Day1 Vue를 시작하자.

자료	Vue
구분	Vue
과목	

1. Hello Vue

먼저, 바탕화면에 빈 폴더를 하나 생성하고 vscode로 열어보자. (디렉터리(폴더) 이름은 영문으로 만들자.)

그리고 다음과 같이 입력해 프로젝트를 생성한다.

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

npm 은 Node Package Manager의 약자로 전 세계 오픈소스 자바스크립트 패키지를 모아 놓은 저장소다.

프로젝트 이름을 정한 후 엔터를 누른다.

```
Need to install the following packages:
create-vue@3.16.4
Ok to proceed? (y) y
```

```
Vue.js - The Progressive JavaScript Framework

// Yes 선택 (스페이스바)

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


// No 선택

◆ Install Oxlint for faster linting? (experimental)
```

```
| ○ Yes / ● No

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

프로젝트 폴더가 새로 생성 되면서 Vue.js 로 프로젝트를 진행 하면서 필요한 최소한의 파일들이 생성된다.
그리고 프로젝트 폴더를 열어보면 `vue-project / package.json` 파일이 있는데 클릭해서 내용을 눈으로 살펴보자.

이는 이 프로젝트의 간단한 정보 및 의존성을 명시해 놓은 것이다.
아래 정보는 참고로 그렇구나.. 하고 넘어가도 좋다. 절대 기억하거나 외울 필요가 없다.

```
{
  "name": "vue-project", // 프로젝트 이름이 vue-project 이다.
  "version": "0.0.0",
  "private": true, // 이 프로젝트가 공개 패키지가 아니라는 의미
  "type": "module",
  "scripts": {    // 프로젝트에서 실행 할 수 있는 명령어들을 정의하는곳
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint . --fix",
    "format": "prettier --write src/"
  },
  "dependencies": {    // 의존성을 말하며 프로젝트 안의 앱들을 실행시 필요한 패키지 들이다.
    "pinia": "^3.0.1",
    "vue": "^3.5.13",
    "vue-router": "^4.5.0"
  },
  "devDependencies": {    // 개발할때만 필요한 패키지들이 열거되는 파트다.
    "@eslint/js": "^9.22.0",
    "@vitejs/plugin-vue": "^5.2.3",
    "@vue/eslint-config-prettier": "^10.2.0",
    "eslint": "^9.22.0",
    "eslint-plugin-vue": "~10.0.0",
    "globals": "^16.0.0",
    "prettier": "3.5.3",
    "vite": "^6.2.4",
    "vite-plugin-vue-devtools": "^7.7.2"
  }
}
```

- `vue-project/` 디렉터리로 이동하자.

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

- 이제 `package.json` 에 명시 되어 있는 의존 패키지 들을 설치 할 것이다.

```
$ npm install
```

`npm install` 을 통해서

`vue-project/package.json` 의 `dependencies` , `devDependencies` 에 기록된 패키지 들을 설치했다.

그럼 node_modules 라는 폴더가 생성 될 것이다.

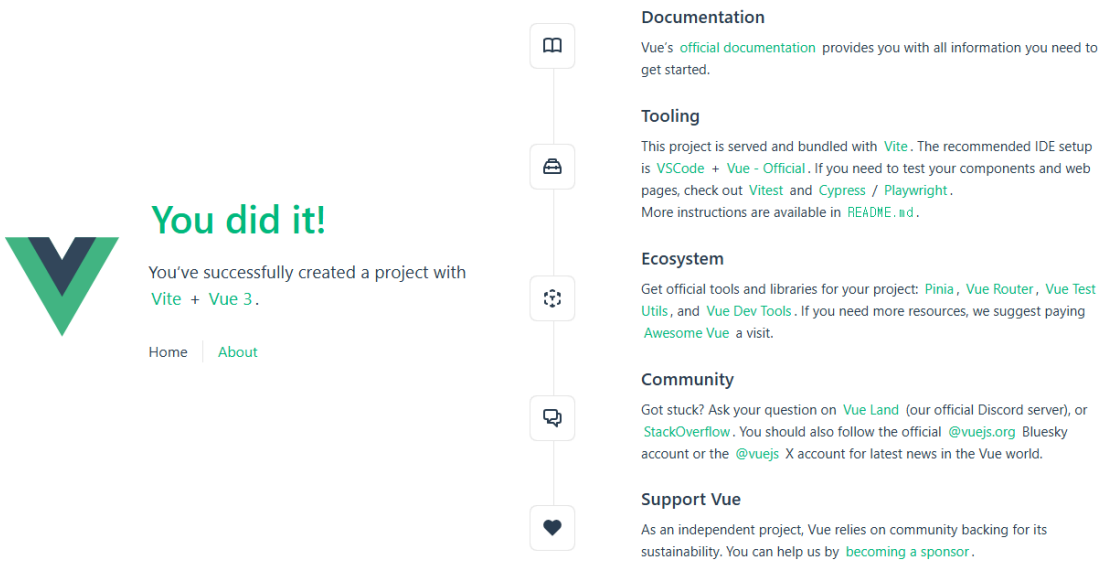
- 그리고 프론트엔드 서버를 구동 해 보자.

