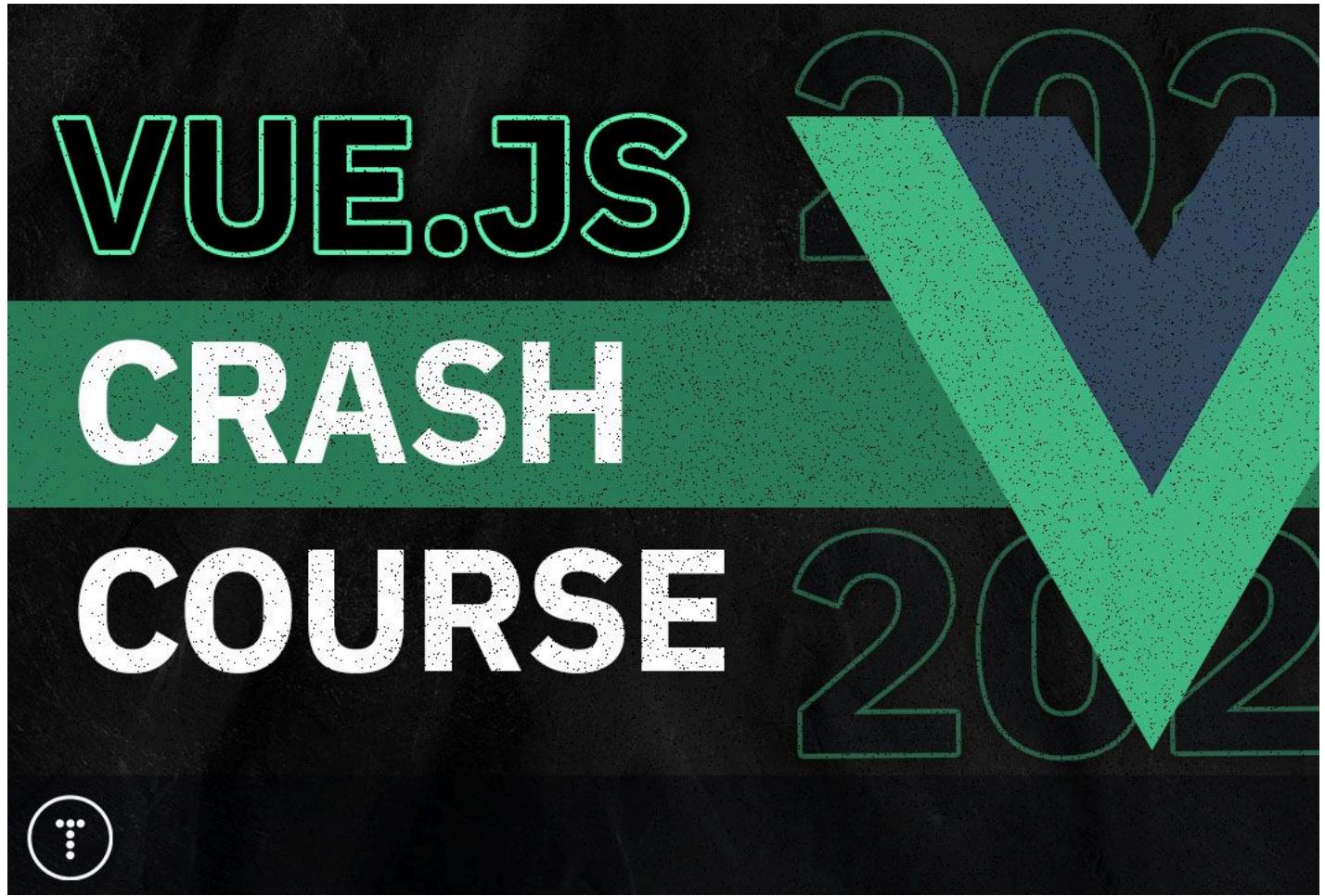


행동 전자책 프리랜서 마스터리 유튜브 자주 묻는 질문 블로그 가이드

로그인

Vue.js Crash Course(2024년 전체)

날짜 2024년 7월 1일



Vue.js 집중 강좌에 오신 것을 환영합니다. 이 강좌는 가능한 한 빨리 Vue.js를 시작하고 실행하도록 설계된 강좌입니다. 초보자를 대상으로 하므로 Vue.js를 사용해 본 적이 없거나 잠깐 사용해 본 적이 있다면 올바른 곳에 있습니다.

이 과정에는 몇 가지 다른 부분이 있습니다. 먼저, 몇 장의 슬라이드를 살펴보고 Vue가 무엇이고 어떻게 작동하는지 이야기하겠습니다. 그렇게 오래 걸리지는 않을 겁니다.

그런 다음 CDN을 사용하여 매우 빠르게 시작하고 실행하는 방법을 보여드리겠습니다. 이것은 Vue를 사용하는 가장 빠른 방법이지만 정말 작은 프로젝트와 테스트에만 사용해야 합니다. 더 큰 프로젝트의 경우 "create-vue"라는 것을 사용하는데, 궁극적으로 이것을 사용할 것입니다.

다음으로 create-vue를 설정하고 구성 요소, 지시문, 데이터, 메서드 등의 기본 사항을 살펴보겠습니다. 매우 기본적인 작업 프로젝트를 살펴보겠습니다. 앱이라고 부르지도 않겠지만, 그저 실험하고 다양한 지시문 등을 살펴보는 프로젝트일 뿐입니다.

마지막으로, 우리는 우리의 주요 구인 공고 웹사이트를 구축할 것입니다. Vue는 프런트엔드 프레임워크이므로, 우리는 데이터를 위한 백엔드가 필요하므로, 모의 REST API를 만들 수 있는 라이브러리인 JSON Server를 사용할 것입니다. 우리는 하드코딩된 일부 구인 공고로 시작하지만, 그런 다음 API에서 동적 데이터를 추가할 것입니다. 저는 여러분이 사용할 수 있도록 Tailwind 클래스가 있는 템플릿 파일을 리포에서 제공할 것입니다. 제가 최근에 수강한 React 충돌 과정을 수강했다면, 그것은 같은 애플리케이션입니다. 그것은 여러분이 두 프레임워크를 비교할 수 있도록 의도된 것입니다. 저는 또한 Angular와 Svelte 버전을 만들 것입니다.

최종 코드의 리포는 다음과 같습니다. <https://github.com/bradtraversy/vue-crash-2024> 전체 비디오 튜토리얼은 다음과 같습니다.
<https://youtu.be/VeNfHj6MhgA>

그러니 편안히 앉아 커피나 차를 마시면서 Vue.js JavaScript 프레임워크에 대해 배워보세요.

Vue.js란 무엇인가?

Vue가 무엇인지 정의하는 것으로 시작해 보겠습니다. Vue.js는 사용자 인터페이스와 단일 페이지 애플리케이션을 구축하는 데 사용되는 진보적인 **JavaScript** 프레임워크입니다. 간단하고 유연하며 점진적으로 채택할 수 있도록 설계되어 애플리케이션의 작은 부분에 사용을 시작한 다음 점진적으로 확장할 수 있습니다. Vue는 기존 웹페이지에 약간의 상호 작용을 추가하는 처음부터 복잡한 웹 애플리케이션을 구축하든 개발자가 모든 규모의 프로젝트에 쉽게 통합할 수 있도록 합니다. 반응형 데이터 바인딩과 구성 요소 기반 아키텍처를 갖춘 Vue.js는 개발자가 간단하고 효율적인 방식으로 사용자를 위한 동적이고 상호 작용적인 경험을 만들 수 있도록 돕습니다.

Vue.js는 2013년 Evan You가 만들었습니다. Evan은 현재 싱가포르에 거주하는 독립 개발자입니다. 적어도 그의 약력에서 알 수 있는 바는 그렇습니다. Vue.js가 Google이나 Facebook에서 만든 것이 아니라 여전히 인기가 있다는 사실이 프레임워크에 대해 많은 것을 말해준다고 생각합니다.

필수 조건

사람들은 항상 "언제 프레임워크를 배울 준비가 되었나요?"라고 묻습니다. 그 질문에 대한 답은 어렵습니다. 모든 사람이 서로 다른 학습 능력과 학습 방식을 가지고 있기 때문입니다. 저는 React나 Vue를 배우는 것과 동시에 JavaScript를 배운 사람들을 알고 있습니다. 하지만 저는 그것이 대부분의 사람들에게 최선이라고 생각하지 않습니다. 먼저 HTML, CSS, 그리고 JavaScript의 기본을 배워야 한다고 생각합니다. 그리고 저는 단순히 변수와 함수가 무엇인지 배우는 것을 말하는 것이 아닙니다. 프레임워크 없이도 어떤 종류의 프런트엔드 프로젝트를 빌드할 수 있을 정도로 배우는 것을 말합니다.

저는 보통 다음을 추천합니다.

- JS 기초 - 루프, 함수, 객체 등
- 이벤트, DOM 조작 등
- API 및 기본 HTTP 가져오기.
- 화살표 함수, 고차 배열 메서드, 구조 분해 등.

저는 traversymedia.com과 [Udemy](https://www.udemy.com)에서 **Modern JavaScript From The Beginning**이라는 코스를 운영하고 있습니다. JavaScript에 대해 알아야 할 모든 것과 더 많은 것을 제공합니다. 그러니 확인하고 싶으시다면 설명에 링크를 걸어두겠습니다.

또한 **NPM(Node Package Manager)**에 대해 알아보는 것이 좋습니다. 3자 패키지를 설치하고 **create-vue** 와 같은 Vue.js 프로젝트를 생성하는 도구를 설정하는 데 자주 사용되기 때문입니다.

프런트엔드 프레임워크의 역할

Vue에 대해 구체적으로 이야기하기 전에, 프런트엔드 프레임워크의 역할과 그것이 만들어진 이유에 대해 알아보겠습니다.

- **향상된 사용자 경험** - 프런트엔드 프레임워크는 개발자가 사용자 인터페이스를 더 쉽게 구축할 수 있도록 설계되었습니다. 이 프레임워크는 기본적으로 많은 기능을 제공하며 애플리케이션에 고유한 것에 집중할 수 있도록 합니다. vanilla JavaScript로 정말 대화형 인터페이스를 구축하려고 하면 정말 빨리 지저분해집니다. 할 수 없다고 말하는 것이 아니라 정말 어려울 수 있으며 바퀴를 새로 발명하는 것과 같습니다. - **구성** - 프런트엔드 프레임워크는 또한 개발자가 코드를 더 쉽게 구성할 수 있도록 설계되었습니다. UI는 구성 요소로 나뉘며 각 구성 요소에는 고유한 상태와 속성이 있습니다. 또한 협업에 필수적입니다. 5명의 개발자가 vanilla JavaScript로 인터페이스를 만들면 완전히 다른 5개의 프로젝트가 생깁니다. 프레임워크를 사용하면 모든 사람이 같은 페이지에 있을 수 있습니다.
- **성능** - 프런트엔드 프레임워크도 성능에 최적화되어 있습니다. 가상 DOM과 같은 많은 내장 기능이 있는데, 이는 매번 전체 DOM을 다시 렌더링하지 않고도 DOM을 렌더링하는 방법입니다. 이는 엄청난 성능 향상이며 사용자에게 정말 빠른 UI를 제공합니다.
- **모듈성** - 프런트엔드 프레임워크의 주요 이점 중 하나는 모듈성입니다. 개발자는 애플리케이션을 더 작고 재사용 가능한 구성 요소로 나눌 수 있습니다. 이 모듈식 접근 방식은 코드베이스를 더 관리하기 쉽고 이해하기 쉽게 만들 뿐만 아니라 재사용성을 촉진합니다.

그래서 그게 우리가 프런트엔드 프레임워크를 사용하는 몇 가지 이유일 뿐입니다. 항상 하나를 사용해야 한다고 말하는 것은 아닙니다. 사실, 사람들이 React나 Vue를 사용하여 매우 간단한 랜딩 페이지를 만드는 것을 많이 봅니다. 그렇게 하면 일이 너무 복잡해질 뿐입니다. 그래서 각 프로젝트를 개별적으로 살펴봐야 합니다.

왜 Vue인가?

그렇다면 React나 Angular 대신 Vue를 사용하는 이유는 무엇일까요?

- **단순성 및 접근성** - Vue.js는 단순성과 기존 프로젝트에 쉽게 통합할 수 있는 것으로 유명합니다. 매우 완만한 학습 곡선을 가지고 있으며 다양한 수준의 경험을 가진 개발자가 쉽게 사용할 수 있어 빠르게 시작하고 생산적으로 될 수 있습니다.
- **유연성** - Vue.js는 점진적으로 도입할 수 있도록 설계되어 프로젝트에 점진적으로 도입할 수 있습니다. 작은 위젯을 빌드하든 본격적인 단일 페이지 애플리케이션 (SPA)을 빌드하든 Vue.js는 다양한 프로젝트 요구 사항을 충족하도록 손쉽게 확장됩니다. Vue 메타 프레임워크로 서버 사이드 렌더링 및 정적 웹사이트를 빌드 할 수도 있습니다.
- **성능 및 크기** - Vue.js는 가상 DOM을 포함한 효율적인 렌더링 메커니즘 덕분에 뛰어난 성능을 제공합니다. 또한 Vue의 핵심 라이브러리는 가벼워서 초기 로드 시간이 더 빠르고 런타임 성능이 더 좋습니다. 가장 빠른 프런트엔드 프레임워크 중 하나로 알려져 있습니다.

- **구성 요소 기반 아키텍처** : 다른 최신 프레임워크와 마찬가지로 Vue.js는 구성 요소 기반 아키텍처를 촉진합니다. 구성 요소는 애플리케이션의 여러 부분에서 재사용할 수 있는 독립형 단위로, 유지 관리성과 코드 재사용성을 촉진합니다.
- **활동적인 커뮤니티와 생태계** : Vue.js는 정말 활기찬 커뮤니티와 라이브러리, 도구, 플러그인의 풍부한 생태계를 가지고 있습니다. 단일 페이지 애플리케이션 (SPA)을 빌드하는 것 외에도 Nuxt.js와 같은 메타 프레임워크가 있는데, 이를 통해 서버 사이드 렌더링(SSR)을 사용할 수 있고 Gridsome과 같은 프레임워크를 통해 정적 웹사이트를 빌드할 수 있습니다. 따라서 Vue는 React의 생태계와 유사한 훌륭한 생태계를 가지고 있습니다.

React나 Angular 대신 Vue.js를 선택하는 것은 종종 개인적인 선호도에 달려 있습니다. 업계, 특히 귀하의 지역에서 무엇이 사용되고 있는지도 고려해야 하지만, 저는 귀하가 작업하는 프레임워크를 즐겨야 한다고 생각합니다. 제 조언은 항상 관심 있는 프레임워크를 시도해 보는 것입니다. 그래서 저는 이러한 충돌 과정을 만들어서 여러분이 밤을 적시고 무언가를 만들어 보고 어떤 것이 여러분에게 정말 잘 맞는지 확인할 수 있도록 합니다. 그래서 저는 여러 프레임워크로 동일한 앱을 만드는 것을 좋아합니다. 각 프레임워크가 어떻게 작동하는지 실제로 느낄 수 있습니다.

Vue 구성 요소

다른 프런트엔드 JavaScript 프레임워크와 마찬가지로 Vue.js는 컴포넌트라는 개념을 중심으로 구축되었습니다. 컴포넌트는 재사용 가능하고 독립적인 코드 조각으로, 다른 프로젝트에 쉽게 적용할 수 있습니다. Vue 컴포넌트는 3개 부분으로 나뉜 매우 간단한 구조를 가지고 있습니다.

