

원쌤의 Vue.js 퀵스타트

12. vue-router와 axios를 사용한 예제



❖ 작성 단계

- 1단계 : vue-router만 적용
- 2단계 : axios로 백엔드 API 호출하도록 변경
- 3단계 : 비동기 호출로 인한 지연시간 동안 스피너 UI를 보여주도록 변경

1. 애플리케이션 아키텍처와 프로젝트 생성

❖ 작성할 화면들

/home, /about 요청시

TodoList App Home About TodoList

Home

/todos 요청시

TodoList App Home About TodoList

할일 추가

ES6학습	삭제	편집
React학습	삭제	편집
ContextAPI 학습 (완료)	삭제	편집
야구경기 관람	삭제	편집

/todos/add 요청시

TodoList App Home About TodoList

할일 추가

할일 :

설명 :

추가 취소

/todos/edit/:id 요청시

TodoList App Home About TodoList

할일 수정

할일:

설명:

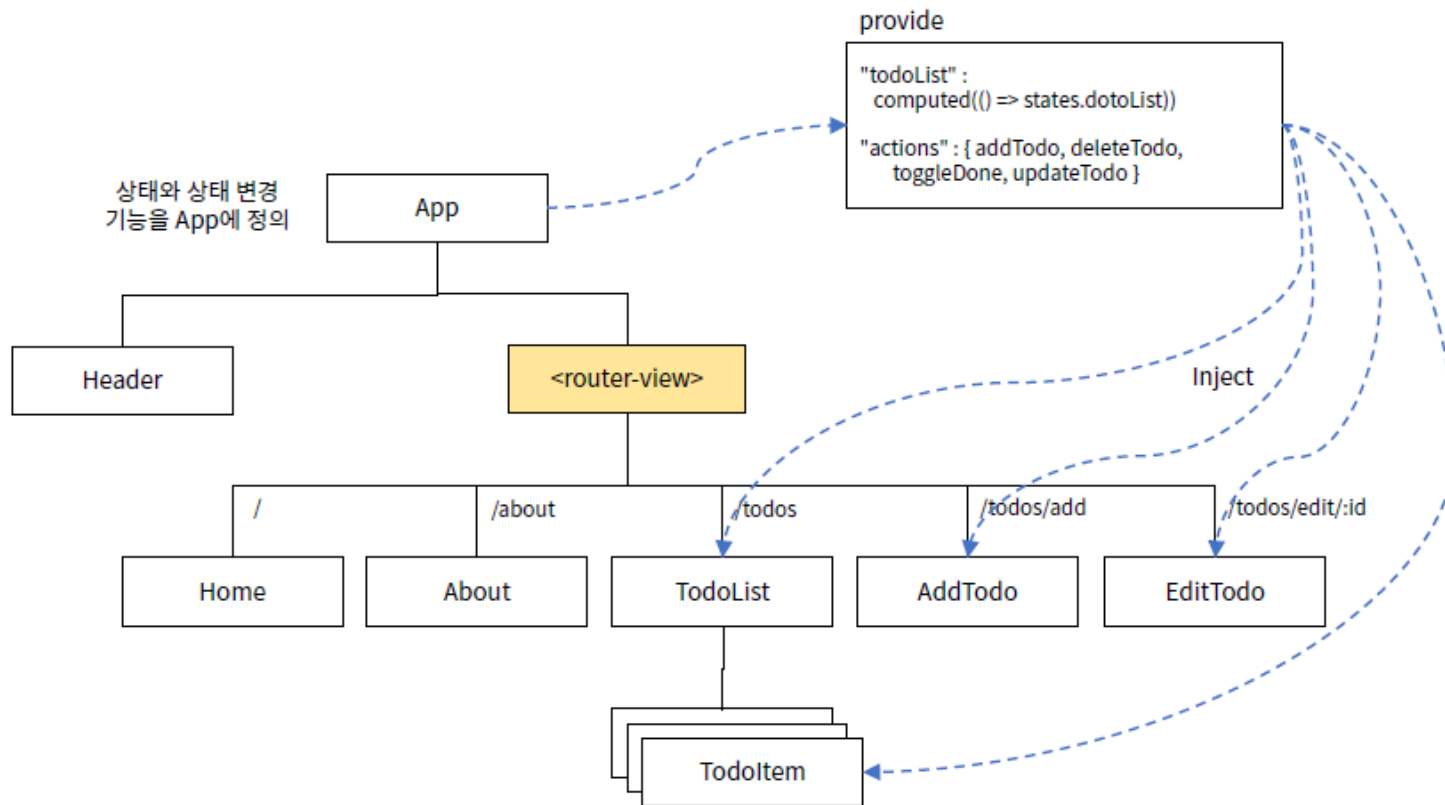
완료여부 : ☐

수정 취소

1. 애플리케이션 아키텍처와 프로젝트 생성

❖ 컴포넌트 계층 구조

- App 컴포넌트에서 상태와 상태 변경 기능 정의하고 provide/inject를 통해 자식컴포넌트에서 이용
 - 반복적으로 속성 전달하지 않아도 됨
- 향후에는 pinia 상태 관리 라이브러리를 사용하도록 변경(13장)