```
$ npm run dev
```

세 가지 명령어를 그대로 따라 치면, 다음과 같은 화면이 나온다.

```
VITE v6.3.5  ready in 3857 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  Vue DevTools: Open http://localhost:5173/__devtools__/ as a separate window
```

`http://localhost:5173` 으로 접속하면 다음과 같다.



이것은 프론트엔드 서버이다.

우리가 vs-code를 통해서 프로젝트 코드를 수정하면 즉각 이 화면에 반영된다.

`ctrl + c` 는 서버를 종료할 때 사용하면 된다.

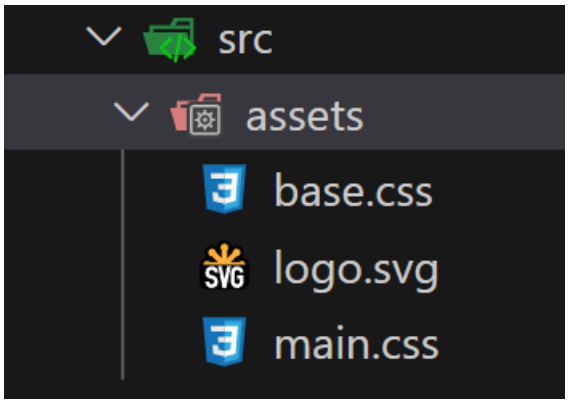
[참고] `package.json` vs `package-lock.json` 차이

항목	`package.json`	`package-lock.json`
역할	어떤 패키지를 쓰는지 **기본 정보와 의존성 버전 범위**를 기록	실제 설치된 **정확한 패키지 버전과 구조**를 기록
사람이 편집?	✔️ 사람이 직접 작성·편집함	❌ npm이 자동으로 생성·관리함
버전	`^1.5.0` → "1.5.0 이상 ~ 2.0.0 미만" 같은 **범위 지정**	`"version": "1.5.2"` 처럼 **정확한 버전 명시**
사용 목적	프로젝트 설정, 의존성 선언, 스크립트 정의 등	동일한 의존성 구조로 설치되도록 보장
Git에 포함?	✔️ 반드시 포함	✔️ 반드시 포함 (버전 통일 위해)

그리고 vite.config.js 는 vue앱을 실행(빌드)시키 위한 “비트(vite)” 라는 도구에 관한 설정이다.

우리의 프로젝트를 함에 있어서 필요 없는 파일을 삭제하자.

그다음, `src/` 에서 필요 없는 파일들을 삭제하겠다.



`src/assets/` 디렉터리부터 정리하겠다.

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

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

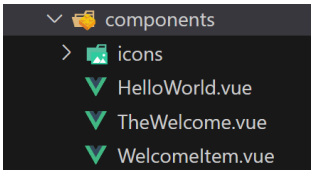
- `main.css` 는 전역 스타일링을 하기 위한 파일이다. (프로젝트 전 범위에 적용)
- `box-sizing` 은 박스의 크기를 화면에 표시하는 방식을 변경하는 속성이다.

테두리가 있는 경우에는 테두리의 두께로 인해서 원하는 크기를 찾기가 어려운데, `box-sizing` 속성을 `border-box`로 지정하면 테두리를 포함한 크기를 지정할 수 있기 때문에 개발을 하면서 크기를 예측하기가 더 쉬워서 넣어주곤 한다. (옵션)

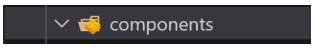
- `margin: 0` 은 각 태그에 기본적으로 잡혀있는 공백을 제거해줬다.

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

[삭제전]

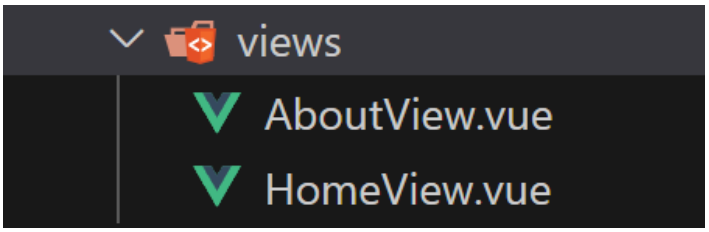


[삭제후]



그다음, `src/views/` 폴더를 클릭 해보자.

해당 폴더 아래의 `HomeView.vue` 와 `AboutView.vue` 를 다음과 같이 변경한다.