- **logic/js**는 상태나 데이터, 메서드, 이벤트, 가져오기 등을 정의하는 곳입니다.
- 렌더링될 HTML로 구성된 템플릿 출력이지만, 지시어라는 것을 사용하여 변수, 루프, 조건문과 같은 동적 요소를 템플릿에 포함할 수도 있습니다. 나중에 이에 대해 다루겠습니다.
- **CSS인 스타일입니다.** "범위 지정"을 추가할 수 있습니다. 즉, 스타일은 해당 특정 항목에만 적용됩니다.

다음은 Vue 구성 요소가 어떻게 포맷되고 구조화되는지에 대한 예입니다.

```
<script>
// Vue component definition
export default {
  name: 'SimpleComponent',
  // Optionally, you can include data, methods, computed properties, etc.
};

</script>

<template>
<div>
  <h2>Hello from Vue.js!</h2>
  <p>This is a simple Vue component.</p>
</div>
</template>

<style scoped>
/* Scoped CSS for this component */
h2 {
  color: #333;
}
p {
  font-size: 16px;
  line-height: 1.6;
}
</style>
```

구성 요소와 그 구조에 대해 이야기할 때 Vue.js에서 로직을 처리하는 방법에는 두 가지가 있다는 것을 아는 것이 중요합니다. 전통적인 방법 options API과 composition API. 옵션 API는 더 간단하고 소규모 프로젝트에 적합한 선택입니다. 그러나 구성 API는 Vue 3과 함께 출시되었으며 더 유연하고 더 복잡한 구성 요소를 만들 수 있습니다. 두 가지 모두의 예를 들어보겠지만 전반적으로 새로운 구성 API를 사용하고 싶습니다. 옵션 API를 정말 알고 싶다면 이전의 Crash Course도 살펴보세요. 두 가지 방법 모두 상태 데이터, 메서드 및 수명 주기 후크를 정의할 수 있습니다. 구성 요소 로드가 완료될 때와 같이 특정 시간에 특정 작업이 발생하도록 할 수 있습니다.

Vue.js 사용하기

Vue.js를 사용하는 방법은 여러 가지가 있습니다. 가장 일반적인 방법은 다음과 같습니다.

- **CDN** - 가장 쉬운 방법은 CDN을 사용하는 것입니다. HTML 파일에 Vue.js를 포함시켜 사용할 수 있지만, 매우 작고 간단한 프로젝트가 아닌 이상은 이것을 제안하지 않을 것입니다. 위젯이나 다른 것에는 어떨까요.
- **Vue CLI** - Vue CLI는 오랫동안 사용되었습니다. 프로젝트를 스캐폴딩하는 명령줄 인터페이스이며, 공식 플러그인과 통합의 풍부한 컬렉션을 포함합니다. 그러나 Vue CLI는 더 이상 새 프로젝트에 권장되지 않으며 유지 관리 모드에 있으므로 버그 수정과 보안 업데이트만 받게 됩니다. Vue CLI 웹사이트로 이동하면 더 이상 권장되지 않으며 **create-vue**를 사용하는 것이 좋습니다.

- **create-vue** - Create Vue는 Vite 웹 서버와 프런트엔드 도구를 사용합니다. 여기에는 핫 리로딩, TypeScript에 대한 기본 지원 및 기타 기능과 같은 기능이 포함됩니다. 또한 플러그인과 통합의 풍부한 생태계도 포함됩니다. 하나의 명령으로 프로젝트를 설정할 수 있으며, 이것이 우리의 구인 광고 앱에 사용될 것입니다.
- **Nuxt & Gridsome** - Vue를 사용하는 또 다른 방법은 메타 프레임워크를 사용하는 것입니다. React에 Next.js가 있는 것처럼 Vue에는 Nuxt.js가 있는데, 이를 통해 서버 사이드 렌더링 애플리케이션을 만들 수 있습니다. Gridsome은 Vue를 사용하는 정적 사이트 생성기입니다. 따라서 이러한 프레임워크에는 설정을 위한 자체 도구가 있으며, 반드시 확인해야 한다고 생각하지만 항상 핵심 프레임워크를 먼저 배우고 단일 페이지 애플리케이션을 만드는 방법을 배우는 것이 좋습니다.

슬라이드는 여기까지입니다. 채용 공고 프로젝트를 시작하기 전에 CDN을 사용하여 Vue.js를 매우 쉽고 빠르게 사용하는 방법을 보여드리고 싶습니다.

CDN으로 빠르게 시작하세요

매우 간단한 프로젝트가 아닌 경우 **create-vue** 또는 **Nuxt** 와 같은 것을 사용하는 것이 좋지만 CDN(Content Delivery Network)을 포함하면 됩니다. CDN은 전 세계에 위치한 서버 네트워크입니다. 즉, Vue.js를 사용하려면 컴퓨터에 다운로드할 필요가 없습니다. 대신 HTML 파일에 포함시켜 사용하면 됩니다.

`vue-example` 우선, 폴더를 만들고 파일을 추가한 `index.html` 다음 기본 HTML을 추가하는 것으로 시작해 보겠습니다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vue Example</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

`<head>` 이제 HTML 파일에 CDN을 추가하세요.

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

저는 VS Code와 Live Server 확장 프로그램을 사용하고 있으므로 마우스 오른쪽 버튼을 클릭하고 "Open With Live Server"를 선택하여 브라우저에서 프로젝트를 엽니다. "Hello World"라는 텍스트만 보일 것입니다.

Vue CDN을 포함했으므로 Vue.js 구문을 사용하여 새 Vue 앱을 만들 수 있습니다. 다음 코드를 `<body>`: 의 닫는 태그 바로 위에 추가합니다.

```
<script>
const app = Vue.createApp({
  data() {
    return {
      message: 'Hello Vue!',
    };
  },
});

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

이 코드를 살펴보겠습니다.

- `const app = Vue.createApp()`- 새로운 Vue 앱/인스턴스를 만들고 있습니다. `createApp` Vue 객체에서 사용할 수 있는 메서드인 메서드를 사용하고 있습니다.
- 우리는 앱의 데이터와 메서드를 담고 있는 객체를 전달하고 있습니다.
- `data()`- Vue에서 데이터 함수는 Vue 인스턴스에 대한 반응형 데이터 속성을 정의하는 데 사용됩니다. 여기서는 'Hello Vue!'로 초기화된 단일 속성 메시지가 있는 객체를 반환합니다.
- `app.mount('#app')`- `app`의 id를 가진 요소에 Vue 인스턴스를 마운트합니다.

이제 우리는 HTML에 해당 요소를 추가할 수 있습니다.

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

는 {{ message }}message 속성의 값을 표시하는 데 사용되는 특수 Vue 구문입니다. 이를 보간이라고 합니다.

이벤트와 메서드를 추가할 수도 있습니다. 걱정하지 마세요. 나중에 이 내용을 훨씬 더 자세히 살펴보겠습니다. 프로젝트에서 이것을 어떻게 사용할 수 있는지에 대한 간단한 예를 보여드리고 싶습니다.

이벤트가 있는 버튼을 만들어 보겠습니다.

```
<button @event='clickMe'>Click Me</button>
```

아제 clickMe에 메서드 옵션을 추가할 수 있습니다.

```
const app = Vue.createApp({
  data() {
    return {
      message: 'Hello Vue!',
    };
  },
  methods: {
    clickMe() {
      console.log('Button Clicked!')
    }
  }
});

app.mount('#app');
```

이제 버튼을 클릭하면 해당 메서드가 실행됩니다.

이것은 아마도 당신이 만들 수 있는 가장 간단한 Vue 프로젝트일 것입니다. 하지만 그것은 당신이 그것을 당신의 프로젝트에 쉽게 추가할 수 있는 방법에 대한 아이디어를 제공합니다.

create-vue

우리는 그 코드를 스크랩하고 도구를 사용하여 새 프로젝트를 만들 수 있습니다 vue-create. React를 사용해 본 적이 있다면, 이것은 create-react-app 도구와 비슷합니다.

터미널 창을 열고 다음을 실행합니다.

```
npm create vue@latest vue-crash-2024
```

다음 옵션을 선택하겠습니다.

- TypeScript: 아니요
- JSX: 아니요
- Vue Router: 필요할 때 직접 구현할 것이므로 여기서는 no를 선택하지 않을 것입니다.
- 피니아: 아니요
- 비테스트: 아니요
- 종단간 테스트: 없음
- ESLint: 아니요
- DevTools: 아니요

그것은 당신의 프로젝트를 발전시키는 데 도움이 될 것입니다.

VS Code나 사용 중인 편집기에서 이 폴더를 엽니다.

```
cd vue-crash-2024
code .
```

Vue VS 코드 확장

VS Code에는 몇 가지 Vue.js 확장 프로그램이 있습니다. 제가 사용하는 것은 **Vue - Official** 과 **Vue 3 Snippets** 확장 프로그램입니다. 하지만 더 마음에 드는 것을 찾았다면 괜찮습니다. Vetur와 Volar가 꽤 인기가 있습니다. .vue 파일에 대한 올바른 구문 강조 표시를 제공하는 것이 필요합니다. 이것은 Vue 구성 요소가 사용하는 파일 확장 프로그램입니다.

서버 실행

아무것도 하기 전에 종속성을 설치하고 개발 서버를 실행해 보겠습니다. 프로젝트 폴더에 있는지 확인하고 다음을 실행하세요.

```
npm install
npm run dev
```

기본적으로 이것은 포트 5173에서 실행됩니다. 저는 프런트엔드 프로젝트에 포트 3000을 사용하는 것을 선호하므로 실제로 vite.config.js 파일에 들어가서 포트를 3000으로 변경하려고 합니다.

```
server: {
  port: 3000,
},
```

<http://localhost:3000> 파일을 저장하면 서버가 자동으로 다시 로드됩니다. 이제 브라우저에서 이동하면 환영 페이지가 표시됩니다.

파일 구조

프로젝트의 파일 구조를 간단히 살펴보겠습니다.

package.json

package.json 파일은 매우 최소한입니다. 라우터를 사용하는 것에 yes라고 대답했기 때문에 vue와 vue-router만 있습니다. 또한 서버이자 터링인 Vite와 개발 종속성인 Vue 플러그인도 있습니다.

스크립트의 경우, dev 개발 서버를 실행하여 build 프로젝트를 빌드하고 preview 프로덕션 빌드를 미리 확인해야 합니다.

vite.config

구성 파일은 Vue 플러그인이 시작되는 곳입니다. 여기서 포트 번호, 프록시 등과 같은 추가 구성을 추가할 수 있습니다.

index.html

이것은 메인 HTML 파일입니다. 매우 간단합니다. Vue 앱이 마운트될 위치의 id가 있는 div가 있습니다 app. 이 프로젝트는 Webpack과 같은 것이 아니라 Vite를 사용하므로 실제 스크립트는 모두 함께 번들링하는 대신 모듈로 포함됩니다. Vite에는 ES 모듈 지원이 포함되어 있기 때문입니다.

제목을 바꾸겠습니다.

```
<title>Vue Jobs | Become a Vue Developer</title>
```

public

여기에 이미지, 글꼴 등 정적 자산을 넣을 수 있습니다.

src

여기가 모든 Vue 구성 요소와 기타 JavaScript 파일이 들어갈 곳입니다.

src/main.js

이것은 애플리케이션의 주요 진입점입니다. 여기서 Vue 앱을 생성하고 파일 #app의 div에 마운트합니다 index.html.

src/App.vue

#app 이것은 파일의 div에 로드될 주요 구성 요소입니다 index.html. 템플릿, 스크립트, 스타일을 포함하는 단일 파일 구성 요소입니다. 여기서 우리는 구인 광고 앱을 빌드할 것입니다.

HelloWorld 기본적으로 및 구성 요소를 가져옵니다 TheWelcome. 해당 구성 요소는 src/components 폴더에 있습니다.

청소하다

이 모든 것을 정리하여 깨끗한 상태로 작업해 보겠습니다. 파일의 모든 코드를 App.vue 다음으로 바꾸세요.

```
<template>
  <h1 class="text-2xl">Vue Jobs</h1>
</template>
```

Tailwind 클래스를 추가했지만 아직 Tailwind를 설정하지 않았으므로 아무런 작업도 일어나지 않습니다.

라는 폴더가 있습니다 components. 그 안에 있는 모든 것을 삭제해 보겠습니다. 우리는 우리만의 컴포넌트를 만들 것입니다.

Tailwind CSS 설정

이 프로젝트에는 Tailwind CSS를 사용할 것입니다. Tailwind에 익숙하지 않다면, Tailwind는 CSS를 작성하지 않고도 사용자 정의 디자인을 빌드할 수 있는 유ти리티 우선 CSS 프레임워크입니다. 매우 인기가 많고 저는 대부분의 프로젝트에서 Tailwind를 사용합니다.

Tailwind를 설치하려면 다음을 실행하세요.

```
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
```

이렇게 하면 Tailwind, PostCSS, Autoprefixer가 개발 종속성으로 설치됩니다. PostCSS는 JavaScript 플러그인으로 CSS를 변환하는 도구입니다. Autoprefixer는 CSS를 구문 분석하고 CSS 규칙에 공급업체 접두사를 추가하는 PostCSS 플러그인입니다.

테일윈드 구성

이제 Tailwind 구성 파일을 만들어야 합니다. 다음을 실행합니다.

```
npx tailwindcss init -p
```

이렇게 하면 프로젝트 루트에 파일이 tailwind.config.js 생성 됩니다 .postcss.config.js

다음을 tailwind.config.js 파일에 추가하세요.

```
module.exports = {
  content: ['./index.html', './src/**/*.{vue,js,ts,jsx,tsx}'],
  theme: {
    extend: {
      fontFamily: {
        sans: ['Poppins', 'sans-serif'],
      },
      gridTemplateColumns: {
        ...
```

```
'70/30': '70% 28%',  
},  
},  
},  
variants: {  
  extend: {},  
},  
plugins: [],  
};
```

이전에는 로 알려진 콘텐츠 배열은 `purge` 프로젝트에서 사용되는 클래스를 스캔할 파일을 지정합니다. 프로덕션에서 사용되는 Tailwind CSS는 최종 CSS 파일 크기를 줄이기 위해 사용되지 않는 스타일을 제거합니다.

테마 객체에 몇 가지 사용자 정의 값도 추가하고 있습니다. 글꼴 패밀리와 그리드 템플릿 열을 추가하고 있습니다. 이것은 단일 구인 목록 페이지를 위한 특수 클래스입니다.

CSS에 Tailwind 포함하기

파일을 열고 `src/assets/main.css` 다음을 추가하세요.

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

이렇게 하면 Tailwind의 CSS 파일에 기본, 구성 요소, 유틸리티 클래스가 추가됩니다.

이제 이것을 `src/main.js` 다음과 같이 파일에 포함하세요.

```
import '@/assets/main.css';
```

서버를 다시 시작하면 Tailwind 스타일이 적용되는 것을 확인할 수 있습니다.

Vue 컴포넌트 구문

구성 요소를 조립하기 전에 파일에서 `App.vue` 잠깐 작업하여 구성 요소가 어떻게 작동하는지 이해하도록 하겠습니다. 슬라이드에서 구성 요소의 구조에 대해 이미 설명했습니다. 출력인 템플릿, 스타일 및 JavaScript로 짓이 있습니다.

이러한 부분을 구분하려면 다음 태그 구문을 사용합니다.

```
<script>  
// Logic goes here  
</script>  
  
<template>  
  <!-- HTML goes here -->  
  <div>  
    <h1 class="text-2xl">Vue Jobs</h1>  
  </div>  
</template>  
  
<style>  
/* Styles go here */  
</style>
```

스타일링을 위해 스타일 태그에 속성을 추가합니다 `scoped`. 이렇게 하면 스타일이 구성 요소 자체에만 적용됩니다.

```
<style scoped>  
h1 {  
  color: red;
```

```
    }
  </style>
```

이제 `h1`이 구성 요소의 태그만 빨간색이 됩니다.

`lang` 스타일 태그에 속성을 추가하여 Sass를 사용할 수도 있습니다.

