router-link to="경로">`

- `<a>` 태그와 동일하게 보여짐
- 클릭 시 `to="경로"` 에 지정된 라우트로 이동
 - `a` tag로 이동하는 경우 깜박임이 일어나지만 `router-link`를 활용하는 경우 깜박임이 일어나지 않는다.

`<router-view />`

- 라우트에 해당하는 화면이 실제 렌더링되는

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link> |
      <router-link to="/board">Board</router-link>
    </nav>
    <router-view />
  </div>
</template>
```

router/index.js 파일

- 각각의 파일 경로 import
- routes 배열은 다음 형태의 여러 개 객체로 구성
 - path
 - 라우트 경로
 - name
 - 라우트 명칭
 - component
 - 라우트에 해당하는 컴포넌트
- router/index.js 파일의 routes 에 등록되어 있어야만 `<router-link to="">` 는 작동

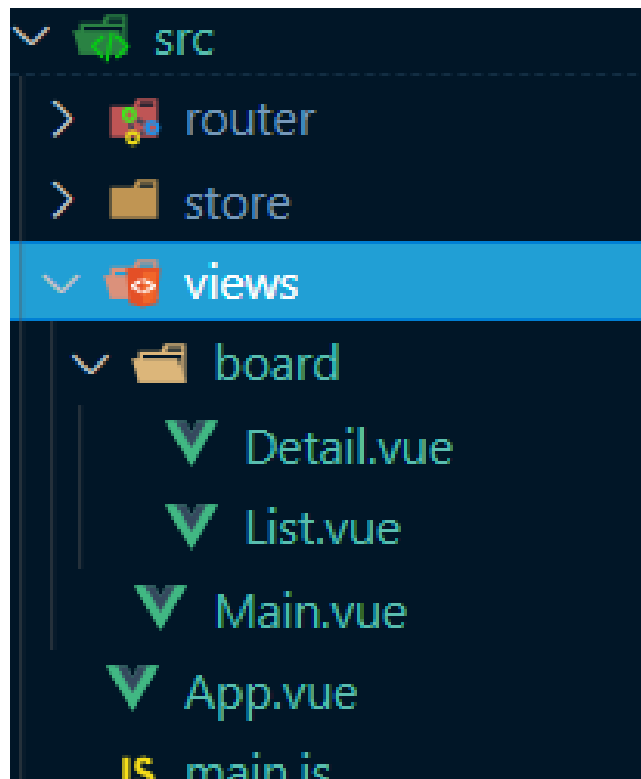
```
const routes = [  
  {  
    path: "/",  
    name: "home",  
    component: Home,  
  },  
  {  
    path: "/about",  
    name: "about",  
    component: About,  
  },  
  {  
    path: "/board",  
    name: "board",  
    component: Board,  
  },  
];
```

라우터 설계

1. 라우터 경로 설계
2. 라우터 경로에 맞는 views 폴더 내 파일.vue 생성
3. 해당 파일.vue를 router/index.js 에서 정의하기
4. 경로가 잘 맞는지 테스트

라우터 설계

- /views
 - Main.vue
 - 메인화면
 - board 폴더
 - List.vue
 - Board의 List 조회
 - Detail.vue
 - 특정 Board의 Detail 조회



라우터 설계

- /
 - main 화면
- /board
 - board-list
- /board/:id
 - board의 :id 번째
 - board-detail

```
import Vue from "vue";
import VueRouter from "vue-router";
import Main from "../views/Main.vue"
import BoardList from "../views/board/List.vue"
import BoardDetail from "../views/board/Detail.vue"

Vue.use(VueRouter);

const routes = [
  {
    path: "/",
    name: "Main",
    component: Main
  },
  {
    path: "/board",
    name: "board-list",
    component: BoardList,
  },
  {
    path: "/board/:id",
    name: "board-detail",
    component: BoardDetail,
  }
];
```

App.vue 꾸미기

- router-link 활용

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">Home</router-link> |
      <router-link to="/board">Board</router-link>
    </nav>
    <router-view />
  </div>
</template>

<script>
  // JS 코드 작성
</script>

<style>
  /* CSS 코드 작성 */
</style>
```


/board (views/board/List.vue)

- /board로 이동
- this.\$router
 - 전반적인 라우터 기능이 들어있는 객체
- this.\$route
 - 현재 있는 라우터에 대한 정보

```
<template>
  <div>

  </div>
</template>

<script>
export default {
  data() {
    return {

    }
  },
  created() {
    console.log(this.$router);
    console.log("-----")
    console.log(this.$route);
  }
}
</script>

<style>

</style>
```



```
[webpack dev server] Server started. Hot Module Replacement enabled.

VueRouter
  afterHooks: []
  app: Vue {__uid: 2, __isVue: true, __v_skip: true, __scope: EffectScope}
  apps: [Vue]
  beforeHooks: []
  fallback: false
  history: HTML5History {router: VueRouter, base: '/', current: {...}, p...}
  matcher: {match: f, addRoute: f, getRoutes: f, addRoutes: f}
  mode: "history"
  options: {mode: 'history', base: '/', routes: Array(3)}
  resolveHooks: []
  currentRoute: (...)
  [[Prototype]]: Object

-----

Object
  fullPath: "/board"
  hash: ""
  matched: [{...}]
  meta: {}
  name: "board-list"
  params: {}
  path: "/board"
  query: {}
  [[Prototype]]: Object
```

/board (views/board/List.vue)

- 라우터를 이동하는 법
 - router-link
 - to="라우터 경로"를 활용해 이동
 - this.\$router.push("라우터 경로")
 - this.\$router.push를 활용해 이동

```
<template>
  <div>
    <h1>라우터를 이동하는 방법</h1>

    <router-link to="/board/1">
      라우터 이동하는 법 1
    </router-link>
    <br>
    <button @click="moveRouter">
      라우터 이동하는법 2
    </button>
  </div>
</template>

<script>
export default {
  data(){
    return{
    }
  },
  created(){
    console.log(this.$router);
    console.log("-----")
    console.log(this.$route);
  },
  methods:{
    moveRouter(){
      this.$router.push("/board/1")
    }
  }
}
</script>

<style>

</style>
```

/board/:id (views/board/Detail.vue)

- params
 - node `/:id`와 동일하게 `this.$route.params`에 `:id`의 값이 담긴다.
- query
 - `/board/:id?name=hello`
 - `?key=value` 형식으로 `this.$route.query`에 담긴다.

```
<template>
  <div>

  </div>
</template>

<script>
export default {
  created(){
    console.log(this.$route);
  }
}
</script>

<style>

</style>
```

```
▼ Object 1
  fullPath: "/board/1"
  hash: ""
  ▶ matched: [{...}]
  ▶ meta: {}
  name: "board-detail"
  ▶ params: {id: '1'}
  path: "/board/1"
  ▶ query: {}
  ▶ [[Prototype]]: Object
```

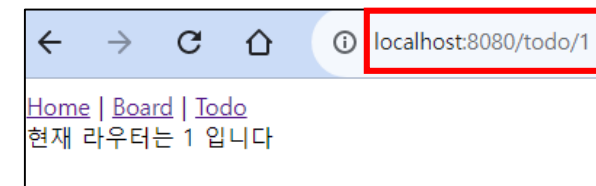
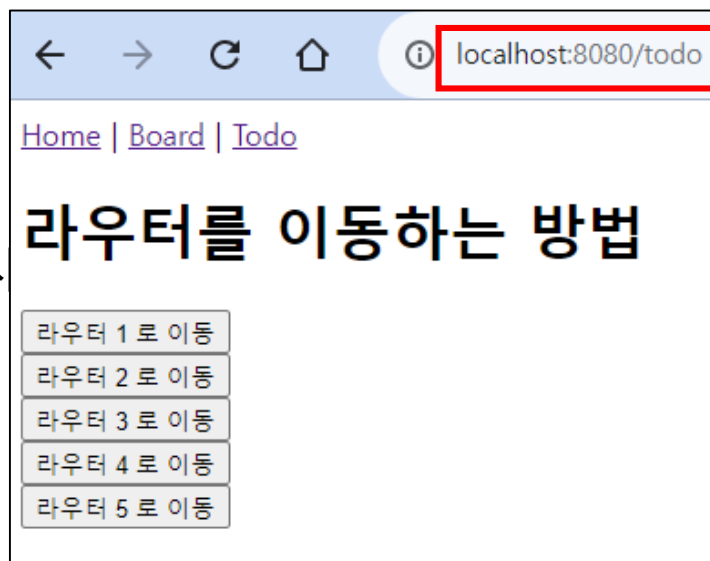
/board/1

```
▼ {name: 'board-detail', meta: {...},
  fullPath: "/board/2?name=hello"
  hash: ""
  ▶ matched: [{...}]
  ▶ meta: {}
  name: "board-detail"
  ▶ params: {id: '2'}
  path: "/board/2"
  ▶ query: {name: 'hello'}
  ▶ [[Prototype]]: Object
```

/board/2?name=hello

라우터 설계 및 views 연동

- 라우터 정의
 - /todo
 - /todo/:id
- /todo
 - data에 arr = [1, 2, 3, 4, 5] 정의
 - v-for 과 this.\$router.push를 활용해 라우터 이동하기
- /todo/:id
 - \$route 활용
 - 현재 라우터는 :id 입니다 표시



Vue Build

Vue.js 는 Single Page Application (SPA) 이다.