```
<!-- views/AboutView -->

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

```
<!-- views/HomeView -->

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

마지막으로, `src/App.vue` 를 다음과 같이 변경한다.

App.vue

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

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

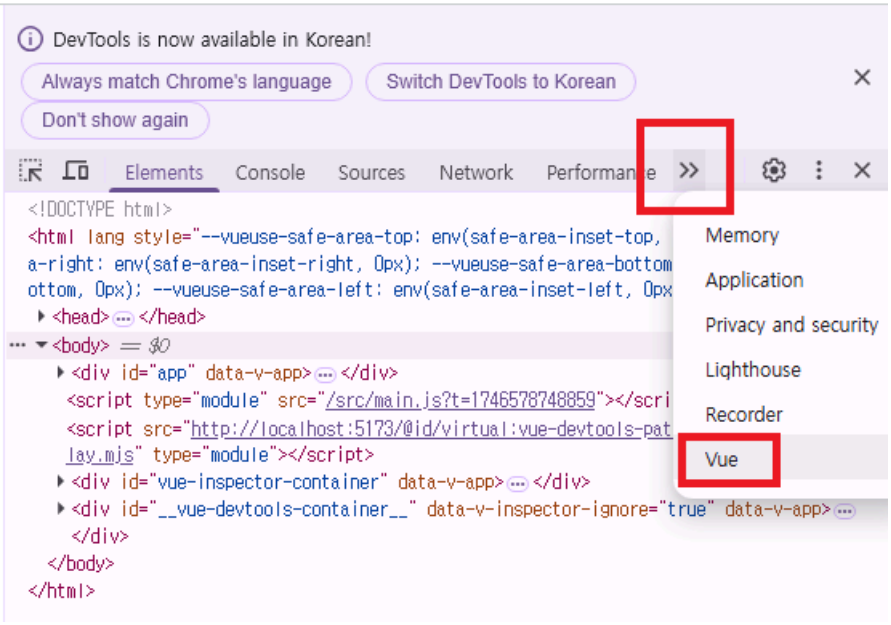
여기까지 완료한 후, 서버를 동작 시킨다.