```
<style lang="scss">
h1 {
  color: red;
}
</style>
```

하지만 이것이 작동하려면 sass 패키지를 설치해야 합니다. 저는 그렇게 하지 않을 것이지만, 원하신다면 다음 명령을 실행하세요:

```
npm install -D sass
```

Tailwind를 사용하고 있으므로 이 앱에 사용자 정의 CSS를 많이 추가할 필요가 없으므로 스타일 태그를 완전히 제거할 수 있습니다.

구성 요소에 데이터 추가

많은 구성 요소에는 상태 또는 데이터가 연관되어 있습니다. 예를 들어, 작업 목록을 표시하는 구성 요소에는 데이터에 작업 목록이 있습니다. 앞서 언급했듯이 옵션 API 또는 구성 API를 사용하여 이를 처리할 수 있습니다. 구성 API는 Vue에서 데이터를 처리하는 데 권장되는 방법입니다. 더 유연하며 궁극적으로 이 프로젝트에서 사용할 것이라면, Vue 개발자로서 옵션 API의 기본 사항을 알고 싶어합니다. 적어도 가까운 미래에는 그럴 것입니다. 옵션 API부터 시작해 보겠습니다.

이름 같은 간단한 것으로 시작해 보겠습니다. `App.vue` 파일의 스크립트 태그에 다음을 추가합니다.

```
<script>
export default {
  data() {
    return {
      name: 'John Doe',
    };
  },
};
</script>
```

보간

`data` 옵션 API는 컴포넌트에 대한 데이터를 반환하기 위해 명명된 함수를 사용합니다. 이 경우, 단일 속성이 'John Doe'로 설정된 객체를 반환합니다. 이제 템플릿에서 이 데이터를 사용할 수 있습니다. 템플릿의 태그를 다음으로 `name` 바꾸세요. `h1`

```
<template>
<div class="flex flex-col items-center mt-10">
  <h1 class="text-2xl">{{ name }}</h1>
</div>
</template>
```

이제 제목에 이름이 표시됩니다. 이를 보간이라고 하며 구성 요소의 데이터를 사용하는 방법입니다. 변수를 이중 중괄호로 묶기만 하면 됩니다.

지시사항

Vue에는 지시어라는 것이 있습니다. 이것은 HTML 요소에 추가하여 구성 요소가 렌더링되는 방식을 변경할 수 있는 특수 속성입니다. 예를 들어 지시어를 사용하여 태그 `v-text` 내부의 텍스트를 변경할 수 있습니다 `h1`.

`h1` 태그를 다음으로 바꾸세요.

```
<h1 class="text-2xl" v-text="name"></h1>
```

이 작업은 H1 태그의 `textContent` 속성을 속성 값으로 업데이트하는 것입니다 `name`. 태그 콘텐츠 내의 모든 것을 덮어씁니다. 값만 표시하려는 이 특정 사례에서는 `v-text`지시문 대신 보간을 사용해야 합니다. 그러니 다시 변경해 보겠습니다.

```
<h1 class="text-2xl">{{ name }}</h1>
```

다음은 몇 가지 일반적인 지침입니다.

- `v-if`- 표현식이 참이면 요소를 렌더링합니다. 또한 및 도 있습니다 `v-else-if`.
- `v-for`- 항목 배열을 반복하고 렌더링합니다.
- `v-bind`- 구성 요소의 속성을 바인딩합니다.
- `v-on`- 이벤트를 핸들에 연결합니다.
- `v-model`- 구성 요소의 속성을 입력을 바인딩합니다.
- `v-show`- 표현식에 따라 요소를 표시하거나 숨깁니다.

이러한 것들과 몇 가지 다른 것들도 살펴보겠습니다.

v-if

지시어 사용의 예를 살펴보겠습니다 `v-if`. 먼저 구성 요소에 새 데이터를 추가합니다. 스크립트 태그에 다음을 추가합니다.

```
<script>
export default {
  data() {
    return {
      name: 'John Doe',
      status: 'active',
    };
  },
}
</script>
```

이제 우리는 `v-if`지시문을 사용하여 상태를 조건부로 렌더링할 수 있습니다. 이를 아래에 다음을 추가합니다.

```
<h1 class="text-2xl">{{ name }}</h1>
<p v-if="status === 'active'" class="text-green-700">User is active</p>
```

이름은 표시되지만 `status`속성을 다른 것으로 변경하면 `p`태그는 표시되지 않습니다.

v-else

조건이 거짓이면 다른 것을 표시하는 데 사용할 수도 있습니다 `v-else`. 템플릿에 다음을 추가합니다.

```
<h1 class="text-2xl">{{ name }}</h1>
<p v-if="status === 'active'" class="text-green-700">User is active</p>
<p v-else class="text-red-700">User is Inactive</p>
```

`status`이 다른 것으로 설정된 경우 사용자가 비활성 상태임을 표시합니다.

v-else-if

`v-else-if` 다른 조건을 추가하는 데 사용할 수도 있습니다.

```
<h1 class="text-2xl">{{ name }}</h1>
<p v-if="status === 'active'" class="text-green-600">User is Active</p>
<p v-else-if="status === 'pending'" class="text-yellow-600">
    User is Pending
</p>
<p v-else class="text-red-700">User is Inactive</p>
```

이제 활성 상태이면 활성으로 표시되고, 보류 중이면 보류 중으로 표시되고, 그 외의 상태이면 비활성으로 표시됩니다.

v-for

또한 항목 배열을 반복하는데 사용할 수 있습니다 v-for. 데이터에 작업 배열을 추가해 보겠습니다.

```
<script>
export default {
  data() {
    return {
      name: 'John Doe',
      status: 'active',
      tasks: ['Task One', 'Task Two', 'Task Three', 'Task Four'],
    };
  },
};
</script>
```

이제 다음을 사용하여 v-for 작업을 반복할 수 있습니다.

```
<h3 class="text-xl my-3">Tasks:</h3>
<ul>
  <li v-for="task in tasks" :key="task" class="flex items-center my-3">{{ task }}</li>
</ul>
```

지시문은 . v-for 형식을 취합니다 v-for="item in items". :key 속성은 Vue가 요소와 해당 상태를 추적하는 데 도움이 되도록 사용할 때 필요합니다 v-for. 이제 페이지에서 작업을 볼 수 있습니다.

v-bind

지시문 v-bind는 속성을 컴포넌트의 속성에 바인딩하는 데 사용됩니다. 예를 들어 href 앵커 태그의 속성을 컴포넌트의 속성에 바인딩할 수 있습니다. 데이터에 링크를 추가해 보겠습니다.

```
<script>
export default {
  data() {
    return {
      name: 'John Doe',
      status: 'active',
      tasks: ['Task One', 'Task Two', 'Task Three', 'Task Four'],
      link: 'https://google.com',
    };
  },
};
</script>
```

href 이제 앵커 태그의 속성을 속성에 바인딩할 수 있습니다 link.

```
<a v-bind:href="link" class="text-blue-600">Link to Google</a>
```

이것을 다음과 같이 줄일 수도 있습니다 :href.

```
<a :href="link" class="text-blue-600">Link to Google</a>
```

v-on& 방법

지시문 v-on은 이벤트를 함수/메서드에 바인딩하는 데 사용됩니다. 예를 들어 클릭 이벤트를 상태를 변경하는 함수에 바인딩할 수 있습니다. 그렇게 해봅시다.

먼저 다음 버튼을 추가합니다.

```
<button
  v-on:click='toggleStatus'
  class='mt-3 px-4 py-2 bg-blue-500 text-white rounded-md'
>
  Change Status
</button>
```

버튼을 클릭하면 클릭 이벤트가 발생합니다. 이제 데이터에 메서드를 추가해 보겠습니다. 여전히 Options API를 사용하고 있다는 점을 기억하세요. 곧 Composition API로 변환하겠습니다.

```
<script>
export default {
  data() {
    return {
      name: 'John Doe',
      status: 'active',
      tasks: ['Task One', 'Task Two', 'Task Three', 'Task Four'],
      link: 'https://google.com',
    };
  },
  methods: {
    toggleStatus() {
      if (this.status === 'active') {
        this.status = 'pending';
      } else if (this.status === 'pending') {
        this.status = 'inactive';
      } else {
        this.status = 'active';
      }
    },
  },
};
</script>
```

보시다시피, 우리는 단지 라는 객체를 가지고 있습니다 methods. 그 객체 안에 우리는 라는 메소드를 가지고 있습니다 toggleStatus. 이 메소드는 사용자의 상태를 바꿀 것입니다. 이제 버튼을 클릭하면 상태가 바뀔 것입니다. 우리는 this키워드를 사용하여 컴포넌트의 데이터에 접근합니다.

v-on속기

이벤트를 처리하는 더 짧은 방법이 있습니다. 심볼을 사용하여 @이벤트를 메서드에 바인딩할 수 있습니다. 예를 들어, click이벤트를 메서드에 바인딩할 수 있습니다 toggleStatus.

```
<button
  @click="toggleStatus"
  class='mt-3 px-4 py-2 bg-blue-500 text-white rounded-md'
>Change Status</button>
```

나중에 다른 지침도 살펴보겠지만, 다음은 가장 일반적인 지침입니다.

지금까지의 전체 코드는 다음과 같습니다.

```

<script>
export default {
  data() {
    return {
      name: 'John Doe',
      status: 'active',
      tasks: ['Task One', 'Task Two', 'Task Three', 'Task Four'],
      link: 'https://google.com',
    };
  },
  methods: {
    toggleStatus() {
      if (this.status === 'active') {
        this.status = 'pending';
      } else if (this.status === 'pending') {
        this.status = 'inactive';
      } else {
        this.status = 'active';
      }
    },
  },
};
</script>

<template>
<div class="flex flex-col items-center mt-10">
  <h1 class="text-2xl">{{ name }}</h1>
  <p v-if="status === 'active'" class="text-green-600">User is Active</p>
  <p v-else-if="status === 'pending'" class="text-yellow-600">
    User is Pending
  </p>
  <p v-else class="text-red-700">User is Inactive</p>
  <button
    @click="toggleStatus"
    class="mt-3 px-4 py-2 bg-blue-500 text-white rounded-md">
    Change Status
  </button>
  <h3 class="text-xl my-3">Tasks:</h3>
  <ul>
    <li v-for="task in tasks" :key="task" class="flex items-center my-3">
      {{ task }}
    </li>
  </ul>
  <a v-bind:href="link" class="text-blue-600">Link to Google</a>
</div>
</template>

```

구성 API

Composition API는 Vue 컴포넌트에서 로직을 처리하는 새로운 방법입니다. 더 유연하고 더 복잡한 컴포넌트를 만들 수 있습니다. 또한 React의 hooks와 더 비슷합니다. 컴포넌트를 App.vueComposition API를 사용하도록 변환해 보겠습니다.

```

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

export default {
  setup() {
    const name = ref('John Doe');
    const status = ref('active');
    const tasks = ref(['Task One', 'Task Two', 'Task Three', 'Task Four']);
    const link = ref('https://google.com');

    const toggleStatus = () => {
      if (status.value === 'active') {
        status.value = 'pending';
      } else if (status.value === 'pending') {
        status.value = 'inactive';
      } else {
        status.value = 'active';
      }
    };
  }
};
</script>

```

```

    }
};

return {
  name,
  status,
  tasks,
  link,
  toggleStatus,
};
},
};

</script>

```

Composition API를 사용하면 `setup`함수를 사용하여 반응형 속성과 메서드를 정의합니다. 함수를 사용하여 `ref`반응형 참조를 만듭니다. 반응형이라는 말은 속성 값을 변경하면 UI가 자동으로 업데이트된다는 뜻입니다. `ref`내부 값을 가져와서 반응형이고 변경 가능한 참조 객체를 반환합니다. 이 객체에는 내부 값을 가리키는 `.value`라는 단일 속성이 있습니다. 속성을 사용하여 참조 값을 직접 변경할 수 있습니다 `.value`. 따라서 React 개발자라면 `useState`자동 방식과 비슷합니다. 참조 값을 수정하면 Vue는 해당 참조가 사용되는 템플릿에서 반응형 업데이트를 자동으로 트리거합니다. 예, 참조 없이 변수를 만들 수 있으며 처음에는 표시되지만 값이 변경되어도 업데이트되지 않습니다. 참조를 사용하여 반응형으로 만들어야 합니다.

언뜻 보기에도 우리가 방금 가진 것보다 훨씬 더 어려워 보인다는 걸 알아요. 저도 같은 생각을 했고 싶어했어요. 하지만 몇 가지 프로젝트에서 사용해 본 후, 데이터와 메서드가 있는 딱딱한 객체보다 얼마나 더 유연한지 알게 되었고 장점이 있다는 걸 알게 되었어요.

제가 방금 보여드린 것도 긴 형식입니다. 우리는 이것을 정말 간소화할 수 있습니다.

setup()함수 속기

첫째, 명시적인 `setup` 함수를 사용할 필요가 없습니다. 함수에 모든 것을 실제로 래핑하는 대신 `setup()`, 다음과 같이 스크립트에 `setup`을 추가할 수 있습니다.

```

<script setup>
// ...
</script>

```

그러면 수출과 반품을 제거하면 다음과 같은 결과가 나옵니다.

```

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

const name = ref('John Doe');
const status = ref('active');
const tasks = ref(['Task One', 'Task Two', 'Task Three', 'Task Four']);
const link = ref('https://google.com');

const toggleStatus = () => {
  if (status.value === 'active') {
    status.value = 'pending';
  } else if (status.value === 'pending') {
    status.value = 'inactive';
  } else {
    status.value = 'active';
  }
};
</script>

```

Google 링크 바인딩과 링크 속성을 제거할 수 있습니다.

작업 추가

제가 하고 싶은 또 다른 일은 새 작업을 추가하는 양식을 추가하는 것입니다. 템플릿에 양식을 추가하면 됩니다.

```

<form @submit.prevent="addTask">
  <label for="newTask" class="block text-xl my-3">Add New Task:</label>
  <input

```

```

        type="text"
        id="newTask"
        v-model="newTask"
        class="border border-gray-300 rounded-md px-3 py-2 w-full"
      />
      <button
        type="submit"
        class="mt-3 px-4 py-2 bg-green-500 text-white rounded-md"
      >
        Add Task
      </button>
    </form>
  
```

v-model

지시문 v-model은 입력을 구성 요소의 속성에 바인딩하는 데 사용됩니다. 이 경우 입력을 속성에 바인딩합니다 newTask. 이렇게 하면 입력을 입력하고 값을 속성에 저장할 수 있습니다 newTask.

스크립트에 속성을 추가해 보겠습니다.

```
const newTask = ref('');
```

이제 addTask스크립트에 메서드를 추가할 수 있습니다.

```

const addTask = () => {
  if (newTask.value.trim() !== '') {
    tasks.value.push(newTask.value);
    newTask.value = ''; // Clear input after adding task
  }
};
  
```

이제 작업이 추가되고, 작업이 반응형이기 때문에 목록이 업데이트됩니다.

작업 삭제

이제 작업을 삭제하는 버튼을 추가해 보겠습니다. 작업과 함께 목록 항목에 버튼을 추가합니다.

```

<li
  v-for="(task, index) in tasks"
  :key="index"
  class="flex items-center my-3"
>
  <span>{{ task }}</span>
  <button
    @click="deleteTask(index)"
    class="ml-2 px-3 py-1 bg-red-500 text-white rounded-md text-xs"
  >
    X
  </button>
</li>
  
```

deleteTask 그런 다음 다음 스크립트에 메서드를 추가합니다 .