- Vue.js 는 한 페이지에서, 내용만 바꾸면서 보여주는 방식이기 때문에 Vue Router가 안에 있는 내용을 바꾸어 보여주는 원리로 구현 된다.
- 서버엔 하나의 HTML 파일만 존재
- 하나의 페이지에서 내용만 변경하여 렌더링
- 최초에 한 번 페이지 전체 로드
- 변경 필요 시 특정 부분만 AJAX 를 통해 데이터 가져와 변경
- 대표적인 SPA 프레임워크: Vue.js, React.js

빌드 시, 단 하나의 HTML 만 생성

- `npm run build`

DONE Compiled successfully in 5297ms

오후 10:06:58

File	Size	Gzipped
dist\js\chunk-vendors.5fed5894.js	128.49 KiB	43.62 KiB
dist\js\app.fd6a4e6e.js	3.19 KiB	1.46 KiB
dist\css\app.03546b2b.css	0.00 KiB	0.02 KiB

Images and other types of assets omitted.

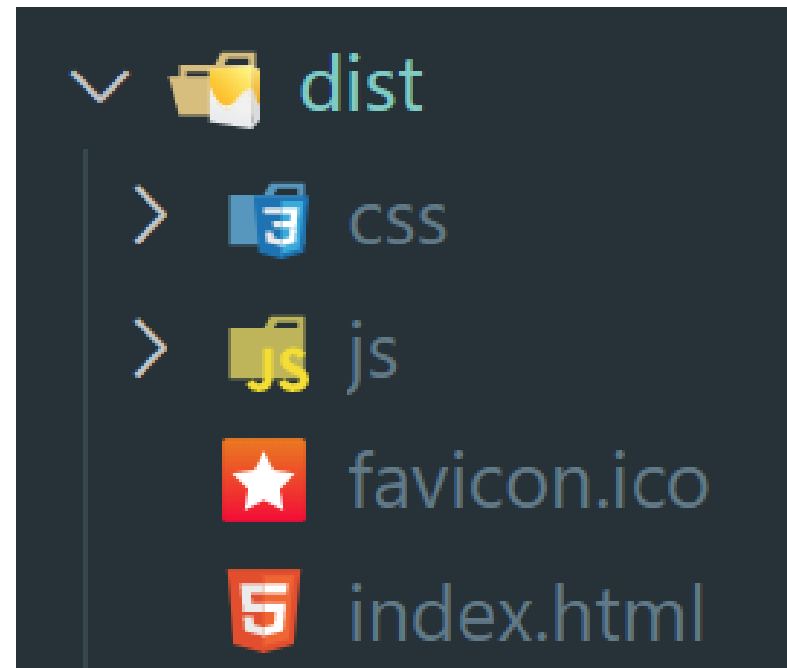
Build at: 2022-08-18T13:06:58.971Z – Hash: fb99f47cd986acfc – Time: 5297ms

DONE Build complete. The `dist` directory is ready to be deployed.

INFO Check out deployment instructions at <https://cli.vuejs.org/guide/deployment.html>

dist

- npm run build 이후 생성된 빌드 결과물
- Vue Router 를 사용했지만, 단 하나의 html 파일만 확인
- 개발 완료 후 dist 의 내용물을 서버에 업로드해 서비스



6장. Vuex

챕터의 포인트

- props
- emit
- Vuex

props

컴포넌트 분리의 장점

- 하나의 화면을 각 컴포넌트별로 개발 후, 하나의 HTML 문서로 결합
- 생산성 향상 및 유지보수 편리

컴포넌트 분리의 단점

- 분리된 컴포넌트 간 데이터 연동을 고려해야
- 부모 컴포넌트가 자식 컴포넌트에 데이터를 전달하고 싶다면?

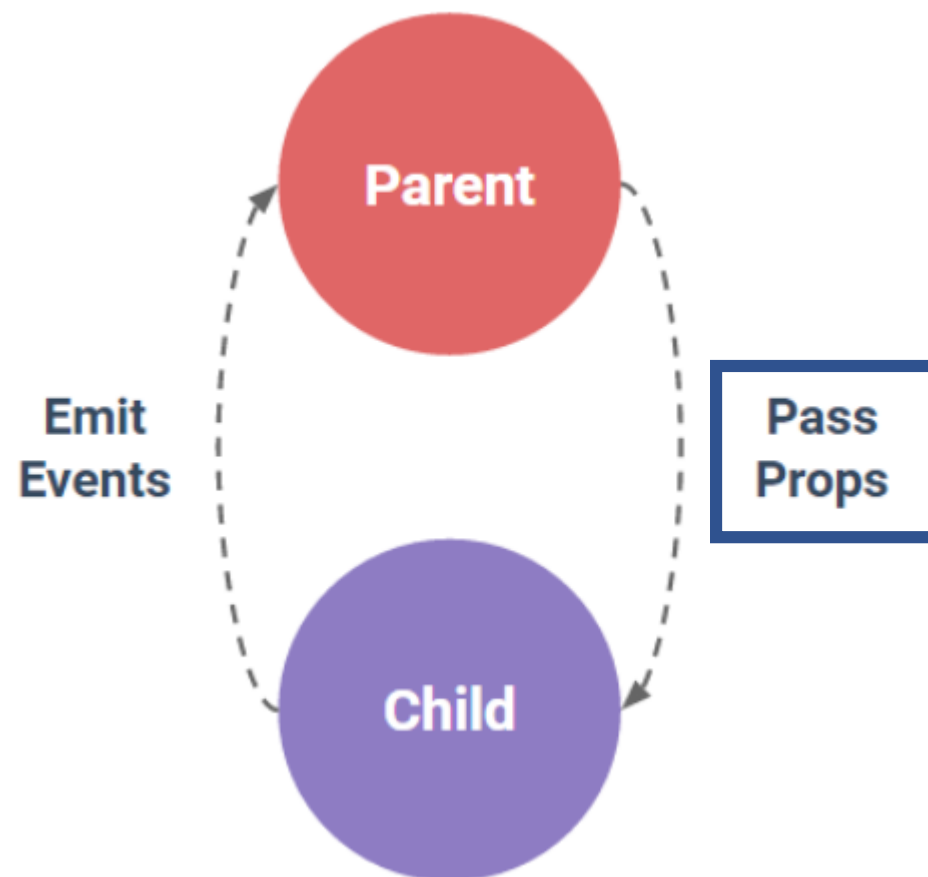
각각의 컴포넌트로 나눈 화면

- 각 컴포넌트마다 일치해야하는 데이터
 - 프로필사진 링크
 - 사용자 아이디
- 컴포넌트마다 변수를 일일이 선언한다
 - 유지/보수 매우 불편
 - 각 컴포넌트마다 서버에 데이터 요청
- 해결책
 - 부모 컴포넌트에서 데이터를 관리
 - 자식에게 전달