```
$ npm run dev


VITE v4.5.0  ready in 383 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

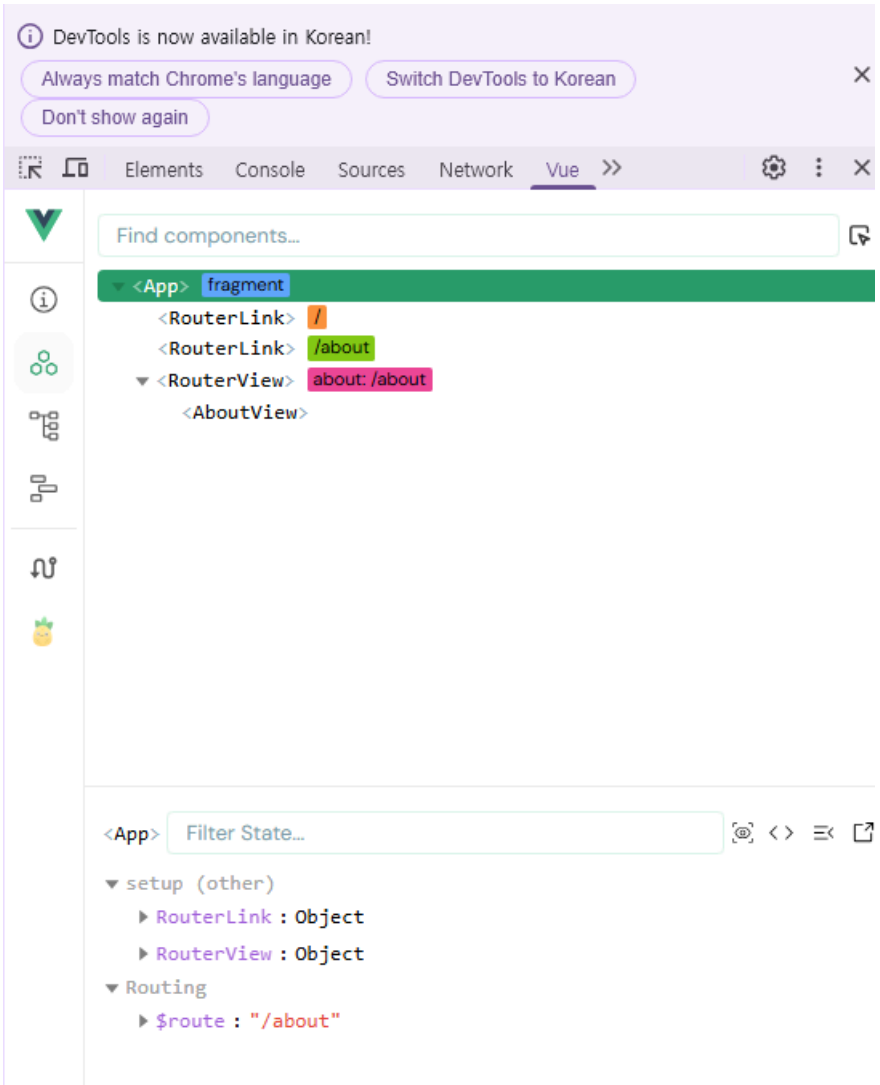
`localhost:5173` 로 접속해보자.

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

`F12` 버튼을 눌러보자.



화살표 `>>` 를 눌러, `Vue` 가 보이는지 확인한다.



앞으로 우리는 특정 시기에 이르기 전까지는

거의 `src/views/HomeView.vue` 를 변경해가며 실습을 진행할 것이다.

위 코드의 대략적인 구조만 설명하자면 먼저 App.vue 파일부터 보자.

App.vue는 무엇인가?

`App.vue`는 SPA(Single Page Application)에서 뼈대 (루트 컴포넌트)의 역할을 한다.

즉,

- 이 안에서 전체 레이아웃(헤더, 푸터, 네비게이션 등)을 구성한 다음
- 필요한 자식 컴포넌트들을 불러와서 조립하며 (components 폴더 안에 있는것들)
- `router-view` 등을 통해 페이지처럼 보이는 화면 전환도 담당하는 부분이다.

즉, `App.vue`는 SPA에서 하나의 주요 HTML 구조를 담당하는 Vue 컴포넌트인 것이다.

2. `{{ }}` (mustache)

`{{ }}` 라는 기호는 콧수염처럼 생겼다고 해서 mustache 라고 부른다.

일단은 무작정 따라서 다음 코드를 입력해보자.

```
<!-- views/HomeView -->

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

const message = ref("Hello Vue");
</script>

<template>
  <div>message: {{ message }}</div>
</template>
```

`HomeView.vue` 처럼, `.vue` 확장자가 붙은 파일을 하나의 "컴포넌트"라고 부른다.

컴포넌트는 파스칼 케이스로 작성하는 것이 관례이다.

PascalCase는 첫 글자가 대문자, 단어마다 대문자로 작성하는 표기법을 말한다.

컴포넌트 안에 들어가는 소스코드의 구조는 다음과 같다.

```
<script setup> // js code 가 들어감

</script>

<template> <!-- html code 가 들어감 -->

</template>

<style scoped> /* css code 가 들어간다 */

</style>
```

이 중, 당장은 스타일링을 할 필요가 없으니 CSS 를 담당하는 `<style>` 은 잠시 생략 한다.

template 에는 화면에 나올 HTML 코드를 작성하는 부분이다.

script 에는 자바스크립트 또는 타입스크립트 등을 이용해 스크립트 코드를 작성한다.

우리는 script를 통해서 템플릿에서 사용할 state 라고 하는 변수를 조작 할 수 있다.

앞에서 Hello Vue 를 출력 했던 코드를 보자.

```
<!-- views/HomeView -->

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

const message = ref("Hello Vue");
</script>

<template>
  <div>메시지: {{ message }}</div>
</template>
```

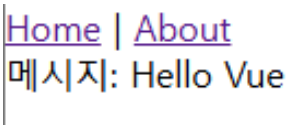
`ref` 객체는 state (상태)라고 불리는 특별한 변수를 생성할 때 사용하며,

`ref` 객체를 사용을 위해서는 `vue` 에서 import 해야 한다.

const message = ref("Hello Vue"); 를 보면

`message` state 의 값은 `ref` 객체의 인자값으로 들어가는 값이다.