```

const deleteTask = (index) => {
  tasks.value.splice(index, 1);
};
  
```

Vue.js 브라우저 Devtools

브라우저에 Vue devtools를 설치하는 것을 제안합니다. Chrome과 Firefox에서 사용할 수 있습니다. 10년 이상 다른 브라우저를 사용하지 않았기 때문에 다른 브라우저는 잘 모르겠습니다. 이렇게 하면 Vue 구성 요소를 검사하고 구성 요소의 데이터, 속성 및 상태를 볼 수 있습니다. 매우 유용한 도구입니다.

다음 URL에서 다운로드하고 설치할 수 있습니다.

- [크롬](#)
- [파이어폭스](#)

설치가 완료되면 DevTools에 "Vue" 옵션이 생깁니다. 거기서 컴포넌트를 클릭하면 props, state 등을 볼 수 있습니다.

라이프사이클 후크

Vue에는 라이프사이클 후크라는 것이 있습니다. 이는 구성 요소의 라이프사이클에서 여러 지점에서 호출되는 함수입니다. 예를 들어, 구성 요소가 생성, 마운트, 업데이트 및 파괴될 때입니다. 이러한 후크를 사용하여 이러한 여러 지점에서 작업을 수행할 수 있습니다. 실제로 Vue 3에서 setup() 함수로 대체되어 제거된 후크가 몇 가지 있지만 주요 후크는 다음과 같습니다.

- onBeforeMount – 마운트가 시작되기 전에 호출됨
- onMounted – 컴포넌트가 마운트될 때 호출됨
- onBeforeUpdate – 반응형 데이터가 변경되고 다시 렌더링되기 전에 호출됩니다.
- onUpdated – 다시 렌더링한 후 호출됨
- onBeforeUnmount – Vue 인스턴스가 파괴되기 전에 호출됨
- onUnmounted – 인스턴스가 파괴된 후 호출됨
- onActivated – 활성 상태인 구성 요소가 활성화될 때 호출됨
- onDeactivated – 살아있는 구성 요소가 비활성화될 때 호출됩니다.
- onErrorCaptured – 자식 구성 요소에서 오류가 캡처될 때 호출됩니다.

사용하려면 가져와야 합니다. 이는 모든 것을 가볍게 유지하기 위한 것입니다.

앱/구성 요소가 로드될 때 데이터를 가져오는 것과 같은 작업에 이를 사용할 수 있습니다. onMounted후크를 사용하여 JSONplaceholder API에서 일부 작업을 가져온 다음 이를 작업 속성에 추가해 보겠습니다.

스크립트 하단에 다음을 추가합니다.

```
onMounted(async () => {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/todos');
    const data = await response.json();
    tasks.value = data.map((task) => task.title);
  } catch (error) {
    console.error('Error fetching tasks:', error);
  }
});
```

배열을 평가한 다음 제목 배열로 변환한 다음 작업 속성에 추가합니다. 이제 API에서 작업을 볼 수 있습니다.

일자리 프로젝트

이제 구성 요소의 구조와 보간, 지시문, 이벤트 등에 대해 알았으니, 메인 프로젝트를 시작해 봅시다. 이 모든 코드를 파일에서 지울 수 있습니다. 실제로는 필요할 경우를 대비해 repo에 보관하기 위해 just App.vue라는 파일에 넣을 것입니다 .App2.vue

테마 파일

이 프로젝트가 보기 좋게 보이길 원했기 때문에 메인 리포에서 .이라는 폴더를 볼 수 있을 겁니다 _theme_files. 여기에 모든 테마 파일을 넣었습니다. 이것들은 Tailwind 클래스가 있는 HTML 파일일 뿐입니다. 쉽게 접근할 수 있도록 다른 텍스트 편집기 창에서 해당 폴더를 여는 것이 좋습니다.

먼저, 파일의 모든 내용을 파일 태그 index.html에 복사해 보겠습니다 .App.vue<template>

로고 이미지에 대한 오류가 발생할 가능성이 높습니다. logo.png테마 파일에서 파일을 가져와서 파일 src/assets/img에 다음을 추가합니다 App.vue.

```
import logo from '@/assets/img/logo.png';
```

우리가 이런 일을 할 수 있다고 생각할 수도 있을 겁니다.

```
<img src={logo} alt='Logo' />
```

React에서는 이런 식으로 하지만 Vue에서는 로고를 이미지 태그에 다음과 같이 바인딩합니다.

```
<img class='h-10 w-auto' v-bind:src='logo' alt='Vue Jobs' />
```

이렇게 하면 더 짧게 만들 수 있습니다.

```

```

이제 로고가 보일 것입니다.

첫 번째 UI 구성 요소

이제 이것을 구성 요소로 나눌 시간입니다. 위에서부터 탐색 모음을 위한 구성 요소를 만들어 보겠습니다. 에서 파일을 만들고 파일에서 와 그 안의 모든 것을 src/components/Navbar.vue 잘라내어 태그로 파일에 붙여 넣습니다 .<nav>App.vue<template>

```
<template>
  <nav class="bg-green-700 border-b border-green-500">
    <div class="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">
      <div class="flex h-20 items-center justify-between">
        <div
          class="flex flex-1 items-center justify-center md:items-stretch md:justify-start"
        >
          <!-- Logo -->
          <a class="flex flex-shrink-0 items-center mr-4" href="index.html">
            
            <span class="hidden md:block text-white text-2xl font-bold ml-2" style="font-family: inherit;">
              Vue Jobs
            
          </a>
          <div class="md:ml-auto">
            <div class="flex space-x-2">
              <a
                href="index.html"
                class="text-white bg-green-900 hover:bg-gray-900 hover:text-white rounded-md px-3 py-2"
              >Home</a>
              >
              <a
                href="jobs.html"
                class="text-white hover:bg-green-900 hover:text-white rounded-md px-3 py-2"
              >Jobs</a>
              >
              <a
                href="add-job.html"
                class="text-white hover:bg-green-900 hover:text-white rounded-md px-3 py-2"
              >Add Job</a>
              >
            </div>
          </div>
        </div>
      </div>
    </div>
  </nav>
</template>
```

이제 로고를 . 대신 이 구성 요소로 가져와야 합니다 App.vue 맨 위에 다음을 추가합니다.

```
<script setup>
import logo from '@/assets/img/logo.png';
</script>
```

Navbar.vue 이제 구성 요소를 구성 요소로 가져와 보겠습니다 App.vue.

```
<script setup>
import Navbar from '@/components/Navbar.vue';
</script>
```

이제 Navbar 구성 요소를 포함합니다.

```
<template>
  <Navbar />
</template>
```

영웅 구성요소

다음으로 Hero 구성 요소를 만들어 보겠습니다. 파일을 만들고 파일에서 src/components/Hero.vue Hero와 그 안의 모든 내용이 있는 섹션을 잘라내어 태그에 파일 App.vue에 붙여넣습니다 .Hero.vue<template>

```
<template>
<section class="bg-green-700 py-20 mb-4">
  <div
    class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 flex flex-col items-center">
    <div class="text-center">
      <h1 class="text-4xl font-extrabold text-white sm:text-5xl md:text-6xl">
        Become a Vue Dev
      </h1>
      <p class="my-4 text-xl text-white">
        Find the Vue job that fits your skills and needs
      </p>
    </div>
  </div>
</section>
</template>
```

그런 다음 이를 기본 구성 요소로 가져와서 포함합니다.

```
<script setup>
import Navbar from '@/components/Navbar.vue';
import Hero from '@/components/Hero.vue';
</script>

<template>
  <Navbar />
  <Hero />
  <!-- //... -->
</template>
```

소품

이제 props에 대해 이야기해 봅시다. React나 Angular를 사용해 본 적이 있다면 props가 무엇인지 알고 있을 것입니다. 사용해 본 적이 없다면 props는 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달하는 방법입니다. 이렇게 하면 컴포넌트를 재사용 가능하게 만들 수 있습니다. props로 데이터, 함수 또는 다른 컴포넌트를 전달 할 수 있습니다.

Hero 컴포넌트에 제목과 부제목 속성을 전달하여 출력에 표시할 수 있게 하고 싶습니다.

파일에서 `App.vue` 두 개의 `props`를 전달해 보겠습니다.

```
<template>
  <Hero title="Test Title" />
</template>
```

우리는 문자열에 불과한 `prop`을 전달하고 있습니다. 이제 파일에 `prop`을 설정해야 합니다 `Hero.vue`. 다음을 스크립트 태그에 추가합니다.

```
<script setup>
import { defineProps } from 'vue';

defineProps({
  title: {
    type: String,
    default: 'Become a Vue dev',
  },
});
</script>
```

`defineProps` 라이브러리에서 함수를 가져오고 있습니다. 이 함수는 `props`를 정의하는 데 사용됩니다. 우리는 의 유형과 의 기본값으로 `vue`로 출되는 `props`를 정의하고 있습니다. 이제 템플릿에서 이 `props`를 사용할 수 있습니다.

```
<h1 class='text-4xl font-extrabold text-white sm:text-5xl md:text-6xl'>
  {{ title }}
</h1>
```

이제 제목이 "테스트 제목"으로 표시되어야 합니다.

자막을 넣어보겠습니다.

```
<template>
  <Hero title="Test Title" subtitle="Test Subtitle" />
</template>
```

다음을 `Hero.vue` 파일에 추가하세요.

```
<script setup>
import { defineProps } from 'vue';

defineProps({
  title: {
    type: String,
    default: 'Become a Vue dev',
  },
  subtitle: {
    type: String,
    default: 'Find the Vue job that fits your skills and needs',
  },
});
</script>
```

출력에서 자막을 볼 수 있어야 합니다.

이렇게 해서 다른 구성 요소로 데이터를 전달할 수 있습니다. 이제 임베드에서 제목과 부제를 제거할 수 있습니다.

```
<Hero />
```

기본값을 확인할 수 있습니다.

홈카드 컴포넌트

구성 요소를 만들고 src/components/HomeCards.vue 다음 섹션을 잘라내어 해당 파일의 템플릿에 붙여넣어 보겠습니다.

```
<template>
<section class="py-4">
  <div class="container-xl lg:container m-auto">
    <div class="grid grid-cols-1 md:grid-cols-2 gap-4 p-4 rounded-lg">
      <div class="bg-gray-100 p-6 rounded-lg shadow-md">
        <h2 class="text-2xl font-bold">For Developers</h2>
        <p class="mt-2 mb-4">
          Browse our Vue jobs and start your career today
        </p>
        <a
          href="jobs.html"
          class="inline-block bg-black text-white rounded-lg px-4 py-2 hover:bg-gray-700">
          Browse Jobs
        </a>
      </div>
      <div class="bg-green-100 p-6 rounded-lg shadow-md">
        <h2 class="text-2xl font-bold">For Employers</h2>
        <p class="mt-2 mb-4">
          List your job to find the perfect developer for the role
        </p>
        <a
          href="add-job.html"
          class="inline-block bg-green-500 text-white rounded-lg px-4 py-2 hover:bg-green-600">
          Add Job
        </a>
      </div>
    </div>
  </div>
</section>
</template>
```

가져오기 및 내장 App.vue:

```
<script setup>
import Navbar from '@/components/Navbar.vue';
import Hero from '@/components/Hero.vue';
import HomeCards from '@/components/HomeCards.vue';
</script>

<template>
<Navbar />
<Hero />
<HomeCards />
<!-- //... -->
</template>
```

래퍼 구성 요소 및 <slot>

Card 콘텐츠를 래핑하는 컴포넌트도 있을 수 있습니다. 텍스트를 래핑하고 스타일을 추가하여 카드처럼 보이게 하는 컴포넌트를 만들어 보겠습니다.

파일을 만들고 src/components/Card.vue 다음을 추가합니다.

```
<template>
<div class="bg-gray-100 p-6 rounded-lg shadow-md">
  <slot></slot>
</div>
</template>
```

는 <slot>컴포넌트에 삽입될 컨텐츠의 플레이스홀더입니다. React에 익숙하다면 prop과 비슷합니다 {children}. 이제 이 컴포넌트를 사용하여 컴포넌트의 텍스트를 래핑할 수 있습니다 HomeCards.vue.

```
<script setup>
import Card from '@/components/Card.vue';
</script>
```

이제 컨텐츠를 감싸는 2개의 div를 교체합니다. 이들은 bg-gray-100 및 클래스를 갖습니다 bg-green-100. 이제 Card.vue 구성 요소를 사용하여 텍스트를 감싸는 것이 가능합니다.

```
<template>
<section class="py-4">
<div class="container-xl lg:container m-auto">
<div class="grid grid-cols-1 md:grid-cols-2 gap-4 p-4 rounded-lg">
<Card>
<h2 class="text-2xl font-bold">For Developers</h2>
<p class="mt-2 mb-4">
    Browse our Vue jobs and start your career today
</p>
<a href="jobs.html" class="inline-block bg-black text-white rounded-lg px-4 py-2 hover:bg-gray-700">
    Browse Jobs
</a>
</Card>
<Card>
<h2 class="text-2xl font-bold">For Employers</h2>
<p class="mt-2 mb-4">
    List your job to find the perfect developer for the role
</p>
<a href="add-job.html" class="inline-block bg-green-500 text-white rounded-lg px-4 py-2 hover:bg-green-600">
    Add Job
</a>
</Card>
</div>
</div>
</section>
</template>
```

이제 텍스트가 카드로 래핑되었습니다. 그러나 두 번째 카드는 녹색 배경을 원합니다. 따라서 카드 구성 요소가 background라는 prop을 가져오도록 합시다 bg. 다음을 Card.vue 파일에 추가합니다.

```
<script setup>
import { defineProps } from 'vue';

defineProps({
  bg: {
    type: String,
    default: 'bg-gray-100',
  },
});
</script>
```

이제 템플릿에서 해당 prop을 사용하세요:

```
<template>
<div :class="`bg p-6 rounded-lg shadow-md`">
<slot></slot>
</div>
</template>
```

클래스에서 동적 값을 사용하므로 `.:class`의 줄임말인 `v-bind:class`를 사용합니다.

이제 파일에서 `prop` `HomeCards.vue`을 사용하여 두 번째 카드의 배경색을 변경할 수 있습니다.

```
<Card bg='bg-green-100'>
  <h2 class='text-2xl font-bold'>For Employers</h2>
  <p class='mt-2 mb-4'>
    List your job to find the perfect developer for the role
  </p>
  <a href='add-job.html' class='inline-block bg-green-500 text-white rounded-lg px-4 py-2 hover:bg-green-600'>
    Add Job
  </a>
</Card>
```

데이터 작업

이제 '직무 탐색' 섹션으로 넘어갑니다. 실제 애플리케이션에서 이러한 직무는 어떤 종류의 데이터베이스나 API에서 나옵니다. 서버에 요청을 하고 데이터를 다시 받습니다. 지금은 JSON 파일을 사용합니다. 나중에 JSON Server라는 것을 사용하여 가짜 REST API를 만듭니다. 지금은 폴더 `jobs.json`에 이름이 지정된 파일을 만들고 `src` 다음 코드를 추가합니다.