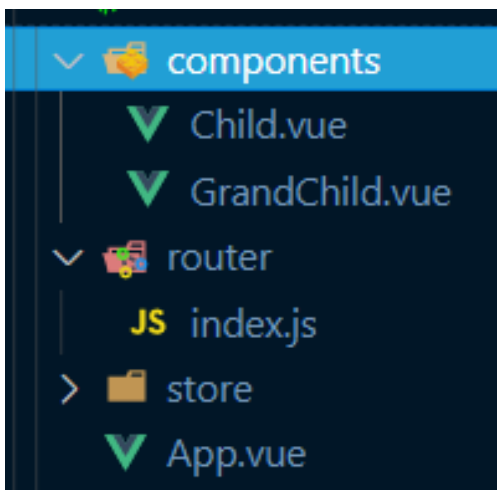
Props

- 부모 컴포넌트에서 자식 컴포넌트로 데이터 전달
- v-bind 로 구현
- 두 개 이상의 props 전달 가능
- props/emit 은 반드시 부모 자식 관계여야만 가능

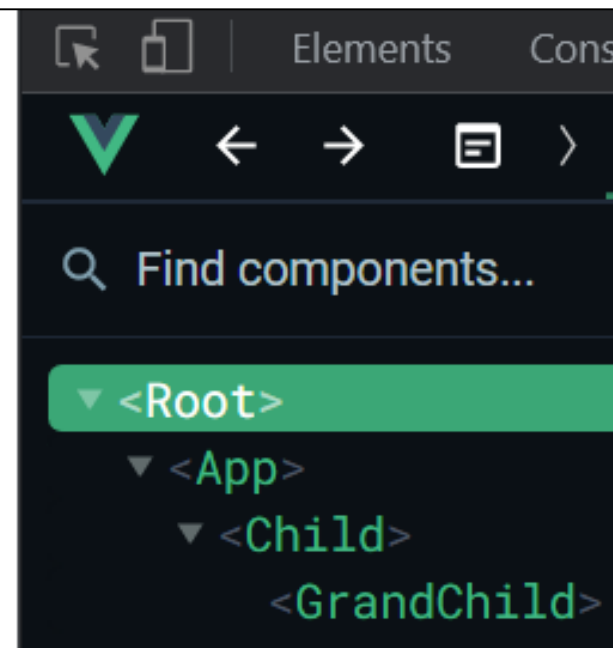


다음 형태의 프로젝트를 준비하자

- 최상위 컴포넌트 App.vue
 - 할아버지
- 자식 컴포넌트 Child
 - 아버지
- 손자 컴포넌트 GrandChild
 - 손자



할아버지
아버지
손자



App 에서 Child 로 데이터 전달

- App 에서 mySchool, myAge 를 Child 로 전달할 것
- v-bind:받을이름="보낼변수"

```
<template>
  <div id="app">
    <h1>할아버지</h1>

    <Child
      v-bind:mySchoolProp="mySchool"
      v-bind:myAgeProp="myAge"
    />
  </div>
</template>

<script>
import Child from "./components/Child";
export default {
  components: {
    Child,
  },
  data() {
    return {
      mySchool: "ssafy",
      myAge: 25,
    };
  },
  methods: {

  },
};
</script>

<style>
</style>
```

Child 에서 전달 받을 때

- props 객체에 배열로 각각을 받음
 - App 에서 지정한 이름과 일치해야
- data 객체 내의 변수 사용법과 동일

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>

    <GrandChild
      v-bind:myAgeProp="myAgeProp"
    />
  </div>
</template>

<script>
import GrandChild from "./GrandChild";
export default {
  components: {
    GrandChild,
  },
  props: ["mySchoolProp", "myAgeProp"],
  methods: {

  },
};
</script>

<style>
</style>
```

props 전달 과정

```
<template>
  <div id="app">
    <h1>할아버지</h1>

    <Child
      v-bind:mySchoolProp="mySchool"
      v-bind:myAgeProp="myAge"
    />
  </div>
</template>

<script>
import Child from "./components/Child";
export default {
  components: {
    Child,
  },
  data() {
    return {
      mySchool: "ssafy",
      myAge: 25,
    };
  },
  methods: {

  },
};
</script>

<style>
</style>
```

App.vue

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>

    <GrandChild
      v-bind:myAgeProp="myAgeProp"
    />
  </div>
</template>

<script>
import GrandChild from "./GrandChild";
export default {
  components: {
    GrandChild,
  },
  props: ["mySchoolProp", "myAgeProp"],
  methods: {

  },
};
</script>

<style>
</style>
```

Child.vue

App 에서 클릭 버튼을 추가해 myAge 변경

- 클릭 시 25세에서 15세로 변경
 - mySchoolProp 또한 변경된것을확인
 - mySchoolProp 역시 바뀐 것으로 확인된다
 - mySchool 과 mySchoolProp 은 같은 주소를 함함
 - 즉, 이름을 하나 더 지정했을 뿐 같은 변수

```
<template>
  <div id="app">
    <h1>할아버지</h1>
    <button
      v-on:click="changeAge"
    >나이 변경</button>
    <Child
      v-bind:mySchoolProp="mySchool"
      v-bind:myAgeProp="myAge"
    />
  </div>
</template>

<script>
import Child from "../components/Child";
export default {
  components: {
    Child,
  },
  data() {
    return {
      mySchool: "ssafy",
      myAge: 25,
    };
  },
  methods: {
    changeAge() {
      this.myAge = 15;
    },
  },
};
</script>
```

할아버지

나이 변경

아버지

학교: ssafy

나이: 15

이번엔, 자식인 Child 에 클릭 버튼 추가해 myAgeProp 변경

- 값 수정 확인되나, warning 발생
 - Vue.js 에선 자식이 부모의 prop 을 함부로 바꾸면 안된다
 - 부모 컴포넌트 렌더링 시 값이 다시 쓰여지는 문제 발생
- 자식에서 부모의 값을 바꾸고 싶을 땐 다음 장에서 배울 emit 을 사용

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>
    <!-- 예러 발생 -->
    <button v-on:click="changeAge">나이 변경</button>
    <!--// 예러 발생 -->
  </div>
</template>

<script>
import GrandChild from "../GrandChild";
export default {
  components: {
    GrandChild,
  },
  props: ["mySchoolProp", "myAgeProp"],
  methods: {
    changeAge() {
      this.myAgeProp = 15;
    },
  },
};
</script>
```

```
✖ [Vue warn]: Avoid mutating a prop directly since the value will be
overwritten whenever the parent component re-renders. Instead, use a data or
computed property based on the prop's value. Prop being mutated:
"myAgeProp"

found in
---> <Child> at src/components/Child.vue
      <App> at src/App.vue
        <Root>
```

자식에서 받은 prop 은 손자에게 전달 가능

- 자식에서 손자로 넘김
- 손자에서 props 배열로 받은 다음 사용 가능

```
<!-- GrandChild.vue (Child of GrandChild) -->
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>
  </div>
</template>

<!-- 예러 발생 -->
<button v-on:click="changeAge">나이 변경</button>
<!--// 예러 발생 -->

<script>
  import GrandChild from "../GrandChild";
  export default {
    components: {
      GrandChild,
    },
    props: ["mySchoolProp", "myAgeProp"],
    methods: {
      changeAge() {
        this.myAgeProp = 15;
      },
    },
  };
</script>

<style>
</style>
```

Child.vue

```
<!-- GrandChild.vue (Child of GrandChild) -->
<template>
  <div>
    <h1>손자</h1>
    <div>나이: {{ myAgeProp }}</div>
  </div>
</template>