- state 는 특별한 변수다. 보통의 변수 값이 바뀐다고 해서 화면 내용이 바뀌진 않는다. 그러나 state 가 변하면 화면에서 해당 state 와 연결된 부분이 다시 그려지게 된다.
- 화면을 그린다는 표현을 프론트엔드 개발에선 렌더링이라고 한다.
- 즉 state 변수의 값이 바뀌면 화면에 랜더링 되는 것이 바뀐다는 것이다.
- 앞으로 일반 변수와 구분하기 위해, state를 변수라고 표기하지 않고 state 라고 언급 할 것이다.



화면에 출력 되는 것을 보면, `{{ message }}` 는 `const message` 로 선언된 state 값인 것을 알 수 있다. 즉, `{{ }}` 는 state 의 값을 가져오는 역할을 한다.

[참고] HTML 은 프로그래밍 언어가 아니라 마크업 언어이다.

즉, HTML 에서는 변수, if, for 등을 사용할 수 없다. 그러나 Vue.js 는 HTML 안에서 변수

(정확하게는 state를) 사용 함으로써 if 와 for 와 같은 문법을 사용해서 화면에 보여지는 것을

손쉽게 컨트롤 할 수 있게 된다.

한 가지만 더 살펴보자.

방금 작성했던 코드에서는 `<script setup>` 이라고 <script> 태그 안에 setup 이라고 작성 해 주었는데 이는 라이브 교재(교안)에 있는 `setup()` 함수 대신 사용해 준 것이다.

`<script setup>` 은 Vue 컴포넌트의 `setup()` 함수를 더 간단하게 작성할 수 있도록 도와주는 기능으로 Vue3 공식 문서 에서도`setup()` 보다는 `<script setup>`을 사용하라고 권장하고 있다.

하지만 이건 공식문서의 입장일 뿐이다. 실제로`<script setup>`을 가지고 개발을 해보면 개발자가 작성한 javascript 코드를 아직 까지는 100% 완벽하게 인식을 못한다.

예를 들면 `<script setup>` 기능은 Javascript의 this 키워드를 사용해서 인스턴스에 접근이 불가능 하다. 뿐만 아니라, Vue3 이상의 버전에서만 사용할 수 있으며 일부 기능, 예를 들면 data, method, computed 와 같은 Options api 사용은 제한될 수 있다는 단점이 있다.

Vue2 버전에서 Vue3로 바뀐 역사는 그리 오래되지 않았다. 그러한 이유로 ssafy 교안 에서는 범용성을 생각해서 `setup()` 컴포지션 함수(Composition API)를 채택해서 사용 한 것 같다는 것이 내 생각이다.

그러나 `<script setup>` 는 javascript가 아닌 typescript 를 사용함에 있어서 아주 좋은 setup() 함수의 대안이라서 앞으로 조금 더 지켜봐야 할 부분이다.

위 내용은 아 그렇구나.. 하고 넘어가자. 아직까지 현업에서는 vue2로 구현된 서비스가 많아서 vue2랑 어떤 점이 다른지도 알고는 있어야 한다고 생각 되어서 적어 보았다.

참고로 vue2 버전에서는 data, method, computed를 사용하는 Options api 라는 것 만 존재 했고 setup()과 같은 Composition api는 vue3에서 생긴 문법이다.

참고로 vue.js 코드를 작성을 하더라도 다양한 방식으로 작성이 가능하다.

- Vue 3 코드 작성 방법

1. `<script setup>` 이용한 방식 (Composition API 최신 스타일)
2. `setup()` 함수를 이용한 방식 (Composition API 기본형)

- Vue 2 코드 작성 방법

3. `data, method, computed` 를 이용한 방식 (Options API 방식 (전통적 방식))

우리가 앞으로 사용을 할 `setup()` 함수는 `<script setup>` 와 함께 Vue3에서 새로 도입된 API 다.