```
[{"id": 1, "title": "Senior Vue Developer", "type": "Full-Time", "description": "We are seeking a talented Front-End Developer to join our team in Boston, MA. The ideal candidate will have strong skills in HTML, CSS, and JavaScript.", "location": "Boston, MA", "salary": "$70K - $80K", "company": {"name": "NewTek Solutions", "description": "NewTek Solutions is a leading technology company specializing in web development and digital solutions. We pride ourselves on delivering high-quality products and exceptional customer service.", "contactEmail": "contact@teksolutions.com", "contactPhone": "555-555-5555"}}, {"id": 2, "title": "Front-End Engineer (Vue)", "type": "Full-Time", "description": "Join our team as a Front-End Developer in sunny Miami, FL. We are looking for a motivated individual with a passion for crafting beautiful and functional user interfaces.", "location": "Miami, FL", "salary": "$70K - $80K", "company": {"name": "Veneer Solutions", "description": "Veneer Solutions is a creative agency specializing in digital design and development. Our team is dedicated to pushing the boundaries of creativity and innovation.", "contactEmail": "contact@lorem ipsum.com", "contactPhone": "555-555-5555"}}, {"id": 3, "title": "Vue.js Developer", "type": "Full-Time", "description": "Are you passionate about front-end development? Join our team in vibrant Brooklyn, NY, and work on exciting projects that make a difference.", "location": "Brooklyn, NY", "salary": "$70K - $80K", "company": {"name": "Dolor Cloud", "description": "Dolor Cloud is a leading technology company specializing in digital solutions for businesses of all sizes. With a focus on innovation and customer satisfaction, we are always looking for talented individuals to join our team.", "contactEmail": "contact@dolorsitamet.com", "contactPhone": "555-555-5555"}}, {"id": 4, "title": "Vue Front-End Developer", "type": "Full-Time", "description": "We are looking for a highly skilled Vue.js developer to join our growing team in San Francisco, CA. You will be responsible for building robust and efficient front-end applications using modern web technologies."}]
```

```

    "type": "Part-Time",
    "description": "Join our team as a Part-Time Front-End Developer in beautiful Phoenix, AZ. We are looking for a self-motivated individual with a passion for",
    "location": "Phoenix, AZ",
    "salary": "$60K - $70K",
    "company": {
      "name": "Alpha Elite",
      "description": "Alpha Elite is a dynamic startup specializing in digital marketing and web development. We are committed to fostering a diverse and inclusi",
      "contactEmail": "contact@adipiscingelite.com",
      "contactPhone": "555-555-5555"
    }
  },
  {
    "id": 5,
    "title": "Full Stack Vue Developer",
    "type": "Full-Time",
    "description": "Exciting opportunity for a Full-Time Front-End Developer in bustling Atlanta, GA. We are seeking a talented individual with a passion for bui",
    "location": "Atlanta, GA",
    "salary": "$90K - $100K",
    "company": {
      "name": "Browning Technologies",
      "description": "Browning Technologies is a rapidly growing technology company specializing in e-commerce solutions. We offer a dynamic and collaborative wo",
      "contactEmail": "contact@consecteturadipiscing.com",
      "contactPhone": "555-555-5555"
    }
  },
  {
    "id": 6,
    "title": "Vue Native Developer",
    "type": "Full-Time",
    "description": "Join our team as a Front-End Developer in beautiful Portland, OR. We are looking for a skilled and enthusiastic individual to help us create",
    "location": "Portland, OR",
    "salary": "$100K - $110K",
    "company": {
      "name": "Port Solutions INC",
      "description": "Port Solutions is a leading technology company specializing in software development and digital marketing. We are committed to providing ou",
      "contactEmail": "contact@ipsumlorem.com",
      "contactPhone": "555-555-5555"
    }
  }
]

```

이제 데이터를 App 구성 요소로 가져와서 구인 공고를 렌더링하는 데 사용할 수 있습니다.

```
import jobData from './jobs.json';
```

데이터를 반응형으로 만들기

이제 데이터가 있으므로 데이터가 변경될 때 구성 요소가 업데이트되도록 반응형으로 만들어야 합니다. `ref`함수를 사용하여 데이터를 반응형으로 만들 수 있습니다. 스크립트에 다음을 추가합니다.

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

이제 스크립트에 다음을 추가하세요.

```
const jobs = ref(jobData);
```

이제 데이터는 반응형이 됩니다.

작업을 반복하고 템플릿에 데이터를 출력해야 합니다. 궁극적으로 각 작업 항목에 대한 구성 요소가 있지만 지금은 HTML을 여기에 추가하기만 합니다. 지시문을 사용하여 작업을 반복하고 데이터를 출력해야 합니다. 이는 React/JavaScript에서 `v-for` 사용하는 것과 비슷합니다. `.jobs.map()`

현재 섹션을 다음으로 바꾸세요:

```
<section class="bg-blue-50 px-4 py-10">
  <div class="container-xl lg:container m-auto">
    <h2 class="text-3xl font-bold text-green-500 mb-6 text-center">
      Browse Jobs
    </h2>
    <div class="grid grid-cols-1 md:grid-cols-3 gap-6">
      <!-- Iterate over jobs array -->
      <div v-for="job in jobs"
           :key="job.id"
           class="bg-white rounded-xl shadow-md relative">
        <div class="p-4">
          <div class="mb-6">
            <div class="text-gray-600 my-2">Full-Time</div>
            <h3 class="text-xl font-bold">Senior Front-End Developer</h3>
          </div>

          <div class="mb-5">
            We are seeking a talented Front-End Developer to join our team in
            Boston, MA. The ideal candidate will have strong skills in HTML,
            CSS, and JavaScript...
          </div>

          <h3 class="text-green-500 mb-2">$70K - $80K / Year</h3>
        </div>

        <div class="border border-gray-100 mb-5"></div>

        <div class="flex flex-col lg:flex-row justify-between mb-4">
          <div class="text-orange-700 mb-3">
            <i class="fa-solid fa-location-dot text-lg"></i>
            Boston, MA
          </div>
          <a href="job.html"
              class="h-[36px] bg-green-500 hover:bg-green-600 text-white px-4 py-2 rounded-lg text-center text-sm">
            Read More
          </a>
        </div>
      </div>
    </div>
  </div>
</section>
```

JSON 파일에 작업 수가 6개이므로 6개의 작업이 표시되어야 합니다.

JobListing 구성 요소

이제 새 파일을 만들고 `src/components/JobListing.vue` 다음을 추가해 보겠습니다.

```
<script setup>
import { defineProps } from 'vue';

const props = defineProps({
  job: Object,
});
</script>

<template>
  <div class="bg-white rounded-xl shadow-md relative">
    <div class="p-4">
      <div class="mb-6">
        <div class="text-gray-600 my-2">{{ job.type }}</div>
        <h3 class="text-xl font-bold">{{ job.title }}</h3>
      </div>
      <div class="mb-5">{{ job.description }}</div>
    </div>
  </div>
</template>
```

```
<h3 class="text-green-500 mb-2">{{ job.salary }} / Year</h3>

<div class="border border-gray-100 mb-5"></div>

<div class="flex flex-col lg:flex-row justify-between mb-4">
  <div class="text-orange-700 mb-3">
    <i class="fa-solid fa-location-dot text-lg"></i>
    {{ job.location }}
  </div>
  <a
    :href="'/job/' + job.id"
    class="h-[36px] bg-green-500 hover:bg-green-600 text-white px-4 py-2 rounded-lg text-center text-sm"
  >
    Read More
  </a>
</div>
</div>
</div>
</template>
```

우리는 라는 prop을 지정하고 있는데 job, 이는 객체가 될 것입니다. 그런 다음 템플릿에서 해당 prop을 사용하여 작업 데이터를 출력합니다.

이제 컴포넌트에서 컴포넌트를 가져와서 각 작업을 prop으로 전달 App해야 합니다. 전체 섹션을 다음으로 바꾸세요. JobListing

```
<section class="bg-green-50 px-4 py-10">
  <div class="container-xl lg:container m-auto">
    <h2 class="text-3xl font-bold text-green-500 mb-6 text-center">
      Browse Jobs
    </h2>
    <div class="grid grid-cols-1 md:grid-cols-3 gap-6">
      <!-- Iterate over jobs array -->
      <JobListing v-for="job in jobs" :key="job.id" :job="job" />
    </div>
  </div>
</section>
```

배열을 반복하면서 jobs각 직무를 prop으로 JobListing컴포넌트에 전달합니다. 여전히 직무 목록을 볼 수 있습니다.

JobListings 구성 요소

이것을 조금 정리해서 이 섹션과 그 아래 섹션(모든 작업 보기 링크)을 라는 컴포넌트에 넣어보겠습니다 JobListings. 파일을 만들고 두 섹션을 잘라내어 붙여넣습니다. 또한 작업 데이터와 컴포넌트 및 기타 항목 src/components/JobListings.vue에 대한 가져오기를 이동해야 합니다. 반드시 한 단계 위로 파일을 이동하세요. 다음과 같아야 합니다. JobListingrefjobs.json

```
<script setup>
import JobListing from '@/components/JobListing.vue';
import jobData from '../jobs.json';

import { ref } from 'vue';

const jobs = ref(jobData);
</script>

<template>
  <section class="bg-green-50 px-4 py-10">
    <div class="container-xl lg:container m-auto">
      <h2 class="text-3xl font-bold text-green-500 mb-6 text-center">
        Browse Jobs
      </h2>
      <div class="grid grid-cols-1 md:grid-cols-3 gap-6">
        <!-- Iterate over jobs array -->
        <JobListing v-for="job in jobs" :key="job.id" :job="job" />
      </div>
    </div>
  </section>

  <section class="m-auto max-w-lg my-10 px-6">
    <a
```

```

    href="jobs.html"
    class="block bg-black text-white text-center py-4 px-6 rounded-xl hover:bg-gray-700"
    >View All Jobs</a>
  >
</section>
</template>

```

이제 App.vue 다음이 있어야 합니다.

```

<script setup>
import Navbar from '@/components/Navbar.vue';
import Hero from '@/components/Hero.vue';
import HomeCards from '@/components/HomeCards.vue';
import JobListings from '@/components/JobListings.vue';
</script>

<template>
<Navbar />
<Hero />
<HomeCards />
<JobListings />
</template>

```

우리는 이 파일을 정말 정리했습니다.

채용 공고 제한

홈페이지에서 우리는 3개의 구인 공고만 보여주고 싶습니다. 컴포넌트 `limit`에 `prop` 을 추가하고 `JobListings`으로 설정해 보겠습니다 3. 그런 다음 컴포넌트에서 `prop`으로 App 전달합니다 `.limit`

스크립트를 변경하여 `JobListings.vue` `limit prop`을 사용합니다.

```

<script setup>
import JobListing from '@/components/JobListing.vue';
import jobData from '../jobs.json';

import { ref, defineProps } from 'vue';

const props = defineProps({
  limit: Number,
});

const jobs = ref(jobData);
</script>

```

이제, 와 를 `v-for` 다음 `JobListing`과 같이 변경합니다.

```

<JobListing
  v-for="(job, index) in jobs.slice(0, limit || jobs.length)"
  :key="job.id || index"
  :job="job"
/>

```

이렇게 하면 작업이 `limit prop` 값으로 제한됩니다. 전달되지 않으면 모든 작업이 표시됩니다.

이제 컴포넌트에서 `prop`으로 App 전달합니다 `.limit`

```
<JobListings :limit="3" />
```

우리가 를 추가하는 이유는 :우리가 그것을 바인딩하고 3을 숫자로 전달하기 때문입니다. 우리가 그것을 사용하지 않으면 그것은 문자열이 될 것입니다.

이제 홈페이지에 세 개의 목록만 표시될 것입니다.

showButton소품

홈 및 채용 페이지에서 이것을 사용할 것입니다. 채용 페이지에 "모든 채용 정보 보기" 버튼은 원하지 않습니다. 따라서 `showButton` and only show if that prop is passed in true라는 prop을 추가해 보겠습니다.

```
const props = defineProps({
  limit: Number,
  showButton: {
    type: Boolean,
    default: false,
  },
});
```

이제 버튼이 있는 하단 섹션에 조건을 추가합니다.

```
<section v-if="showButton" class='m-auto max-w-lg my-10 px-6'>
  <RouterLink
    to='/jobs'
    class='block bg-black text-white text-center py-4 px-6 rounded-xl hover:bg-gray-700'
  >
    View All Jobs
  </RouterLink>
</section>
```

이제 :에 있는 embed에 prop을 추가하세요 HomeView.

```
<JobListings :limit="3" :showButton="true" />
```

computed& 단축된 설명

함수에 대해 이야기해 봅시다 `computed`. 이것은 다른 값에 따라 값을 반환하는 함수입니다. 이 함수는 해당 값이 변경될 때마다 실행됩니다. React를 알고 있다면, 이것은 후크의 종속성 배열과 비슷합니다 `useEffect()`. 해당 종속성이 변경되면 효과가 실행됩니다. 이를 사용하여 설명의 단축된 버전을 보여 봅시다.

다음을 `JobListing.vue` 구성 요소에 추가합니다 `<script>`.

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

const props = defineProps({
  job: Object,
});

const showFullDescription = ref(false);

const toggleDescription = () => {
  showFullDescription.value = !showFullDescription.value;
};

const truncatedDescription = computed(() => {
  let description = props.job.description;
  if (!showFullDescription.value) {
    description = description.substring(0, 90) + '...';
  }
  return description;
});
```

먼저 라는 반응형 변수를 만들어 `showFullDescription`로 설정합니다 `false`. 그런 다음 `toggleDescription`값을 토글하는 라는 함수를 만듭니다 `showFullDescription`. 그런 다음 `truncatedDescription` 전체 설명을 반환하고 이면 잘린 버전을 반환하는 라는 `showFullDescription` 계산 `true`된 속성을 만듭니다 `false`. 이제 다음 코드를 바꿉니다.

```
<div class="mb-5">{{ job.description }}</div>
```

이것으로:

```
<div>{{ truncatedDescription }}</div>

<button
  @click="toggleDescription"
  class="text-green-500 hover:text-green-600 mb-5"
>
  {{ showFullDescription ? 'Less' : 'More' }}
</button>
```

이제 잘린 설명을 전환할 수 있습니다.

아이콘(PrimeIcons)

우리는 프로젝트에서 몇 가지 아이콘을 사용할 것입니다. 아이콘 라이브러리는 여러 개 있습니다. 우리가 사용할 라이브러리는입니다 `primeIcons`. 설치해 보겠습니다.

```
npm install primeicons
```

이제 다음을 바꾸세요.

```
<i className='fa-solid fa-location-dot text-lg'></i>
```

다음과 같은 사항 포함:

```
<i class='pi pi-map-marker text-orange-700'></i>
```

이제 지도 마커 아이콘이 보일 것입니다. 사용 가능한 모든 아이콘은 여기에서 볼 수 있습니다 - <https://primevue.org/icons/>

뷰 라우터

지금까지 우리는 모든 구성 요소에 하나의 페이지/URL만 사용했습니다. 실제 애플리케이션에서는 여러 페이지가 있고 그 사이를 탐색해야 합니다. 여기서 Vue Router가 등장합니다. Vue Router는 Vue.js의 공식 라우터입니다. 라우터가 필요한 이유는 구인 및 단일 구인 목록을 위한 별도의 페이지와 목록을 추가하고 업데이트하는 페이지가 있기 때문입니다. 처음부터 라우터를 구현하기로 선택할 수도 있지만, 직접 구현하는 방법을 보여드리고 싶었습니다.

Vue Router를 설치해 보겠습니다.

```
npm install vue-router
```

이제 라우터를 설정해야 합니다. 파일을 만들고 `src/router/index.js` 다음을 추가합니다.

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

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView,
    },
  ],
});
```

```
],
});

export default router;
```

`createRouter` 우리는 에서 및 `createWebHistory`를 가져옵니다 `vue-router`. 그런 다음 함수를 사용하여 라우터 인스턴스를 만듭니다 `createRouter`. 우리는 `history` 모드와 경로를 전달합니다. 우리는 기본 URL을 설정합니다 `import.meta.env.BASE_URL`. 이는 앱의 기본 URL입니다. 그런 다음 라우터 인스턴스를 내보냅니다. 우리는 아직 만들지 않은 허브를 로드하기 위한 허브 URL에 대한 하나의 경로를 가지고 있습니다.

이제 파일을 열고 `src/main.js` 라우터를 적용해 보겠습니다.