<script>
  export default {
    props: ["myAgeProp"],
  };
</script>

<style>
</style>
```

GrandChild.vue

정리

- props는 부모에서 자식으로 데이터를 넘겨준다.
- 부모 -> 자식 -> 손자 에게도 데이터를 넘겨주는것이 가능
 - 자식에서 데이터를 변경하기 X

Props 활용하기

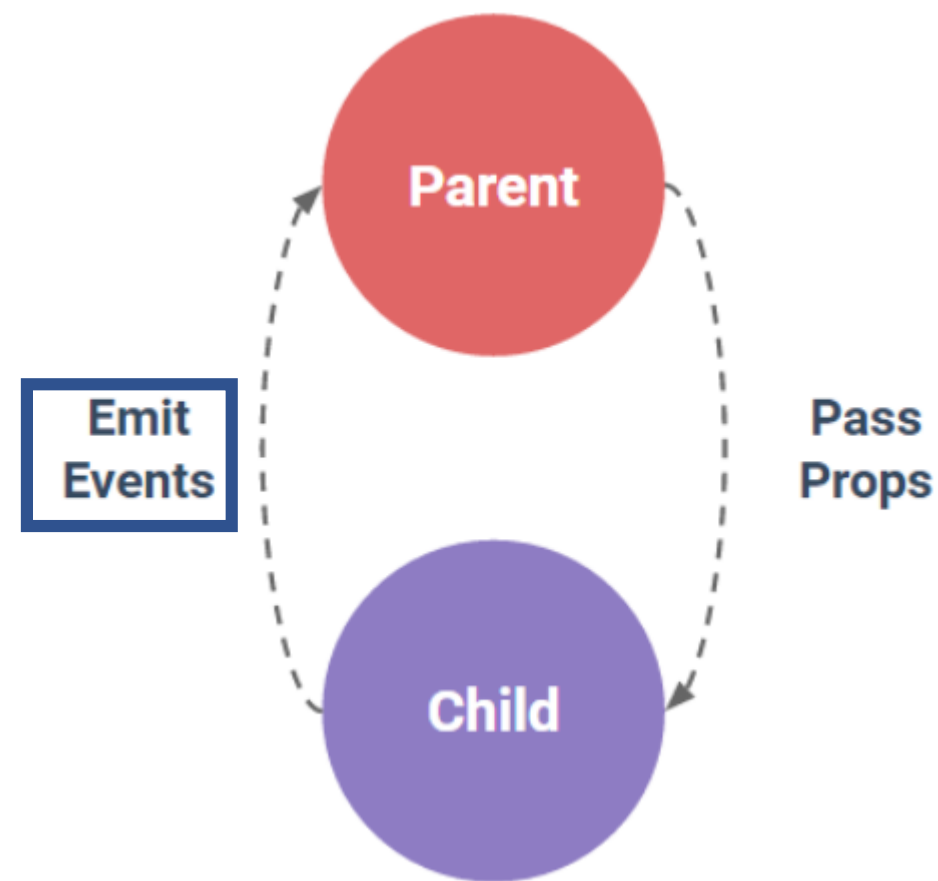
- App 컴포넌트
 - data { salt: 30, title: “치킨은 맛있다”}가 있다.
 - App.vue 컴포넌트 안에 FriedChicken 컴포넌트
 - FriedChicken 컴포넌트 안에 ChickenChild 컴포넌트
 - ChickenChild 컴포넌트 안에는 버튼이 있다.
 - 치킨의 간을 맞추기 위해 소금을 넣으려고 한다.
 - ChickenChild 의 버튼 클릭시 salt 와 title 을 `{{ salt }}` `{{ title }}` 로 보여주세요



emit

emit

- 자식 컴포넌트에서 부모 컴포넌트의 메서드 실행
- props 는 v-bind를 사용하지만 emit은 v-on을 활용
- props/emit 은 반드시 부모자식 관계여야만 가능



GrandChild 에서 myAge 변경

- 제시된 코드처럼, 직접 변경 불가능 - 손자의 값만 바뀜
- 부모에게 바꿔달라고 "요청"해야한다

```
<h1>손자</h1>
<div>나이: {{ myAgeProp }}</div>
<button
  v-on:click="changeAge"
>나이 변경</button>
```

```
export default {
  props: ["myAgeProp"],
  methods: {
    changeAge() {
      this.myAgeProp = 15;
    },
  },
};
```

할아버지

아버지

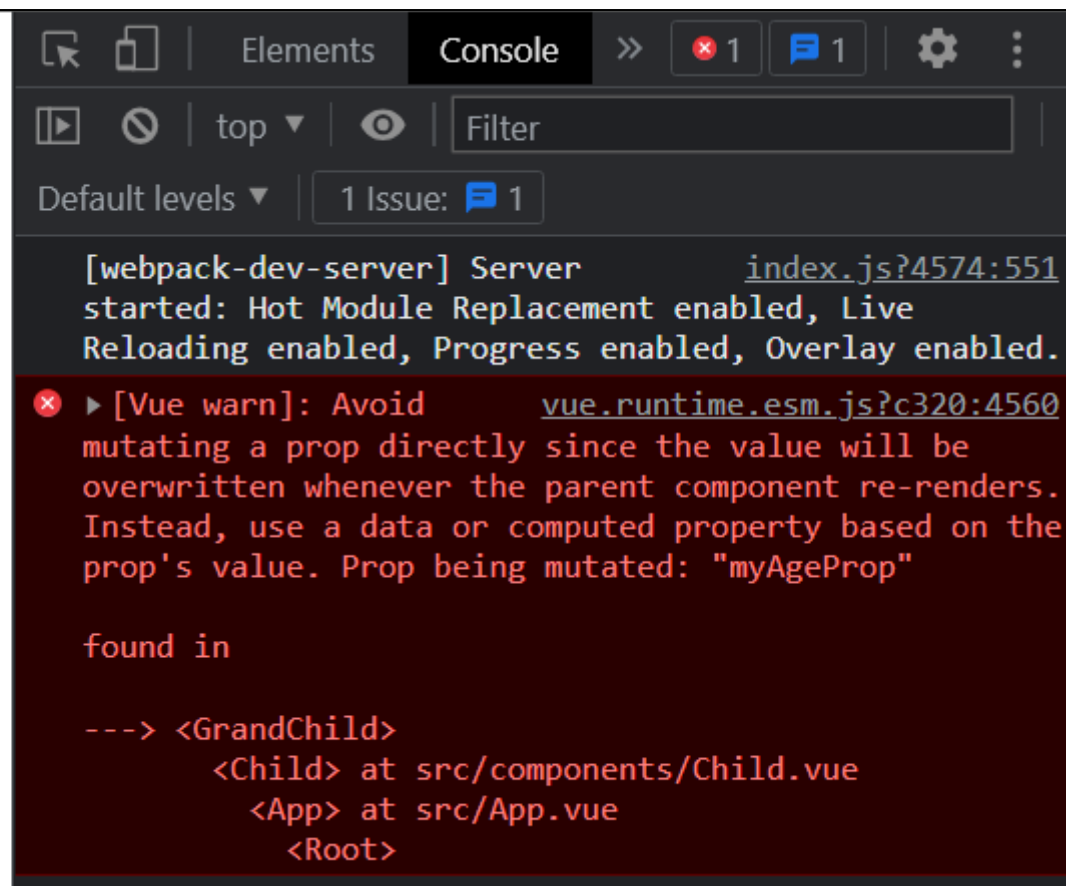
학교: ssafy

나이: 25

손자

나이: 25

나이 변경



자식에서 부모로 이벤트 전달

`this.$emit("이벤트", 매개변수)`

- 클릭 시 부모 컴포넌트에 emitTest 라는 이벤트 전달
- 메서드가 아니라, "우리가 직접 이름지은" 이벤트 이름
- 매개변수를 전달하고 싶을 경우, 콤마 , 로 구분해 기입

```
<template>
  <div>
    <h1>손자</h1>
    <div>나이: {{ myAgeProp }}</div>
    <button
      v-on:click="changeAge"
    >나이 변경</button>
  </div>
</template>

<script>
export default {
  props: ["myAgeProp"],
  methods: {
    changeAge() {
      this.$emit("emitTest", 15);
    },
  },
};
</script>

<style>
</style>
```

부모에서 이벤트 수신

- v-on:자식이벤트="부모메서드"
 - emitTest 이벤트가 들어올 경우,
부모에서 실행시킬 메서드 지정
- 부모에선 changeAge() 메서드 실행시킴
- 이 메서드는 자식의 changeAge() 와
이름이 같을 뿐, 전혀 다른 메서드
- 보내준 매개변수 갯수와 일치하게 파라미터로 받음

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>
    <!-- Child -->
    <GrandChild
      v-bind:myAgeProp="myAgeProp"
      v-on:emitTest="changeAge"
    />
  </div>
</template>

<script>
import GrandChild from "../GrandChild";
export default {
  components: {
    GrandChild,
  },
  props: ["mySchoolProp", "myAgeProp"],
  methods: {
    // Child
    changeAge(age) {
      console.log(age)
    },
  },
};
</script>

<style>
</style>
```