[Tip] 나중에 블로그 검색하다가 찾은 `<script setup>` 형식으로 작성한 vue 코드는
GPT한테 `setup()` 함수의 형식으로 바꿔달라고 하면 기가 막히게 잘 바꿔준다.
또는 Vue2 코드를 Vue3로 아주 뛰어나게 잘 변환 해준다. 엄청 편하다 ㅎㅎ

추가적으로, 한가지만 더 설명하자.
지금까지 해당 PDF를 따라서 한 vue 실습방식은 vue.js를 npm 방식으로 설치 했다.
하지만, 이제부터는 SSAFY 교안에 있는대로 cdn 방식으로 먼저 실습을 진행 하도록 하겠다.
나중에는 결국 npm 방식으로 넘어 간다.

[참고] NPM방식 CDN방식이 무엇?

Vue에서 Vue.js를 프로젝트에 추가하는 두 가지 주요 방법으로는 NPM 방식과 CDN방식 이 있다.

1. npm 이란?
npm은 node package manager의 줄임말이다.
자바스크립트의 라이브러리를 관리해주는 도구라고 생각하면 된다.
전체적인 자바스크립트 라이브러리를 npm을 통해서 관리할 수 있다.

2. cdn 이란?
CDN은 Contents Delivery Network의 약자로 물리적으로 멀리 떨어져 있는 사용자에게
특정 컨텐츠를 빠르게 제공할 수 있는 기술을 의미한다.

CDN 방식은 Script로딩에 되는데 네트워크가 느려서 페이지가 로딩하는데
약간의 기다리는 시간이 발생함으로 NPM방식보다는 조금 더 느리다.

또한 CDN방식은 CDN서버가 터지거나 CDN 버전이 크게 바뀌어 기존 버전이 버려지는 경우에는
해당 CDN을 이용하여 개발한 서비스를 더이상 제공을 할 수 없게 된다.

따라서 간단한 작업이나 테스트용 페이지를 개발 할 경우 CDN방식을 이용하며,
크고 복잡한 앱을 개발 할 때에는 NPM방식을 사용한다.

미안하지만 지금까지 작석했던 코드는 다 지우자. ㅎㅎ (폴더를 지워버리자)

바탕화면에 새로 폴더 만들어서 오늘은 ssafy 교안의 방식대로 (setup 함수방식)

빈 html 파일을 새로 만들어서 실습 해 보자. 😊

빈 디렉터리(폴더)를 생성하고 빈 html 파일을 하나 만들자.

그리고 아래 코드를 따라서 작성해보자.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- cdn 복붙하기 -->
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

  <div id="app">
    <h1>{{ message }}</h1>
  </div>

  <script>
    const { createApp, ref } = Vue;

    const app = createApp({
      setup() {
        const message = ref('hello Vue');

        return {
          message
        };
      }
    });

    app.mount('#app');
  </script>
</body>
</html>
```

앞으로 가능한 한글사용은 자제하자.

setup 함수를 구성하는 내용은 javascript처럼 스크립트 태그를 하나 만들어서 작성하면 된다.

setup 함수는 객체를 반환하는데 이 객체에는

javascript의 이벤트핸들러 함수 그리고 화면을 담당하는 HTML에서 사용할 변수들이 들어 간다.

위의 코드에서 script 파트에 주석을 달면 다음과 같을 것이다.

```
<script>
  const {createApp, ref}=Vue // 구조 분해 할당
  // 1. 속성명 축약한 것이다.
  // const createApp = Vue.createApp
  // const ref = Vue.ref

  const app=createApp({
    setup(){ // ref 객체를 사용하기 위한 함수

      const message=ref('hello Vue') // ref 객체를 이용해 message를 ref 객체로 만듬

      return { // template에 전달할 데이터 반환
            // 반환하는 객체 안에는 javascript의 이벤트핸들러 함수 그리고(또는)
            // 화면을 담당하는 HTML에서 사용할 state변수가 반환
        message
      }
    }

  })
  app.mount('#app') // id가 app인곳에 랜더링(붙이기)
</script>
```

자. 이제 `message` 의 state 를 변경해 보는것을 실습해보자

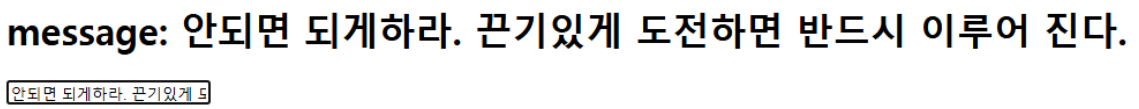
3. v-model

`v-odel` 은 `<input>` 태그에 사용된다. 코드를 다음과 같이 수정해보자.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

  <div id="이채은교육생이졸지않기를기원합니다">
    <h1> message:{{ hello }}</h1>
    <input type="text" v-model="hello">
```

```
  </div>

  <script>
    const { createApp, ref } = Vue

    const app = createApp({
      setup() {
        const hello = ref("")

        return {
          hello
        }
      }
    })
    app.mount('#이채은교육생이졸지않기를기원합니다')
  </script>
</body>
</html>
```