```
import { createApp } from 'vue';
import App from './App.vue';
import '@/assets/main.css';
import 'primeicons/primeicons.css';
import router from './router';

const app = createApp(App);

app.use(router);

app.mount('#app');
```

우리는 라우터를 가져왔고, 그런 다음 `createApp` 함수에서 사용하고 있습니다. 이제 우리는 라우터를 컴포넌트에서 사용할 수 있습니다.

조회수

`views` 폴더에 폴더를 만들 것입니다 `src`. 여기에 페이지로 사용될 구성 요소를 넣을 것입니다. 파일을 만들고 구성 요소에서 `src/views/HomeView.vue` `Navbar` 구성 요소를 제외한 모든 것을 `App.vue` 그 안으로 옮깁니다. 모든 페이지에 `Navbar`를 원하므로 구성 요소에 그대로 둡니다 `App.vue`. `HomeView.vue` 파일은 다음과 같아야 합니다.

```
<script setup>
import Hero from '@/components/Hero.vue';
import HomeCards from '@/components/HomeCards.vue';
import JobListings from '@/components/JobListings.vue';
</script>

<template>
<Hero />
<HomeCards />
<JobListings :limit="3" />
</template>
```

이제 `App.vue` 파일에 다음이 포함되어야 합니다.

```
<script setup>
import { RouterView } from 'vue-router';
import Navbar from '@/components/Navbar.vue';
</script>

<template>
<Navbar />
<RouterView />
</template>
```

구성 `RouterView` 요소는 현재 활성화된 구성 요소를 렌더링하는 특수 구성 요소입니다. 이 출력의 다른 모든 것은 `navbar`과 같은 모든 페이지에 있습니다.

이상한 오류가 발생하면 `node_modules` 폴더를 삭제하고 종속성을 다시 설치해보세요 `npm install`.

이제 모든 것이 이전과 동일하게 보일 것입니다.

일자리 보기

새로운 페이지와 작업 페이지에 대한 경로를 만들어 보겠습니다. 이렇게 하면 모든 작업이 나열됩니다. `JobListings`을 위해 구성 요소를 사용할 수 있습니다.

파일을 만들고 `src/views/JobsView.vue` 다음을 추가합니다.

```
<script setup>
import JobListings from '@/components/JobListings.vue';
</script>

<template>
<JobListings />
</template>
```

일자리 경로

이제 이 페이지에 대한 경로를 파일에 추가해야 합니다 `router/index.js`.

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

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView,
    },
    {
      path: '/jobs',
      name: 'jobs',
      component: JobsView,
    },
  ],
});

export default router;
```

이제 <http://localhost:3000/jobs>로 이동하면 채용 정보 페이지가 보일 것입니다.

모래밭

이제 페이지 사이를 탐색할 링크를 추가해야 합니다. 컴포넌트를 사용하여 `router-link` 링크를 만들 수 있습니다. 컴포넌트로 이동하여 컴포넌트를 `Navbar.vue`에 임포트해 보겠습니다. `router-link`.

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

이제 모든 태그를 구성 요소로 대체하고 싶습니다 `router-link`. 로고부터 시작해 보겠습니다. 다음과 같아야 합니다.

```
<RouterLink class='flex flex-shrink-0 items-center mr-4' to='/'>
  <img class='h-10 w-auto' v-bind:src='logo' alt='Vue Jobs' />
  <span class='hidden md:block text-white text-2xl font-bold ml-2'>
    Vue Jobs
  </span>
</RouterLink>
```

`href` 반드시로 변경 `to`하고 `index.html`로 변경하세요 /.

이제 탐색 링크를 만드세요:

```
<div class='md:ml-auto'>
  <div class='flex space-x-2'>
    <RouterLink
      to='/'
      class='text-white bg-green-900 hover:bg-gray-900 hover:text-white rounded-md px-3 py-2'>
      Home
    </RouterLink>
    <RouterLink
      to='/jobs'
      class='text-white hover:bg-green-900 hover:text-white rounded-md px-3 py-2'>
      Jobs
    </RouterLink>
    <RouterLink
      to='/jobs/add'
      class='text-white hover:bg-green-900 hover:text-white rounded-md px-3 py-2'>
      Add Job
    </RouterLink>
  </div>
</div>
```

활성 링크

라우팅은 작동해야 하지만, 활성 링크는 항상 흄 링크입니다. 활성 링크가 현재 페이지인지 확인하기 위해 몇 가지 논리를 추가해야 합니다.

`useRoute` 페키지에서 가져온 내용은 다음과 같습니다 `vue-router`.

```
import { RouterLink, useRoute } from 'vue-router';
```

이제 다음 기능을 추가하세요.

```
// Function to determine if the link is active
const isActiveLink = (routePath) => {
  const route = useRoute();
  return route.path === routePath;
};
```

우리는 단순히 현재 경로 경로가 우리가 전달하는 경로 경로와 같은지 확인하고 있습니다. 이제 이 함수를 사용하여 링크가 활성화되었는지 확인할 수 있습니다.

```
<div class="flex space-x-2">
  <RouterLink
    to="/"
    :class="[
      'text-white',
      isActiveLink('/')
        ? 'bg-green-900'
        : 'hover:bg-gray-900 hover:text-white',
      'rounded-md',
      'px-3',
      'py-2',
    ]"
    >Home</RouterLink>
  >
  <RouterLink
    to="/jobs"
    :class="[
      'text-white',
      isActiveLink('/jobs')
        ? 'bg-green-900'
        : 'hover:bg-green-900 hover:text-white',
      'rounded-md',
      'px-3',
      'py-2',
    ]"
    >Jobs</RouterLink>
</div>
```

```

        ]"
      >Jobs</RouterLink
    >
    <RouterLink
      to="/jobs/add"
      :class="[
        'text-white',
        isActiveLink('/jobs/add')
          ? 'bg-green-900'
          : 'hover:bg-green-900 hover:text-white',
        'rounded-md',
        'px-3',
        'py-2',
      ]"
      >Add Job</RouterLink
    >
</div>

```

이제 올바른 링크가 활성화되어야 합니다.

모든 일자리 링크 보기

View All Jobs 링크에 올바른 링크를 추가해 보겠습니다. 컴포넌트에서 JobListingsimport RouterLink:

```
import { RouterLink } from 'vue-router';
```

그런 다음 링크/버튼이 있는 섹션을 다음과 같이 바꾸세요.

```

<section class="m-auto max-w-lg my-10 px-6">
  <RouterLink
    to="/jobs"
    class="block bg-black text-white text-center py-4 px-6 rounded-xl hover:bg-gray-700">
    >
    View All Jobs
  </RouterLink>
</section>

```

"개발자를 위한" 섹션의 링크도 편집해야 합니다. src/components/HomeCards.vue 파일을 업니다. 두 링크를 모두 올바른 경로로 변경해 보겠습니다.

```

<script setup>
import Card from '@/components/Card.vue';
import { RouterLink } from 'vue-router';
</script>

<template>
<section class="py-4">
  <div class="container-xl lg:container m-auto">
    <div class="grid grid-cols-1 md:grid-cols-2 gap-4 p-4 rounded-lg">
      <Card>
        <h2 class="text-2xl font-bold">For Developers</h2>
        <p class="mt-2 mb-4">
          Browse our Vue jobs and start your career today
        </p>
        <RouterLink
          to="/jobs"
          class="inline-block bg-black text-white rounded-lg px-4 py-2 hover:bg-gray-700">
          >
          Browse Jobs
        </RouterLink>
      </Card>
      <Card bg="bg-green-100">
        <h2 class="text-2xl font-bold">For Employers</h2>
        <p class="mt-2 mb-4">
          List your job to find the perfect developer for the role
        </p>
        <RouterLink

```

```

        to="/jobs/add"
        class="inline-block bg-green-500 text-white rounded-lg px-4 py-2 hover:bg-green-600"
      >
      Add Job
    </RouterLink>
  </Card>
</div>
</div>
</section>
</template>

```

찾을 수 없는 페이지

지금, 존재하지 않는 경로를 방문하면 빈 페이지가 표시됩니다. 간단한 찾을 수 없음 페이지를 만들어 보겠습니다. 에서 파일을 만들고 `src/views/NotFoundView.vue` 다음을 추가합니다.

```

<script setup>
import { RouterLink } from 'vue-router';
</script>

<template>
<section class="text-center flex flex-col justify-center items-center h-96">
  <i class="pi pi-exclamation-triangle text-yellow-500 text-7xl mb-5"></i>
  <h1 class="text-6xl font-bold mb-4">404 Not Found</h1>
  <p class="text-xl mb-5">This page does not exist</p>
  <RouterLink
    to="/"
    class="text-white bg-green-700 hover:bg-green-900 rounded-md px-3 py-2 mt-4"
  >
    Go Back
  </RouterLink>
</section>
</template>

```

이제 이 페이지에 대한 경로를 파일에 추가해야 합니다 `router/index.js`.

```

import NotFoundView from '@/views/NotFoundView.vue';

//...

{
  path: '/:catchAll(.*)', // Match any path that hasn't been matched by other routes
  name: 'not-found',
  component: NotFoundView,
},

```

단일 작업 보기

단일 작업 뷰를 만들어 보겠습니다. .에서 파일을 만듭니다 `src/views/JobView.vue`. `theme_files` 파일에서 섹션을 복사하여 `job.html` 템플릿에서 사용할 수 있습니다.

```

<template>
<section class="bg-green-50">
  <div class="container m-auto py-10 px-6">
    <div class="grid grid-cols-1 md:grid-cols-70/30 w-full gap-6">
      <main>
        <div
          class="bg-white p-6 rounded-lg shadow-md text-center md:text-left"
        >
          <div class="text-gray-500 mb-4">Full-Time</div>
          <h1 class="text-3xl font-bold mb-4">Senior Vue Developer</h1>
          <div
            class="text-gray-500 mb-4 flex align-middle justify-center md:justify-start"
          >
            <1

```

```

        class="fa-solid fa-location-dot text-lg text-orange-700 mr-2"
      ></i>
      <p class="text-orange-700">Boston, MA</p>
    </div>
  </div>

  <div class="bg-white p-6 rounded-lg shadow-md mt-6">
    <h3 class="text-green-800 text-lg font-bold mb-6">
      Job Description
    </h3>

    <p class="mb-4">
      We are seeking a talented Front-End Developer to join our team in
      Boston, MA. The ideal candidate will have strong skills in HTML,
      CSS, and JavaScript, with experience working with modern
      JavaScript frameworks such as Vue or Angular.
    </p>

    <h3 class="text-green-800 text-lg font-bold mb-2">Salary</h3>

    <p class="mb-4">$70k - $80K / Year</p>
  </div>
</main>

<!-- Sidebar -->
<aside>
  <!-- Company Info -->
  <div class="bg-white p-6 rounded-lg shadow-md">
    <h3 class="text-xl font-bold mb-6">Company Info</h3>

    <h2 class="text-2xl">NewTek Solutions</h2>

    <p class="my-2">
      NewTek Solutions is a leading technology company specializing in
      web development and digital solutions. We pride ourselves on
      delivering high-quality products and services to our clients while
      fostering a collaborative and innovative work environment.
    </p>

    <hr class="my-4" />

    <h3 class="text-xl">Contact Email:</h3>

    <p class="my-2 bg-green-100 p-2 font-bold">
      contact@newteksolutions.com
    </p>

    <h3 class="text-xl">Contact Phone:</h3>

    <p class="my-2 bg-green-100 p-2 font-bold">555-555-5555</p>
  </div>

  <!-- Manage -->
  <div class="bg-white p-6 rounded-lg shadow-md mt-6">
    <h3 class="text-xl font-bold mb-6">Manage Job</h3>
    <a
      href="add-job.html"
      class="bg-green-500 hover:bg-green-600 text-white text-center font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline mt-4"
      >Edit Job</a>
    >
    <button
      class="bg-red-500 hover:bg-red-600 text-white font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline mt-4 block"
    >
      Delete Job
    </button>
  </div>
</aside>
</div>
</div>
</section>
</template>

```

이제 파일에 경로를 생성하세요 router/index.js.

```
import JobView from '@/views/JobView.vue';
//...
{
  path: '/jobs/:id',
  name: 'job',
  component: JobView,
},
}
```

<http://localhost:8080/jobs/1> 브라우저에 넣으면 작업 보기 가 보일 것입니다.

구성 요소의 링크를 업데이트해 보겠습니다 `src/components/JobListing.vue`.

```
import { RouterLink } from 'vue-router';
//...
<RouterLink
:to="/jobs/" + job.id"
class="h-[36px] bg-green-500 hover:bg-green-600 text-white px-4 py-2 rounded-lg text-center text-sm"
>
  Read More
</RouterLink>
```

라우터 링크를 가져와서 버튼에 추가했습니다. 작업 ID가 있는 동적 경로를 사용했기 때문에 `:to`를 사용해야 합니다 `:to`.

분명히 지금 당장은 데이터가 하드코딩된 HTML일 뿐입니다. 백엔드 서버를 만들고 거기에서 데이터를 가져올 때까지는 그대로 두겠습니다. 이제 JSON 서버로 해보겠습니다.

JSON 서버 및 데이터 가져오기

지금 당장, 우리의 데이터는 JSON 파일에서 나옵니다. 우리는 이것을 실제 애플리케이션처럼 작동시키고 싶으므로 JSON Server를 사용하여 가짜 REST API를 만들 것입니다. JSON Server는 코딩이 전혀 없는 완전한 가짜 REST API입니다.

먼저 JSON 서버를 설치해 보겠습니다.

```
npm install json-server
```

`jobs.json` 이제 우리가 사용하던 파일을 사용할 수 있습니다. 그러나 배열의 이름을 지정해야 합니다 `jobs`. 따라서 파일을 열고 모든 것을 둘러싼 중괄호를 추가하고 `jobs` 다음과 같이 배열의 이름을 추가합니다.

```
{
  "jobs": [
    {
      "id": 1,
      "title": "Senior Vue Developer",
      "type": "Full-Time",
      "description": "We are seeking a talented Front-End Developer to join our team in Boston, MA. The ideal candidate will have strong skills in HTML, CSS, and",
      "location": "Boston, MA",
      "salary": "$70K - $80K",
      "company": {
        "name": "NewTek Solutions",
        "description": "NewTek Solutions is a leading technology company specializing in web development and digital solutions. We pride ourselves on delivering",
        "contactEmail": "contact@teksolutions.com",
        "contactPhone": "555-555-5555"
      }
    },
    {
      "id": 2,
      "title": "Front-End Engineer (Vue)",
      "type": "Full-Time",
      "description": "Join our team as a Front-End Developer in sunny Miami, FL. We are looking for a motivated individual with a passion for crafting beautiful",
      "location": "Miami, FL",
      "salary": "$70K - $80K",
    }
  ]
},
```

```

"company": {
  "name": "Veneer Solutions",
  "description": "Veneer Solutions is a creative agency specializing in digital design and development. Our team is dedicated to pushing the boundaries of
  "contactEmail": "contact@loremipsum.com",
  "contactPhone": "555-555-5555"
}
},
{
  "id": 3,
  "title": "Vue.js Developer",
  "type": "Full-Time",
  "description": "Are you passionate about front-end development? Join our team in vibrant Brooklyn, NY, and work on exciting projects that make a difference
  "location": "Brooklyn, NY",
  "salary": "$70K - $80K",
  "company": {
    "name": "Dolor Cloud",
    "description": "Dolor Cloud is a leading technology company specializing in digital solutions for businesses of all sizes. With a focus on innovation and
    "contactEmail": "contact@dolorsitamet.com",
    "contactPhone": "555-555-5555"
  }
},
{
  "id": 4,
  "title": "Vue Front-End Developer",
  "type": "Part-Time",
  "description": "Join our team as a Part-Time Front-End Developer in beautiful Phoenix, AZ. We are looking for a self-motivated individual with a passion fo
  "location": "Phoenix, AZ",
  "salary": "$60K - $70K",
  "company": {
    "name": "Alpha Elite",
    "description": "Alpha Elite is a dynamic startup specializing in digital marketing and web development. We are committed to fostering a diverse and inclu
    "contactEmail": "contact@adipisicingelit.com",
    "contactPhone": "555-555-5555"
  }
},
{
  "id": 5,
  "title": "Full Stack Vue Developer",
  "type": "Full-Time",
  "description": "Exciting opportunity for a Full-Time Front-End Developer in bustling Atlanta, GA. We are seeking a talented individual with a passion for b
  "location": "Atlanta, GA",
  "salary": "$90K - $100K",
  "company": {
    "name": "Browning Technologies",
    "description": "Browning Technologies is a rapidly growing technology company specializing in e-commerce solutions. We offer a dynamic and collaborative
    "contactEmail": "contact@consecteturadipiscing.com",
    "contactPhone": "555-555-5555"
  }
},
{
  "id": 6,
  "title": "Vue Native Developer",
  "type": "Full-Time",
  "description": "Join our team as a Front-End Developer in beautiful Portland, OR. We are looking for a skilled and enthusiastic individual to help us creat
  "location": "Portland, OR",
  "salary": "$100K - $110K",
  "company": {
    "name": "Port Solutions INC",
    "description": "Port Solutions is a leading technology company specializing in software development and digital marketing. We are committed to providing
    "contactEmail": "contact@ipsumlorem.com",
    "contactPhone": "555-555-5555"
  }
}
]
}

```

이제 구인 공고가 깨졌지만 괜찮습니다.