동작 정리

- GrandChild 클릭시 emit으로 부모컴포넌트 전달
- 부모 컴포넌트인 Child에서 emitTest를 changeAge로 수신

```
<template>
  <div>
    <h1>손자</h1>
    <div>나이: {{ myAgeProp }}</div>
    <button
      v-on:click="changeAge"
    >나이 변경</button>
  </div>
</template>

<script>
export default {
  props: ["myAgeProp"],
  methods: {
    changeAge() {
      this.$emit("emitTest", 15);
    },
  },
};
</script>

<style>
</style>
```

GrandChild.vue

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>
    <!-- Child -->
    <GrandChild
      v-bind:myAgeProp="myAgeProp"
      v-on:emitTest="changeAge"
    />
  </div>
</template>

<script>
import GrandChild from "./GrandChild";
export default {
  components: {
    GrandChild,
  },
  props: ["mySchoolProp", "myAgeProp"],
  methods: {
    // Child
    changeAge(age) {
      console.log(age)
    },
  },
};
</script>

<style>
</style>
```

Child.vue

하지만, myAge 는 App 에 있으므로 한번 더 올라가야 함

- emit 을 한번 더 작성
 - Child.vue에서 부모 컴포넌트인 App.vue에 emit으로 요청을 보내고 App.vue에서는 changeAge라는 메서드로 받아온다
 - 클릭하면, 아버지와 손자에 있는 Age 가 한 번에 변경됨을 알 수 있다.

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ mySchoolProp }}</div>
    <div>나이: {{ myAgeProp }}</div>
    <!-- Child -->
    <GrandChild
      v-bind:myAgeProp="myAgeProp"
      v-on:emitTest="changeAge"
    />
  </div>
</template>

<script>
import GrandChild from "../GrandChild";
export default {
  components: {
    GrandChild,
  },
  props: ["mySchoolProp", "myAgeProp"],
  methods: {
    // Child
    changeAge(age) {
      this.$emit("emitTest", age);
    },
  },
};
</script>

<style>
</style>
```

Child.vue

```
<template>
  <div id="app">
    <h1>할아버지</h1>
    <Child
      v-bind:mySchoolProp="mySchool"
      v-bind:myAgeProp="myAge"
      v-on:emitTest="changeAge"
    />
  </div>
</template>

<script>
import Child from "../components/Child";
export default {
  components: {
    Child,
  },
  data() {
    return {
      mySchool: "ssafy",
      myAge: 25,
    };
  },
  methods: {
    changeAge(age) {
      this.myAge = age;
    },
  },
};
</script>
```

App.vue

할아버지

아버지

학교: ssafy
나이: 15

손자

나이: 15

나이 변경

실제 데이터 위치

mySchool: "ssafy"
myAge: 25

mySchoolProp: "ssafy"
myAgeProp: 25

myAgeProp: 25

App

Child

GrandChild



v-on:자식이벤트="부모메서드"
실제 변수 변경

v-on:자식이벤트="부모메서드"
this.\$emit("이벤트", 매개변수)
즉, 중계만 담당

this.\$emit("이벤트", 매개변수)

왜 자식에서 props 를 함부로 바꿀 수 없을까?

- 모든 컴포넌트에서 데이터를 마음대로 바꿀 수 있을 경우
 - 전체 데이터 흐름이 엉망이 되어버림
- 하나의 방향으로만 데이터 전달을 허용할 경우
 - 데이터의 일관성 유지

Emit, Props 활용하기

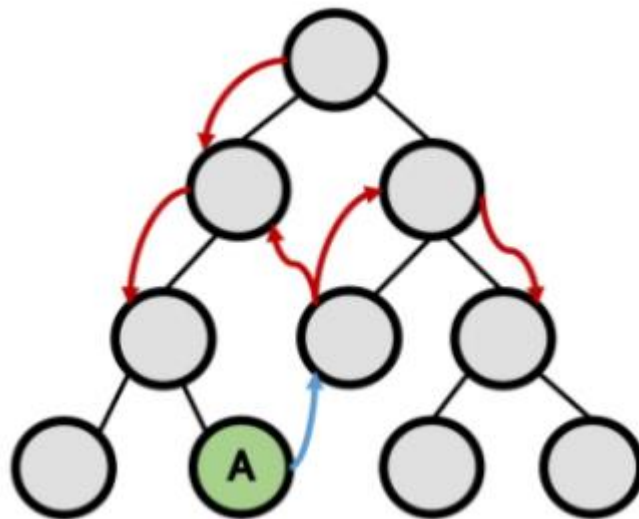
- 구조
 - App.vue
 - FriedChicken.vue
 - ChickenChild.vue
 - button
- App.vue
 - title 치킨은 맛있다, salt: 30 를 ChickenChild까지 전달
 - ChickenChild의 button 클릭시
 - salt를 20으로 변경



Vuex

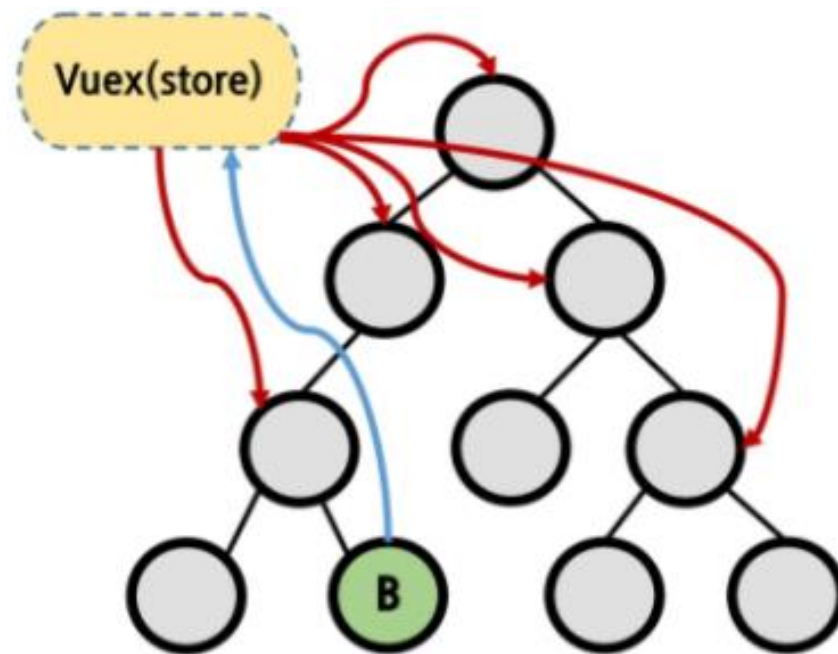
만약, 컴포넌트가 스무 개 정도 된다면?

- 반복되는 props와 emit으로 상태 관리 어려워짐
 - * 프론트엔드 프레임워크에서, 데이터를 다른 말로 상태(state)라고 함



Vuex

- Vue의 상태 관리 패턴 + 라이브러리
- 모든 컴포넌트의 중앙 집중식 저장소 역할
- 데이터 위치 파악 및 예측 가능
 - Vue Router 처럼 패키지 설치 필요



Vuex 를 배웠으면, props/emit 이 필요없는가?

- Vuex 를 도입해도 props/emit 과 함께 사용 가능
- 부모 자식 컴포넌트 간 간단한 데이터 통신은 props/emit 사용
- Vuex 없이도 간단한 프로젝트 개발 가능