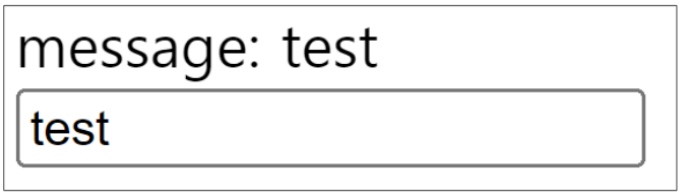
`message` 를 빈 문자열로 `""` 두었고, `<input>` 태그에 `v-model="hello"` 를 달았다.
브라우저로 확인해 보자.



입력할 때마다 메세지가 바뀌어서 랜더링이 되는 신기한 현상을 경험할 수 있다.
즉, `v-model` 은 사용자 입력값을 state 에 실시간 저장할 때 사용한다.

[참고] <script setup> 옵션 코드는 눈으로만 봐주세요.

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

const message = ref("");
</script>

<template>
  <div>message: {{ message }}</div>
  <input type="text" v-model="message" />
</template>
```



`v-model` 은 모든 `<input>` 태그에서 사용이 가능하다.

체크박스도 물론 활용 가능하다. 해보자.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- cdn 복붙하기 -->
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

  <div id="app">
    <h1>message: {{ message }}</h1>
    <input type="text" v-model="message">

    <p>
      <input type="checkbox" id="checkbox" v-model="checked">
      <label for="checkbox">{{checked}}</label>
    </p>

  </div>

  <script>
    const { createApp, ref } = Vue

    const app = createApp({
      setup() {
        const message = ref('')
        const checked = ref(false)

        return {
          message,
          checked,
        };
      }
    });
```

```
  app.mount('#app')
 </script>
</body>
</html>
```

[<script setup> 옵션은 참고용으로만 봐주세요]

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

const checked = ref(false);
</script>

<template>
 <input type="checkbox" id="checkbox" v-model="checked" />
 <label for="checkbox">{{ checked }}</label>
</template>
```

클릭 할 때마다 `true` , `false` 가 변경 되는 것을 알 수 있다.

- `v-model` 은 `<input>` 태그에서 쓰이고, `<input>` 태그가 나오면 무조건 `v-model` 을 사용한다. 이 원칙은 변하지 않으며, 추후 배우게 될 `v-bind` 를 `<input>` 태그에 사용하는 일은 없도록 하자.

==========

4. `@` (`v-on`)

이번에는 이벤트를 받아 보자.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <!-- cdn 복붙하기 -->
```

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

<div id="app">
  <h1>message: {{ message }}</h1>
  <input type="text" v-model="message">

  <p>
    <input type="checkbox" id="checkbox" v-model="checked">
    <label for="checkbox">{{checked}}</label>
  </p>

  <p>
    <div>{{ text }}</div>
    <button v-on:click="changeText">클릭하면 글자가 변해!</button>
  </p>

</div>

<script>
  const { createApp, ref } = Vue;

  const app = createApp({
    setup() {
      const message = ref('');
      const checked = ref(false);
      const text = ref("???")
      const changeText=()=>{
        text.value="짜잔"
      }

      return {
        message,
        checked,
        text,
        changeText,
      };
    }
  });

  app.mount('#app');
</script>
</body>
</html>
```

[<script setup> 옵션은 참고용으로만 봐주세요]

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

const text = ref("???");

function changeText() {
 text.value = "짜잔";
}
</script>

<template>
 <div>{{ text }}</div>
 <button v-on:click="changeText">클릭하면 글자가 변해!</button>
</template>
```

이벤트를 받을 땐 `v-on` 으로 받으며, 콜론 `:` 뒤에 받을 이벤트명 `click` 을 적어준다.

```
<p>
 <div>{{ text }}</div>
 <button v-on:click="changeText">클릭하면 글자가 변해!</button>
</p>
```

이꼴 `=` 기호 다음에 이벤트 발생 시 실행할 함수를 콜백으로 달아주면 된다.