JSON 서버 스크립트

JSON 서버를 실행하려면 NPM 스크립트를 만들어야 합니다. package.json 파일을 열고 scripts 아래에 다음을 추가합니다.

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview",
  "server": "json-server --watch src/jobs.json --port 5000" // Add this
},
```

이제 새 터미널을 열고 `npm run server`. 을 실행하세요.

열면 `http://localhost:5000/jobs` 데이터가 보입니다.

데이터 가져오기

이제 JSON 서버에서 데이터를 가져와야 합니다. 저는 라이브러리를 사용할 것이라 Fetch API를 사용하여 이를 수행할 수도 있습니다. 구성 요소가 마운트될 때 데이터를 가져오려면 라이프사이클 허크를 Axios 사용할 것입니다 `.onMounted`

먼저 Axios를 설치해 보겠습니다.

```
npm install axios
```

스크립트는 다음과 같습니다.

```
<script setup>
import JobListing from '@/components/JobListing.vue';
import { RouterLink } from 'vue-router';
import axios from 'axios';

import { ref, defineProps, onMounted } from 'vue';

const props = defineProps({
  limit: Number,
});

const jobs = ref([]);

onMounted(async () => {
  try {
    const response = await axios.get('http://localhost:5000/jobs');
    jobs.value = response.data;
  } catch (error) {
    console.error('Error fetching jobs:', error);
  }
});
</script>
```

그래서 마운트할 때 JSON 서버에서 데이터를 가져와서 `jobs`에 설정합니다. 오류가 있으면 콘솔에 로깅합니다.

이제 다시 구인 공고를 볼 수 있지만 이제는 서버에서 나옵니다. 이는 파일에 저장하는 것보다 더 현실적입니다.

사용 중 reactive

우리가 이렇게 하는 것은 괜찮을 것이지만, `reactive` 대신을 사용하여 다른 방법을 보여드리고 싶습니다 `ref`. 하나의 변수에 상태에 대한 다른 값을 가진 객체를 갖고 싶다면, 예를 들어 일자리를 원하지만 데이터를 가져올 때 `true`로 설정하고 완료되면 `false`로 설정할 수 있는 로딩 값도 원한다면 `reactive` 대신을 사용할 수 있습니다 `ref`.

ref vs 반응형

몇 가지 차이점은 다음과 같습니다.

- `reactive` 객체만 받습니다. 문자열, 숫자, 부울과 같은 기본형은 받지 않습니다.
- `ref` 객체나 원시 객체를 사용할 수 있습니다. 후드 아래를 사용합니다.
- `ref`의 `.value` 재할당을 위한 속성이 있지만 `reactive` 사용하지 않으면 `.value` 재할당할 수 없습니다.

React에서 왔다면, 제가 보는 방식은 React에서 `useState`문자열, 이름 같은 것에 `a`를 사용한다면, 저는 Vue에서 를 사용할 것입니다 . React `ref`에서 객체를 사용한다면, 저는 Vue에서 를 사용할 것입니다. 이는 결코 확정된 관례가 아니며, 제가 개인적으로 생각하는 방식일 뿐입니다.

다음과 같이 바꿔보겠습니다.

```
<script setup>
import JobListing from '@/components/JobListing.vue';
import { RouterLink } from 'vue-router';
import axios from 'axios';

import { reactive, defineProps, onMounted } from 'vue';

const props = defineProps({
  limit: Number,
});

const state = reactive({
  jobs: [],
  isLoading: true,
});

onMounted(async () => {
  try {
    const response = await axios.get('http://localhost:5000/jobs');
    state.jobs = response.data;
  } catch (error) {
    console.error('Error fetching jobs:', error);
  } finally {
    state.isLoading = false;
  }
});
</script>
```

이제 우리는 및 값을 `state`포함하는 객체를 가지고 있습니다. 데이터를 페치할 때 값을 설정한 다음 완료할 때 값을 설정할 수 있습니다.
`jobs`
`isLoading`
`true`
`false`

스피너를 보여주기 위해 매우 간단한 라이브러리를 설치해 보겠습니다. 다음을 실행하세요.

```
npm install vue-spinner
```

맨 위에서 가져옵니다:

```
import PulseLoader from 'vue-spinner/src/PulseLoader.vue';
```

그러면 템플릿을 다음과 같이 변경할 수 있습니다.

```
<template>
<section class="bg-blue-50 px-4 py-10">
  <div class="container-xl lg:container m-auto">
    <h2 class="text-3xl font-bold text-green-500 mb-6 text-center">
      Browse Jobs
    </h2>
    <!-- Show loading spinner or placeholder while isLoading is true -->
    <div v-if="state.isLoading" class="text-center text-gray-500 py-6">
      <PulseLoader />
    </div>

    <!-- Show job listings when isLoading is false -->
    <div v-else class="grid grid-cols-1 md:grid-cols-3 gap-6">
      <JobListing
        v-for="(job, index) in state.jobs.slice(
          0,
          limit || state.jobs.length
        )"
        :key="job.id || index"
        :job="job"
      />
    </div>
  </div>
</section>
```

```

</div>
</div>
</section>

<section class="m-auto max-w-lg my-10 px-6">
  <RouterLink
    to="/jobs"
    class="block bg-black text-white text-center py-4 px-6 rounded-xl hover:bg-gray-700"
  >View All Jobs</RouterLink>
</section>
</template>

```

이제 데이터를 가져오는 동안 표시되고 Loading 그런 다음 구인 공고가 표시됩니다. 이렇게 할 필요는 없고, 그냥 또 다른 옵션일 뿐입니다.

스피너가 작동하는 모습을 보기 위해 속도를 늦추려면 다음을 사용할 수 있습니다 `setTimeout`.

```

onMounted(async () => {
  try {
    const response = await axios.get('http://localhost:5000/jobs');
    // Simulate a 2-second delay
    await new Promise((resolve) => setTimeout(resolve, 2000));
    state.jobs = response.data;
  } catch (error) {
    console.error('Error fetching jobs:', error);
  } finally {
    // Set isLoading to false after the delay
    state.isLoading = false;
  }
});

```

단일 작업에 대한 데이터

단일 작업 보기에는 하드코딩된 HTML만 있습니다. 서버에서 작업을 가져와서 페이지에 표시해 보겠습니다.

다음 스크립트를 추가하여 데이터를 가져와 상태에 넣습니다.

```

<script setup>
import PulseLoader from 'vue-spinner/src/PulseLoader.vue';
import axios from 'axios';

import { reactive, onMounted } from 'vue';
import { useRoute, RouterLink } from 'vue-router';

const route = useRoute();

const jobId = route.params.id;

const state = reactive({
  job: {},
  isLoading: true,
});

onMounted(async () => {
  try {
    const response = await axios.get(`http://localhost:5000/jobs/${jobId}`);
    state.job = response.data;
  } catch (error) {
    console.error('Error fetching job:', error);
  } finally {
    state.isLoading = false;
  }
});
</script>

```

우리는 후크를 사용하여 URL에서 ID를 얻을 수 있습니다 `useRoute`. 그런 다음 이를 사용하여 `jobId` 서버에서 작업을 페치하여 `state` 객체에 넣을 수 있습니다.

Vue 개발자 도구를 열어 데이터가 상태에 있는지 확인할 수 있습니다.

템플릿에 다음을 추가하세요.

```
<template>
<section v-if="!state.isLoading" class="bg-green-50">
  <div class="container m-auto py-10 px-6">
    <div class="grid grid-cols-1 md:grid-cols-70/30 w-full gap-6">
      <main>
        <div
          class="bg-white p-6 rounded-lg shadow-md text-center md:text-left"
        >
          <div class="text-gray-500 mb-4">{{ state.job.type }}</div>
          <h1 class="text-3xl font-bold mb-4">{{ state.job.title }}</h1>
          <div
            class="text-gray-500 mb-4 flex align-middle justify-center md:justify-start"
          >
            <i class="pi pi-map-marker text-orange-700 mr-2 text-xl"></i>
            <p class="text-orange-700 font-bold">{{ state.job.location }}</p>
          </div>
        </div>
      </div>

      <div class="bg-white p-6 rounded-lg shadow-md mt-6">
        <h3 class="text-green-800 text-lg font-bold mb-6">
          Job Description
        </h3>

        <p class="mb-4">
          {{ state.job.description }}
        </p>

        <h3 class="text-green-800 text-lg font-bold mb-2">Salary</h3>

        <p class="mb-4">{{ state.job.salary }} / Year</p>
      </div>
    </main>

    <!-- Sidebar -->
    <aside>
      <!-- Company Info -->
      <div class="bg-white p-6 rounded-lg shadow-md">
        <h3 class="text-xl font-bold mb-6">Company Info</h3>

        <h2 class="text-2xl">NewTek Solutions</h2>

        <p class="my-2">
          {{ state.job.company.description }}
        </p>

        <hr class="my-4" />

        <h3 class="text-xl">Contact Email:</h3>

        <p class="my-2 bg-green-100 p-2 font-bold">
          {{ state.job.company.contactEmail }}
        </p>

        <h3 class="text-xl">Contact Phone:</h3>

        <p class="my-2 bg-green-100 p-2 font-bold">
          {{ state.job.company.contactPhone }}
        </p>
      </div>

      <!-- Manage -->
      <div class="bg-white p-6 rounded-lg shadow-md mt-6">
        <h3 class="text-xl font-bold mb-6">Manage Job</h3>
        <RouterLink
          :to="/jobs/edit/" + state.job.id"
          class="bg-green-500 hover:bg-green-600 text-white text-center font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline mt-4"
        >Edit Job</RouterLink>
        <button
          class="bg-red-500 hover:bg-red-600 text-white font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline mt-4 block">
          Delete Job
        </button>
      </div>
    </aside>
  </div>
</section>
```

```

        </button>
    </div>
</aside>
</div>
</div>
</section>

<!-- Loading State --&gt;
&lt;div v-else class="text-center text-gray-500 py-6"&gt;
    &lt;PulseLoader /&gt;
&lt;/div&gt;
&lt;/div&gt;
&lt;/template&gt;
</pre>

```

이제 단일 작업 보기 페이지에서 작업 데이터를 볼 수 있습니다.

뒤로가기 버튼

채용정보 페이지로 쉽게 돌아갈 수 있는 뒤로가기 버튼 구성 요소를 만들어 보겠습니다.

BackButton.vue 폴더에서 라는 새 파일을 만듭니다 components. 다음을 추가합니다.

```

<script setup>
import { RouterLink } from 'vue-router';
</script>

<template>
<section>
    <div class="container m-auto py-6 px-6">
        <RouterLink
            to="/jobs"
            class="text-green-500 hover:text-green-600 flex items-center">
            >
                <i class="fas fa-arrow-left mr-2"></i> Back to Job Listings
            </RouterLink>
        </div>
    </section>
</template>

```

이제 이것을 JobView.vue 구성 요소로 가져오세요:

```
import BackButton from '@/components/BackButton.vue';
```

템플릿의 섹션 위에 삽입:

```

<template>
    <BackButton />
    <section v-if="!state.isLoading" class="bg-green-50">
        // ...
    </template>

```

프록시 요청

지금 당장 데이터를 가져오라는 요청을 할 때, 우리는 에 요청을 합니다 <http://localhost:5000/jobs>. 이는 우리가 에서 서버를 실행하고 있기 때문입니다 <http://localhost:5000>. 하지만 앱을 배포할 때, 우리는 에 요청을 할 수 없습니다 <http://localhost:5000>. 우리는 앱이 호스팅되는 동일한 도메인에 요청을 해야 합니다. 우리는 이를 위해 프록시를 사용할 수 있습니다.

vue.config.js 프로젝트 루트를 열고 proxy 다음 설정으로 객체를 추가합니다.

```
export default defineConfig({
    plugins: [vue()],
    resolve: {
```

```

alias: {
  '@': fileURLToPath(new URL('./src', import.meta.url)),
},
},
server: {
  port: 3000,
  proxy: {
    '/api': {
      target: 'http://localhost:5000',
      changeOrigin: true,
      rewrite: (path) => path.replace(/^\api/, ''),
    },
  },
},
);

```

이 작업은 URL을 . 또는 우리의 도메인 <http://localhost:5000/jobs>으로 바꾸는 것입니다.<http://localhost:3000/api/jobs>

이제 요청의 URL을 . axios으로 변경할 수 있습니다. 파일과 파일 `/api/jobs`에서 이 작업을 수행합니다. `src/components/JobListings.vue`
`src/views/JobView.vue`

작업 추가 페이지

이제 새 작업을 추가할 페이지를 만들어 보겠습니다. 파일을 만들고 `src/views/AddJobView.vue` 지금은 다음을 추가합니다.

```
<template>This is the add page</template>
```

파일에 추가하세요 `router/index.js`:

```

import AddJobView from '@/views/AddJobView.vue';

//...

{
  path: '/jobs/add',
  name: 'add-job',
  component: AddJobView,
},

```

이제 .에서 작업 추가 페이지를 볼 수 있습니다 <http://localhost:3000/jobs/add>.

theme_files 페이지에서 html을 복사 `add-job.html`하여 템플릿에 붙여넣습니다.