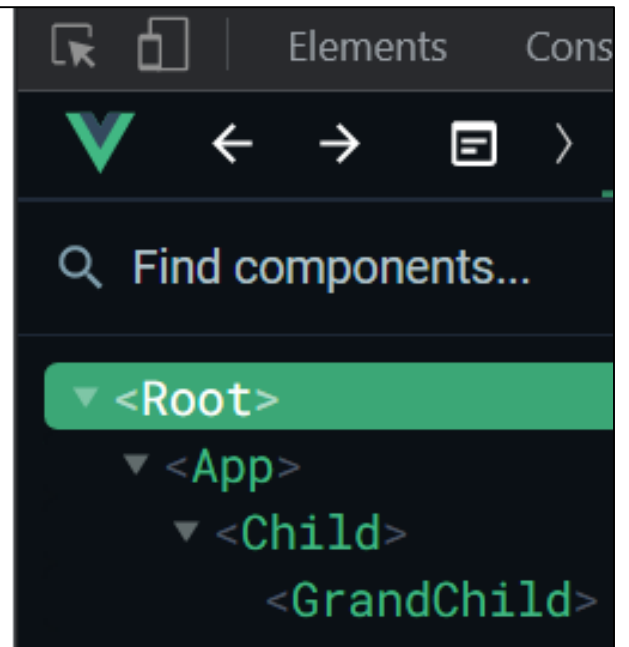
다음 형태의 프로젝트를 준비하자

- 최상위 컴포넌트 App
 - 자식 컴포넌트 Child
 - 손자 컴포넌트 GrandChild

할아버지

아버지

손자



src/store/index.js

- Vuex 설정파일
- state
 - 상태를 가지고 있는 객체
 - 상태 저장소
- mutations(state, data)
 - state를 변경하기 위해서는 mutations를 거쳐야한다.
 - 주로 대문자로 선언

```
import Vue from "vue";
import Vuex from "vuex";

Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    myName: "싸피",
    myAge: 25,
  },
  mutations: {
    SET_MY_NAME(state, data) {
      state.myName = data;
    },
    SET_MY_AGE(state, data) {
      state.myAge = data;
    },
  },
});
```

state의 데이터 가져오기

- 필요한 컴포넌트에서 Vuex 에 직접 요청
 - 부모에서 데이터를 받아 쓰는 게 아닌 Vuex의 store에 요청한다
- state에 접근하는 법
 - this.\$store.state
 - store의 state에 접근할수 있는 방법이다.
 - state를 변경하는경우 mutations를 거쳐야한다
 - 데이터의 일관성 유지

```
<template>
  <div>
    <h1>아버지</h1>
    <div>학교: {{ $store.state.myName }}</div>
    <div>나이: {{ $store.state.myAge }}</div>
    <GrandChild />
  </div>
</template>

<script>
import GrandChild from "../GrandChild";
export default {
  components: {
    GrandChild,
  },
  created(){
    console.log(this.$store.state);
  }
};
</script>

<style>
</style>
```

아버지

학교: 싸피
나이: 25

mutations를 통해 state 변경하기

- `this.$store.commit("mutations의 메서드 이름", 전달할 값)`

```
<template>
  <div>
    <h1>손자</h1>
    <button v-on:click="changeAge">나이 변경</button>
  </div>
</template>

<script>
export default {
  methods: {
    changeAge() {
      this.$store.commit("SET_MY_AGE", 15)
    },
  },
};
</script>

<style>
</style>
```

```
export default new Vuex.Store({
  state: {
    myName: "싸피",
    myAge: 25,
  },
  mutations: {
    SET_MY_NAME(state, data) {
      state.myName = data;
    },
    SET_MY_AGE(state, data) {
      state.myAge = data;
    },
  },
});
```

Chicken 리팩토링

- props/emit 을 사용한 Chicken 코드를 Vuex 를 사용해 리팩토링해보기
 - App.vue
 - FriedChicken.vue
 - title 치킨은 맛있다, salt: 30
 - ChickenChild.vue
 - button
 - 버튼 클릭시 salt를 20으로 변경



7장. Bootstrap Vue

챕터의 ^{• • •}포인트

- Bootstrap Vue


Bootstrap Vue


Bootstrap Vue

- Virtual DOM 을 사용하기 때문에
 - bootstrap 을 바로 적용시 DOM을 다시 그리는 부분에서 성능 이슈가 날 수 있음
- 일반적인 Bootstrap의 사용법에 Component 방식이 더해짐


BootstrapVue

With **BootstrapVue** you can build responsive, mobile-first, and ARIA accessible projects on the web using Vue.js and the world's most popular front-end CSS library — Bootstrap v4.

 **Bootstrap v4** is the world's most popular framework for building responsive, mobile-first sites.

 **Vue.js** (pronounced /vju:/, like view) is a progressive framework for building user interfaces.

Current Version
v2.22.0



컴포넌트 형태로 제작이 되어있다.

- vue에 맞게 data바인딩이 최적화 되어있음.
- bootstrap의 기능들 또한 사용 가능

☒ I accept the terms and use
State: **accepted**

```
<template>
  <div>
    <b-form-checkbox
      id="checkbox-1"
      v-model="status"
      name="checkbox-1"
      value="accepted"
      unchecked-value="not_accepted"
    >
      I accept the terms and use
    </b-form-checkbox>

    <div>State: <strong>{{ status }}</strong></div>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        status: 'not_accepted'
      }
    }
  }
</script>
```

```
$ npm i bootstrap-vue bootstrap@4.6.1
```

- Bootstrap Vue 는
Bootstrap 5 버전을 지원하지 않기 때문에 Bootstrap 버전 명시 필요
- package.json 에서 설치된 버전 확인

```
"dependencies": {  
  "bootstrap": "^4.6.1",  
  "bootstrap-vue": "^2.22.0",  
  "core-js": "^3.8.3",  
  "vue": "^2.6.14",  
  "vue-router": "^3.5.1",  
  "vuex": "^3.6.2"  
},
```

src/main.js 수정

- main.js 는 프로젝트 전역 JS 설정 파일
- 각각의 컴포넌트에서
import 할 필요 없이 Bootstrap 사용 가능

src > JS main.js > ...

```
1  import Vue from 'vue'
2  import App from './App.vue'
3  import router from './router'
4  import store from './store'
5
6  Vue.config.productionTip = false
7
8  import { BootstrapVue } from "bootstrap-vue";
9  import "bootstrap/dist/css/bootstrap.css";
10 import "bootstrap-vue/dist/bootstrap-vue.css";
11 Vue.use(BootstrapVue);
12
13 new Vue({
14   router,
15   store,
16   render: h => h(App)
17 }).$mount('#app')
```

버튼을 하나 달아보자

- Bootstrap 은 class 를 사용해 CSS 와 JS 를 적용했지만,
- Bootstrap Vue 는 컴포넌트 방식 채택

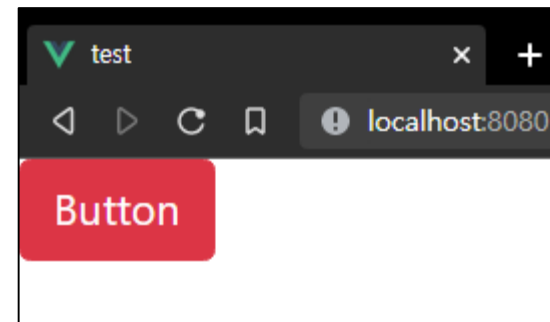
```
<template>
  <div id="app">
    <b-button variant="danger"> Button</b-button>
  </div>
</template>

<script>
export default {
  data() {

  },
};
</script>

<style>
</style>
```

App.vue



Vue.js 지시자 사용 가능

- b-form-input 컴포넌트에서 v-model 을 사용
 - 각각의 Bootstrap Vue 컴포넌트는 Vue.js 문법 사용 가능

```
<template>
  <div id="app">
    <b-form-input
      v-model="myName"
      placeholder="이름을 입력하세요"
    />
    <div>이름: {{ myName }}</div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      myName: "",
    };
  },
};
</script>

<style>
</style>
```

싸피

이름: 싸피

Vuex를 활용한 로딩 화면 구현

- Vue-bootstrap + router + Vuex를 활용한 로딩화면 구현
 - `npm i axios bootstrap-vue bootstrap@4.6.1`
 - jsonplaceholder 활용



```
<div>  
  <b-spinner label="Loading..."></b-spinner>  
</div>
```