클릭하면 `changeText` 함수가 실행되며, `text` 는 `"???"` 에서 `"짜잔"` 으로 바뀐다.

```
const text = ref("???")
const changeText=()=>{
 text.value="짜잔"
}
```

- 함수 안에서 state 의 값에 접근하려면, 반드시 `.value` 를 붙여줘야 한다. 이 경우엔 `text.value` 로 접근한다.
- 그러나, `<template>` 안에선 `.value` 를 붙이지 않고 그냥 사용하면 된다.

`v-on:` 은 `@` 으로 축약이 가능한데, 다음과 같이 변경 가능하다.

```
<button v-on:click="changeText">클릭하면 글자가 변해!</button>

<button @click="changeText">클릭하면 글자가 변해!</button>
```

- 축약구문을 사용하든, 풀네임을 사용하든 개인의 자유 겠지만,

  프로젝트 전체에 축약을 사용할 것이라면 축약만, 풀네임을 사용할 것이라면 풀네임만 쓰도록 한다.
- 화살표 함수도 잘 작동하지만, Vue 공식문서에서 `function` 키워드를 사용하므로 되도록 function 키워드로 사용하자.

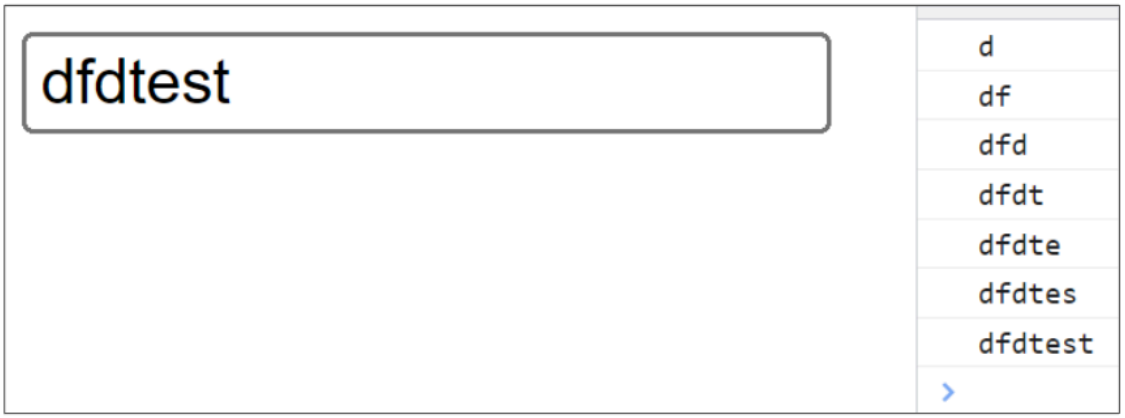
이벤트는 `click` 만 있는게 아니다. `keyup` 을 사용해보자.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- cdn 복붙하기 -->
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

  <div id="app">
    <h1>message: {{ message }}</h1>
    <input type="text" v-model="message">

    <p>
      <input type="checkbox" id="checkbox" v-model="checked">
      <label for="checkbox">{{checked}}</label>
    </p>

    <p>
      <div>{{ text }}</div>
      <button v-on:click="changeText">클릭하면 글자가 변해!</button>
    </p>

    <p>
      <input type="text" @keyup="go" />
    </p>
  </div>

  <script>
    const { createApp, ref } = Vue;

    const app = createApp({
      setup() {
        const message = ref('');
        const checked = ref(false);
        const text = ref("???")
```

```
      const changeText=()=>{
        text.value="짜잔"
      }

      function go(evt) {
        console.log(evt.target.value);
      }

      return {
        message,
        checked,
        text,
        changeText,
        go
      };
    }
  });

  app.mount('#app');
 </script>
</body>
</html>
```

```
<script setup>
function go(evt) {
 console.log(evt.target.value);
}
</script>

<template>
 <input type="text" @keyup="go" />
</template>
```

이벤트는 매개변수 `evt` 로 받는다.



사용자가 입력할 때마다, 콘솔이 찍히는것을 확인할 수 있다.

Vue.js에서 제공하는 이벤트는 여러가지 있다.
이벤트 목록은 다음에서 확인이 가능하다.
https://www.w3schools.com/jsref/dom_obj_event.asp

굉장히 많은 이벤트가 있으며, 처음 개발을 시작할 때에는 클릭만 제대로 익혀도
대부분의 개발은 가능하다.

오늘은 여기까지 보자. 😊