```

<template>
<section class="bg-green-50">
  <div class="container m-auto max-w-2xl py-24">
    <div
      class="bg-white px-6 py-8 mb-4 shadow-md rounded-md border m-4 md:m-0"
    >
      <form>
        <h2 class="text-3xl text-center font-semibold mb-6">Add Job</h2>

        <div class="mb-4">
          <label for="type" class="block text-gray-700 font-bold mb-2">
            Job Type
          </label>
          <select
            id="type"
            name="type"
            class="border rounded w-full py-2 px-3"
            required
          >
            <option value="Full-Time">Full-Time</option>
            <option value="Part-Time">Part-Time</option>
            <option value="Remote">Remote</option>
            <option value="Internship">Internship</option>
          </select>
        </div>
      </form>
    </div>
  </div>
</section>

```

```
</div>

<div class="mb-4">
  <label class="block text-gray-700 font-bold mb-2">
    >Job Listing Name</label>
  >
  <input type="text" id="name" name="name" class="border rounded w-full py-2 px-3 mb-2" placeholder="eg. Beautiful Apartment In Miami" required />
</div>

<div class="mb-4">
  <label for="description" class="block text-gray-700 font-bold mb-2">
    >Description</label>
  >
  <textarea id="description" name="description" class="border rounded w-full py-2 px-3" rows="4" placeholder="Add any job duties, expectations, requirements, etc"></textarea>
</div>

<div class="mb-4">
  <label for="type" class="block text-gray-700 font-bold mb-2">
    >Salary</label>
  >
  <select id="salary" name="salary" class="border rounded w-full py-2 px-3" required>
    <option value="Under $50K">under $50K</option>
    <option value="$50K - $60K">$50 - $60K</option>
    <option value="$60K - $70K">$60 - $70K</option>
    <option value="$70K - $80K">$70 - $80K</option>
    <option value="$80K - $90K">$80 - $90K</option>
    <option value="$90K - $100K">$90 - $100K</option>
    <option value="$100K - $125K">$100 - $125K</option>
    <option value="$125K - $150K">$125 - $150K</option>
    <option value="$150K - $175K">$150 - $175K</option>
    <option value="$175K - $200K">$175 - $200K</option>
    <option value="Over $200K">Over $200K</option>
  </select>
</div>

<div class="mb-4">
  <label class="block text-gray-700 font-bold mb-2"> Location </label>
  <input type="text" id="location" name="location" class="border rounded w-full py-2 px-3 mb-2" placeholder="Company Location" required />
</div>

<h3 class="text-2xl mb-5">Company Info</h3>

<div class="mb-4">
  <label for="company" class="block text-gray-700 font-bold mb-2">
    >Company Name</label>
  >
  <input type="text" id="company" name="company" class="border rounded w-full py-2 px-3" placeholder="Company Name" required />
</div>
```

```

        placeholder="Company Name"
      />
</div>

<div class="mb-4">
  <label
    for="company_description"
    class="block text-gray-700 font-bold mb-2"
  >Company Description</label>
  >
  <textarea
    id="company_description"
    name="company_description"
    class="border rounded w-full py-2 px-3"
    rows="4"
    placeholder="What does your company do?"
  ></textarea>
</div>

<div class="mb-4">
  <label
    for="contact_email"
    class="block text-gray-700 font-bold mb-2"
  >Contact Email</label>
  >
  <input
    type="email"
    id="contact_email"
    name="contact_email"
    class="border rounded w-full py-2 px-3"
    placeholder="Email address for applicants"
    required
  />
</div>
<div class="mb-4">
  <label
    for="contact_phone"
    class="block text-gray-700 font-bold mb-2"
  >Contact Phone</label>
  >
  <input
    type="tel"
    id="contact_phone"
    name="contact_phone"
    class="border rounded w-full py-2 px-3"
    placeholder="Optional phone for applicants"
  />
</div>

<div>
  <button
    class="bg-green-500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline"
    type="submit"
  >
    Add Job
  </button>
</div>
</form>
</div>
</div>
</section>
</template>

```

스크립트에서 reactive 품 데이터에 대한 객체를 가져와서 만들어 보겠습니다. 또한 품 제출을 처리하는 함수도 있습니다.

```

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

const form = reactive({
  type: 'Full-Time',
  title: '',
  description: '',
  salary: '',

```

```

location: '',
company: {
  name: '',
  description: '',
  contactEmail: '',
  contactPhone: '',
},
contact: '',
});

const handleSubmit = (event) => {
  event.preventDefault();
  console.log(form);
};

</script>

```

이제 폼 입력을 데이터와 연결해야 합니다. 다음을 사용하여 이를 수행할 수 있습니다 v-model.

```

<template>
<section class="bg-green-50">
  <div class="container m-auto max-w-2xl py-24">
    <div
      class="bg-white px-6 py-8 mb-4 shadow-md rounded-md border m-4 md:m-0"
    >
      <form @submit.prevent="handleSubmit">
        <h2 class="text-3xl text-center font-semibold mb-6">Add Job</h2>

        <div class="mb-4">
          <label for="type" class="block text-gray-700 font-bold mb-2">
            >Job Type</label>
          <select
            v-model="form.type"
            id="type"
            name="type"
            class="border rounded w-full py-2 px-3"
            required
          >
            <option value="Full-Time">Full-Time</option>
            <option value="Part-Time">Part-Time</option>
            <option value="Remote">Remote</option>
            <option value="Internship">Internship</option>
          </select>
        </div>

        <div class="mb-4">
          <label class="block text-gray-700 font-bold mb-2">
            >Job Listing Name</label>
          <input
            type="text"
            id="name"
            name="name"
            v-model="form.title"
            class="border rounded w-full py-2 px-3 mb-2"
            placeholder="eg. Beautiful Apartment In Miami"
            required
          />
        </div>

        <div class="mb-4">
          <label for="description" class="block text-gray-700 font-bold mb-2">
            >Description</label>
          <textarea
            id="description"
            v-model="form.description"
            name="description"
            class="border rounded w-full py-2 px-3"
            rows="4"
            placeholder="Add any job duties, expectations, requirements, etc"
          ></textarea>
        </div>
    </form>
  </div>
</section>

```

```

<div class="mb-4">
  <label for="type" class="block text-gray-700 font-bold mb-2">
    >Salary</label>
  >
  <select id="salary" v-model="form.salary" name="salary" class="border rounded w-full py-2 px-3" required>
    <option value="Under $50K">under $50K</option>
    <option value="$50 - 60K">$50 - $60K</option>
    <option value="$60 - 70K">$60 - $70K</option>
    <option value="$70 - 80K">$70 - $80K</option>
    <option value="$80 - 90K">$80 - $90K</option>
    <option value="$90 - 100K">$90 - $100K</option>
    <option value="$100 - 125K">$100 - $125K</option>
    <option value="$125 - 150K">$125 - $150K</option>
    <option value="$150 - 175K">$150 - $175K</option>
    <option value="$175 - 200K">$175 - $200K</option>
    <option value="Over $200K">Over $200K</option>
  </select>
</div>

<div class="mb-4">
  <label class="block text-gray-700 font-bold mb-2"> Location </label>
  <input type="text" id="location" v-model="form.location" name="location" class="border rounded w-full py-2 px-3 mb-2" placeholder="Company Location" required />
</div>

<h3 class="text-2xl mb-5">Company Info</h3>

<div class="mb-4">
  <label for="company" class="block text-gray-700 font-bold mb-2">
    >Company Name</label>
  >
  <input type="text" id="company" v-model="form.company.name" name="company" class="border rounded w-full py-2 px-3" placeholder="Company Name" />
</div>

<div class="mb-4">
  <label for="company_description" class="block text-gray-700 font-bold mb-2">
    >Company Description</label>
  >
  <textarea id="company_description" v-model="form.company.description" name="company_description" class="border rounded w-full py-2 px-3" rows="4" placeholder="What does your company do?"></textarea>
</div>

<div class="mb-4">
  <label for="contact_email" class="block text-gray-700 font-bold mb-2">
    >Contact Email</label>
</div>

```

```
>
<input
  type="email"
  id="contact_email"
  v-model="form.company.contactEmail"
  name="contact_email"
  class="border rounded w-full py-2 px-3"
  placeholder="Email address for applicants"
  required
/>
</div>
<div class="mb-4">
  <label
    for="contact_phone"
    class="block text-gray-700 font-bold mb-2"
    >Contact Phone</label>
  >
  <input
    type="tel"
    id="contact_phone"
    v-model="form.company.contactPhone"
    name="contact_phone"
    class="border rounded w-full py-2 px-3"
    placeholder="Optional phone for applicants"
  />
</div>
<div>
  <button
    class="bg-green-500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline"
    type="submit"
  >
    Add Job
  </button>
</div>
</form>
</div>
</div>
</section>
</template>
```

@submit.prevent

일반적으로 양식을 제출할 때 양식 제출의 기본 동작을 방지하고 싶습니다. 이를 추가하는 대신 양식에 지시문을 `event.preventDefault()`로 추가할 수 있습니다. 이렇게 하면 양식 제출의 기본 동작이 방지됩니다.

`event.preventDefault()` 이벤트 핸들러에서 를 삭제 `submit`하고 `form` 태그에 다음을 추가합니다.

```
<form @submit.prevent="handleSubmit"></form>
```

양식 제출 처리

이제 폼 제출을 처리해야 합니다. Axios를 사용하여 데이터를 서버로 보냅니다. 또한 폼이 제출된 후 사용자를 구인 공고 페이지로 리디렉션합니다.

스크립트에 다음을 추가합니다.

```
const handleSubmit = async () => {
  const newJob = {
    title: form.title,
    type: form.type,
    location: form.location,
    description: form.description,
    salary: form.salary,
    company: {
      name: form.company.name,
      description: form.company.description,
      contactEmail: form.company.contactEmail,
    }
  }
  // ...
}
```

```

    contactPhone: form.company.contactPhone,
  },
};

try {
  const response = await axios.post('/api/jobs', newJob);
  toast.success('Job Added Successfully');
  router.push(`./jobs/${response.data.id}`);
} catch (error) {
  console.error('Error adding job:', error);
  toast.error('Failed to add job');
}
};

```

Vue 토스트화

우리는 .라는 토스트 라이브러리를 사용하고 있습니다 vue-toastification. 설치해 보겠습니다.

```
npm install vue-toastification@next
```

Toast와 CSS에 대한 가져오기를 추가하고 이를 App.vue 파일에서 사용해야 합니다.

```

import { createApp } from 'vue';
import App from './App.vue';
import Toast from 'vue-toastification'; // Add this line
import 'vue-toastification/dist/index.css'; // Add this line
import '@/assets/main.css';
import 'primeicons/primeicons.css';
import router from './router';

const app = createApp(App);

app.use(router);
app.use(Toast); // Add this line

app.mount('#app');

```

파일로 돌아가서 AddJobView.vue, 를 axios 가져옵니다 .useToastuseRouter

```

import axios from 'axios';
import { useRouter } from 'vue-router';
import { useToast } from 'vue-toastification';

```

그리고 다음 form 변수로 초기화합니다.

```

const router = useRouter();
const toast = useToast();

```

양식을 제출하면 토스트 메시지가 나타나고 구인 공고 페이지로 이동하게 됩니다.

작업 삭제

이제 우리는 작업을 삭제할 수 있기를 원합니다. 우리는 이미 버튼을 가지고 있습니다. 서버에 삭제 요청을 보내서 작동하도록 만들어야 합니다.

파일의 삭제 버튼에 클릭 이벤트를 추가해 보겠습니다 JobView.vue.

```

<button @click="deleteJob"
  class="bg-red-500 hover:bg-red-600 text-white font-bold py-2 px-4 rounded-full w-full focus:outline-none focus:shadow-outline mt-4 block">

```

```
    Delete Job
  </button>
```

수입 useRouter 및 useToast:

```
import { useRouter, RouterLink, useRoute } from 'vue-router';
import { useToast } from 'vue-toastification';
```

초기화합니다:

```
const router = useRouter();
const toast = useToast();
```

기능을 추가합니다 deleteJob:

```
const deleteJob = async () => {
  try {
    await axios.delete(`api/jobs/${jobId}`);
    toast.success('Job Deleted Successfully');
    router.push('/jobs');
  } catch (error) {
    console.error('Error deleting job:', error);
    toast.error('Failed to delete job');
  }
};
```

이제 작업을 삭제할 수 있습니다.

작업 편집

마지막 주요 기능은 작업 편집입니다. 에서 새 뷰를 만들어 보겠습니다 src/views/EditJobView.vue. 다음을 추가합니다.

```
<template>Edit page</template>
```

이제 파일에 뷰를 추가하세요 router/index.js.

```
import EditJobView from '@/views/EditJobView.vue';

// ..

{
  path: '/jobs/edit/:id',
  name: 'edit-job',
  component: EditJobView,
},
```

어떤 채용 페이지로 가서 편집 버튼을 클릭하면, 해당 페이지/뷰로 이동하게 됩니다.

양식 추가

페이지에서 코드를 복사 AddJobView에서 페이지에 붙여넣기 만 하면 됩니다 EditJobView. 제목을 "Edit Job"으로 변경하고 버튼을 "Update Job"으로 변경합니다.

작업 데이터 가져오기

우리는 양식을 채우기 위해 데이터를 가져와야 합니다. 우리는 경로 매개변수에서 ID를 가져와서 작업 데이터를 가져오는 데 사용할 수 있습니다.

스크립트에 다음을 추가합니다.

```
import { reactive, onMounted } from 'vue';
import { useRouter, useRoute } from 'vue-router';

const route = useRoute();
const jobId = route.params.id;

const state = reactive({
  job: {},
  isLoading: true,
});

onMounted(() => {
  try {
    const response = await axios.get(`api/jobs/${jobId}`);
    state.job = response.data;
    // Populate form fields after fetching job data
    form.type = state.job.type;
    form.title = state.job.title;
    form.description = state.job.description;
    form.salary = state.job.salary;
    form.location = state.job.location;
    form.company.name = state.job.company.name;
    form.company.description = state.job.company.description;
    form.company.contactEmail = state.job.company.contactEmail;
    form.company.contactPhone = state.job.company.contactPhone;
  } catch (error) {
    console.error('Error fetching job:', error);
  } finally {
    state.isLoading = false;
  }
});
```

onMounted 데이터를 가져오고 작업 상태를 채우는 것을 만듭니다.

```
onMounted(async () => {
  try {
    const response = await axios.get(`api/jobs/${jobId}`);
    state.job = response.data;
    // Populate form fields after fetching job data
    form.type = state.job.type;
    form.title = state.job.title;
    form.description = state.job.description;
    form.salary = state.job.salary;
    form.location = state.job.location;
    form.company.name = state.job.company.name;
    form.company.description = state.job.company.description;
    form.company.contactEmail = state.job.company.contactEmail;
    form.company.contactPhone = state.job.company.contactPhone;
  } catch (error) {
    console.error('Error fetching job:', error);
  } finally {
    state.isLoading = false;
  }
});
```

마지막으로, handleSubmit 작업을 업데이트하려면 다음을 업데이트합니다.

```
const handleSubmit = async () => {
  const updatedJob = {
    title: form.title,
    type: form.type,
    location: form.location,
    description: form.description,
    salary: form.salary,
    company: {
      name: form.company.name,
      description: form.company.description,
      contactEmail: form.company.contactEmail,
      contactPhone: form.company.contactPhone,
    },
  };
  try {
    await axios.put(`api/jobs/${jobId}`, updatedJob);
    toast.success('Job Updated Successfully');
    router.push(`jobs/${jobId}`);
  } catch (error) {
    console.error('Error updating job:', error);
    toast.error('Failed to update job');
  }
};
```

이제 원가를 바꾸고 구인공고를 업데이트해 보세요.

다 됐어요, 친구들. Vue 3, 구성 API, JSON 서버를 갖춘 완전한 CRUD 애플리케이션이 있습니다. 여러분이 이것으로부터 많은 것을 배우고 자신의 Vue.js 프로젝트를 시작할 수 있기를 바랍니다.

최신 소식과 업데이트를 받아보세요!

저희 팀의 최신 뉴스와 업데이트를 받으려면 저희 메일링 목록에 가입하세요.

걱정하지 마세요. 귀하의 정보는 공유되지 않습니다.

이메일

구독하다

우리는 스팸을 싫어합니다. 우리는 어떤 이유로든 귀하의 정보를 판매하지 않습니다.

🔍

카테고리

[모든 카테고리](#) [부트스트랩](#) [작업 코스](#) 및 [공지사항](#) [데이터 베이스](#) [HTML](#) 및 [CSS](#) [html 파일](#) [자바스크립트](#) [자바스크립트 빌드 도구](#) [다음.js](#) [노드.js](#) [오픈 ai](#) [생산력](#) [파이썬](#) [반응하다](#) [뷰](#)

우리를 따르세요

f X @ YouTube

© 2024 Traversy Media. 모든 권리 보유

X @ YouTube in



무료 체험판에 참여하세요

이 일생일대의 기회가 끝나기 전에 오늘 시작하세요.