Vue-bootstrap 셋팅

- src/main.js 수정
 - bootstrap을 위한 셋팅 추가

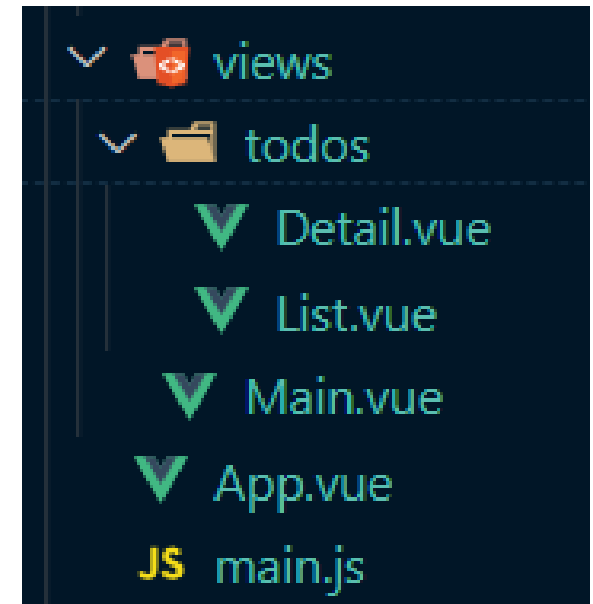
```
src > JS main.js > ...
1  import Vue from 'vue'
2  import App from './App.vue'
3  import router from './router'
4  import store from './store'
5
6  Vue.config.productionTip = false
7
8  import { BootstrapVue } from "bootstrap-vue";
9  import "bootstrap/dist/css/bootstrap.css";
10 import "bootstrap-vue/dist/bootstrap-vue.css";
11 Vue.use(BootstrapVue);
12
13 new Vue({
14   router,
15   store,
16   render: h => h(App)
17 }).$mount('#app')
```

router 정의

- /
 - main 화면
- /todos
 - jsonplaceholder의 todos를 보여주는 화면
- /todos/:id
 - jsonplaceholder의 todos/:id를 보여주는 화면

views 폴더 및 파일 생성

- Main.vue
 - Main 화면입니다. 작성
- todos 폴더
 - List.vue
 - <div>List입니다</div> 작성
 - Detail.vue
 - <div>Detail입니다</div>



만든 파일을 기반으로 라우터 생성

- 미리 정의한 라우터 + /views 내부 파일로 라우터를 정의
- router/index.js 수정

```
import Vue from 'vue'
import VueRouter from 'vue-router'

import Main from "../views/Main.vue"
import List from "../views/todos/List.vue"
import Detail from "../views/todos/Detail.vue"

Vue.use(VueRouter)

const routes = [
  {
    path: "/",
    name: "Main",
    component: Main
  },
  {
    path: "/todos",
    name: "List",
    component: List
  },
  {
    path: "/todos/:id",
    name: "Detail",
    component: Detail
  },
]
```

utils/axios.js

- src에 utils 폴더생성
 - axios.js 생성
 - axios.create
 - 해당 request 요청을 보낼때
정의된 내용을 바탕으로 요청이 보내진다.
 - ex) request.get("/todos")
 - axios.get("https://jsonplaceholder.typicode.com/todos")와 동일

```
1  import axios from "axios";
2
3  const request = axios.create({
4    |    baseUrl : "https://jsonplaceholder.typicode.com"
5  |  })
6
7  export const api = {
8    |    jsonplaceholder:{
9    |      |    findAll:() => request.get("/todos"),
10   |      |    findOne:(id) =>request.get(`/todos/${id}`)
11   |      |  }
12   |    }
13  }
```

라우터 테스트

- 정의한 라우터에 잘 도착하는지 테스트를 진행한다.
- `http://localhost:8080/`
- `http://localhost:8080/todos`
- `http://localhost:8080/todos/1`

App.vue

- 로딩 관련 b-spinner를 삽입한다.
- App.vue 에 있기때문에 모든 라우터마다 로딩화면이 보이게 된다.
- 각 라우터 별로 로딩화면이 보이는지 테스트

```
<template>
  <div>

    <router-view/>

    <div class="spinning" >
      <b-spinner variant="primary" label="Spinning"></b-spinner>
    </div>
  </div>
</template>

<style>
.spinning{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
</style>
```

store/index.js

- 로딩 화면을 다루기 위한 state 및 mutations 정의

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    loading: false
  },
  getters: {
  },
  mutations: {
    SET_LOADING(state, data){
      state.loading = data;
    }
  },
  actions: {
  },
  modules: {
  }
})
```

App.vue

- loading= true인 경우에만 보일수있도록 변경시키기

```
<template>
  <div>

    <router-view/>

    <div class="spinning" v-if="$store.state.loading">
      <b-spinner variant="primary" label="Spinning"></b-spinner>
    </div>

  </div>
</template>

<style>
.spinning{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
</style>
```

/ (views/Main.vue)

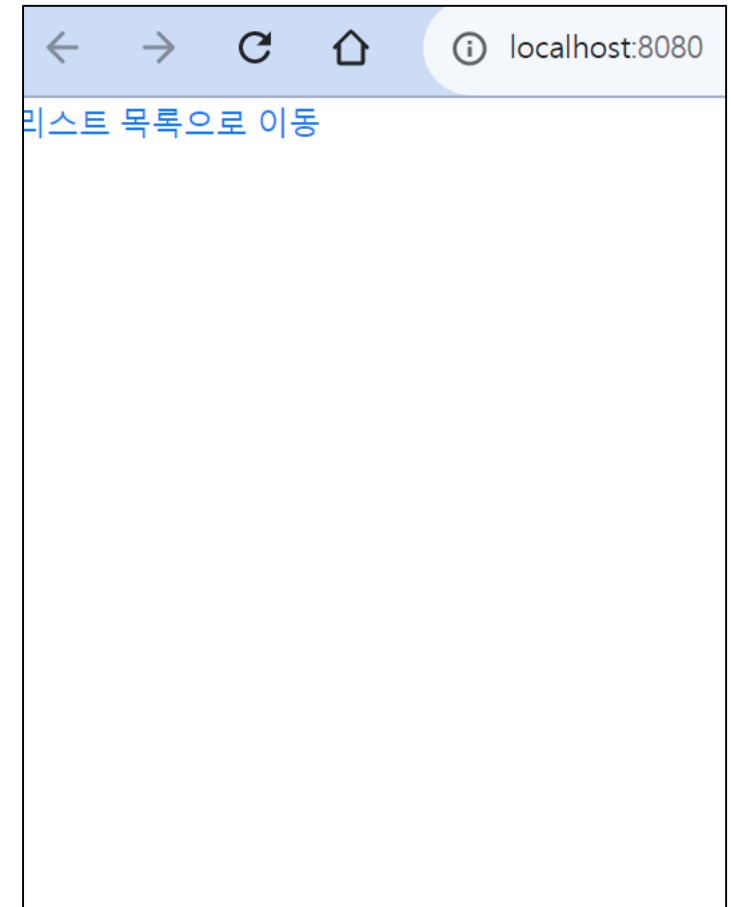
- 단순 router-link 추가

```
<template>
  <div>
    메인화면입니다.
    <router-link to="/todos">
      | 리스트 목록으로 이동
    </router-link>
  </div>
</template>

<script>
export default {
}
</script>

<style>

</style>
```



/todos (views/todos/List.vue)

- utils/axios.js에 정의한 api 호출 및 테스트
- axios 요청의 data값을 lists에 넣는다

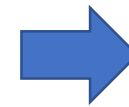
```
<template>
  <div>

  </div>
</template>

<script>

import { api } from "../../utils/axios"
export default {

  data(){
    return {
      lists: []
    }
  },
  async created(){
    const result = await api.jsonplaceholder.findAll();
    console.log(result);
    this.lists = result.data;
  }
}
</script>
```



```
▼ Object i
  ► config: {transiti
  ▼ data: Array(200)
    ► [0 ... 99]
    ► [100 ... 199]
    length: 200
```

/todos (views/todos/List.vue)

- 로딩 화면 활용
 - create로 요청 하기전
SET_LOADING true를 통해 loading 활성화
 - await 요청으로 데이터 요청이 끝나게 되면
SET_LOADING false를 통해 loading 비활성화

```
<template>
  <div>

  </div>
</template>

<script>
import { api } from "../../utils/axios";
export default {
  data() {
    return {
      lists: [],
    };
  },
  async created() {
    this.$store.commit("SET_LOADING", true);
    const result = await api.jsonplaceholder.findAll();
    console.log(result);
    this.lists = result.data;
    this.$store.commit("SET_LOADING", false);
  },
};
</script>
```

/todos (views/todos/List.vue)

- lists를 활용해서 v-for로 나타내기

```
<template>
  <div>

    <div v-for="list in lists" :key="list.id" class="list-wrapper">
      <div>
        {{list.title}}
      </div>
    </div>

  </div>
</template>

<script>

import { api } from "../../utils/axios"
export default {

  data(){
    return {
      lists: []
    }
  },
  async created(){
    this.$store.commit("SET_LOADING",true)
    const result = await api.jsonplaceholder.findAll();
    console.log(result);
    this.lists = result.data;
    this.$store.commit("SET_LOADING",false)
  }
}
</script>

<style>
.list-wrapper{
  border:1px solid black;
  padding:20px;
  cursor: pointer;
}
</style>
```

← → ↺ 🏠 ⓘ localhost:8080/todos
delectus aut autem
quis ut nam facilis et officia qui
fugiat veniam minus
et porro tempora
laboriosam mollitia et enim quasi adipisci quia provident illum

/todos (views/todos/List.vue)

- MoveDetail 메서드 추가
 - list.id값을 전달
 - /todos/:id 형식으로 넘겨준다.

```
<template>
  <div>
    <div
      @click="moveDetail(list.id)"
      v-for="list in lists"
      :key="list.id"
      class="list-wrapper"
    >
      <div>
        {{ list.title }}
      </div>
    </div>
  </div>
</template>

<script>
import { api } from "../../utils/axios";
export default {
  data() {
    return {
      lists: [],
    };
  },
  async created() {
    this.$store.commit("SET_LOADING", true);
    const result = await api.jsonplaceholder.findAll();
    console.log(result);
    this.lists = result.data;
    this.$store.commit("SET_LOADING", false);
  },
  methods: {
    moveDetail(id) {
      this.$router.push(`/todos/${id}`);
    },
  },
};
</script>
```

/todos/:id (views/todos/Detail.vue)

- route의 params 접근
 - this.\$route.params 에 :id값이 담긴다

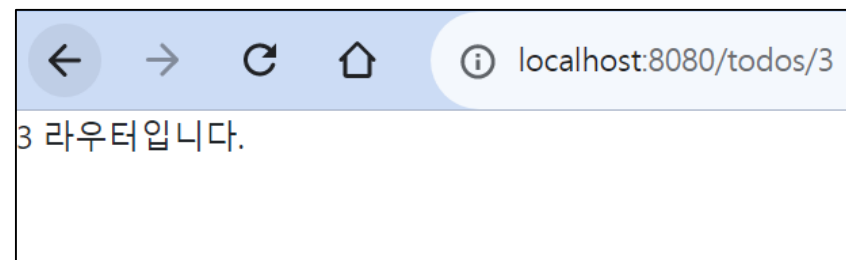
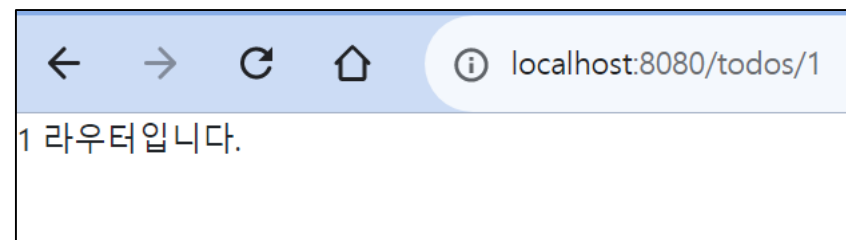
```
<template>
  <div>
    | | {{$route.params.id}} 라우터입니다.
  </div>
</template>

<script>
export default {

}
</script>

<style>

</style>
```



/todos/:id (views/todos/Detail.vue)

- \$route.params.id 활용
 - \$route.params.id로 axios 요청
 - 요청의 전과 끝에 loading 조작

```
<template>
  <div>
    {{ $route.params.id }} 라우터입니다.
  </div>
</template>

<script>
import { api } from '../utils/axios'
export default {
  data() {
    return {
      data: {}
    }
  },
  async created() {
    this.$store.commit("SET_LOADING", true);
    const result = await api.jsonplaceholder.findOne(this.$route.params.id);
    console.log(result);
    this.data = result.data;
    this.$store.commit("SET_LOADING", false);
  }
}
</script>
```

/todos/:id (views/todos/Detail.vue)

- data 활용
 - this.data에 담긴 data로 detail 화면을 꾸며준다.

```
<template>
  <div>
    <div>제목: {{data.title}}</div>
    <div>완료 여부: {{data.completed}}</div>
  </div>
</template>

<script>
import { api } from '../utils/axios'
export default {
  data(){
    return{
      data:{}
    }
  },
  async created(){
    this.$store.commit("SET_LOADING", true);
    const result = await api.jsonplaceholder.findOne(this.$route.params.id);
    console.log(result);
    this.data = result.data;
    this.$store.commit("SET_LOADING", false);
  }
}
```

```
▼ data:
  completed: false
  id: 3
  title: "fugiat veniam minus"
  userId: 1
```

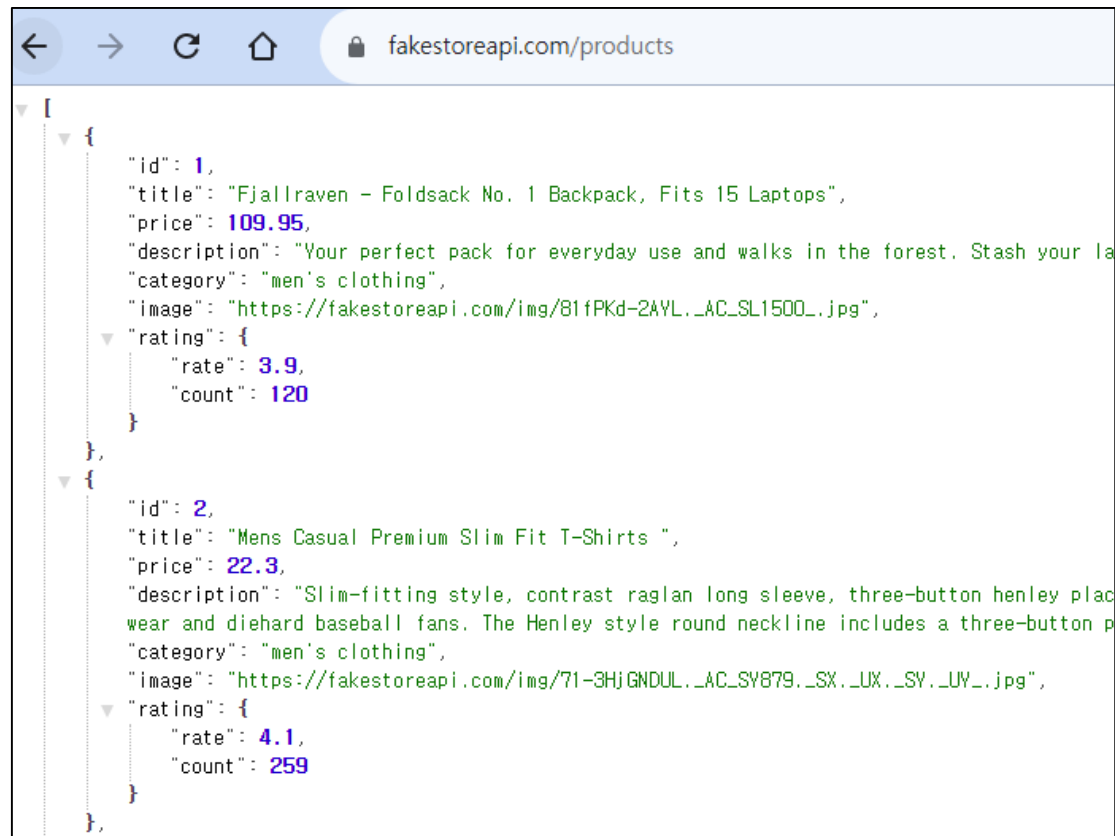
← → ↺ 🏠 ⓘ localhost:8080/todos/3

제목: fugiat veniam minus

완료 여부: false

쇼핑몰 사이트 구현 프로젝트

- <https://fakestoreapi.com>
 - 최대 20개의 products를 가져오는 TEST API
 - <https://fakestoreapi.com/products>
 - list를 가져온다
 - <https://fakestoreapi.com/products/:id>
 - id에 해당되는 produc를 가져온다
 - ex) <https://fakestoreapi.com/products/20>
- 요구 기능을 따르되 그외는 자율 구현



요구 기능

- 로딩 화면 구현
 - vue-bootstrap 활용
- 라우터
 - /
 - `https://fakestoreapi.com/products` 의 Data를 보여준다.
 - 특정 리스트 클릭시 그에 맞는 id로 라우터 이동
 - `/:id`
 - `https://fakestoreapi.com/products/:id` 의 Data를 보여준다.

내일 방송에서 만나요!

삼성 청년 SW 아카데미