❖ 상태와 상태 변경 기능 정의

[상태 데이터]

```
{
  todoList : [
    { id: 1, todo: "ES6학습", desc: "설명1", done: false },
    .....
  ]
}
```

[상태 변경 기능]

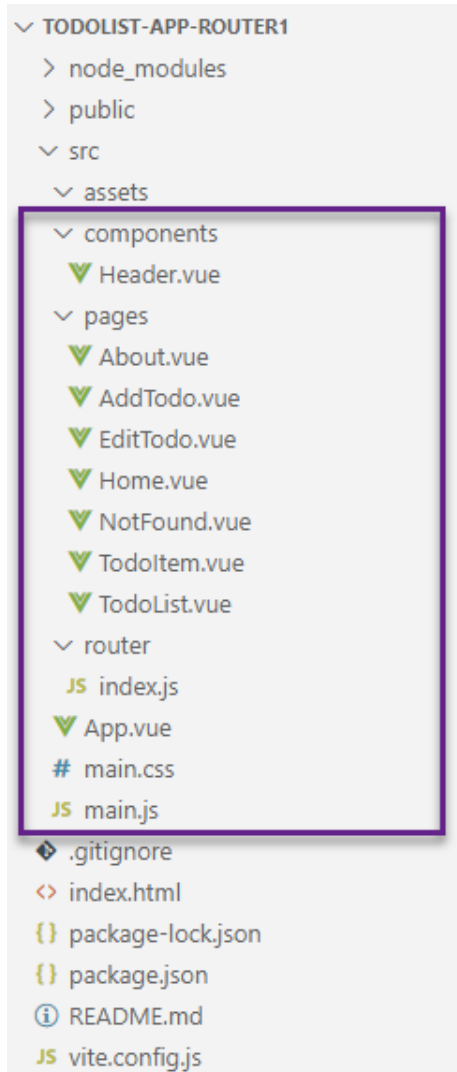
```
addTodo : ({ todo, desc }) => { }
updateTodo : ({ id, todo, desc, done }) => { }
deleteTodo : (id) => { }
toggleDone : (id) => { }
```

1. 애플리케이션 아키텍처와 프로젝트 생성

❖ 프로젝트 생성

```
npm init vue todolist-app-router  
cd todolist-app-router  
npm install  
npm install vue-router@4 bootstrap@5
```

■ 디렉토리 구조 -->



2. 1단계 예제 작성

❖ 예제 12-01 : src/router/index.js

```
1  import { createRouter, createWebHistory } from 'vue-router'
2  import Home from '@pages/Home.vue';
3  import About from '@pages/About.vue';
4  import TodoList from '@pages/TodoList.vue';
5  import AddTodo from '@pages/AddTodo.vue';
6  import EditTodo from '@pages/EditTodo.vue';
7  import NotFound from '@pages/NotFound.vue';
8
9  const router = createRouter({
10     history: createWebHistory(),
11     routes: [
12         { path: '/', component: Home },
13         { path: '/about', component: About },
14         { path: '/todos', component: TodoList },
15         { path: '/todos/add', component: AddTodo },
16         { path: '/todos/edit/:id', component: EditTodo },
17         { path: '/*', component: NotFound },
18     ]
19 })
20
21 export default router;
```

2. 1단계 예제 작성

❖ 예제 12-02~03 : src/main.js 변경, src/main.css 작성

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import 'bootstrap/dist/css/bootstrap.css'
4 import router from './router/index.js'
5 import './main.css';
6
7 const app = createApp(App)
8 app.use(router);
9 app.mount('#app')
```

```
1 body { margin: 0; padding: 0; font-family: sans-serif; }
2 .title { text-align: center; font-weight: bold; font-size: 20pt; }
3 .todo-done { text-decoration: line-through; }
4 .container { padding: 10px 10px 10px 10px; }
5 .panel-borderless { border: 0; box-shadow: none; }
6 .pointer { cursor: pointer; }
```


2. 1단계 예제 작성

❖예제 12-04 : src/App.vue

```
1 <template>
2   <div class="container">
3     <Header />
4     <router-view />
5   </div>
6 </template>
7
8 <script setup>
9 import { reactive, computed, provide } from 'vue'
10 import Header from '@components/Header.vue'
11
12 const states = reactive({
13   todoList : [
14     { id: 1, todo: "ES6학습", desc: "설명1", done: false },
15     { id: 2, todo: "React학습", desc: "설명2", done: false },
16     { id: 3, todo: "ContextAPI 학습", desc: "설명3", done: true },
17     { id: 4, todo: "야구경기 관람", desc: "설명4", done: false },
18   ]
19 })
20
```

```
21 const addTodo = ({ todo, desc }) => {
22   states.todoList.push({ id: new Date().getTime(), todo, desc, done: false })
23 };
24
25 const updateTodo = ({ id, todo, desc, done }) => {
26   let index = states.todoList.findIndex((todo) => todo.id === id);
27   states.todoList[index] = { ...states.todoList[index], todo, desc, done };
28 };
29
30 const deleteTodo = (id) => {
31   let index = states.todoList.findIndex((todo) => todo.id === id);
32   states.todoList.splice(index, 1);
33 }
34
35 const toggleDone = (id) => {
36   let index = states.todoList.findIndex((todo) => todo.id === id);
37   states.todoList[index].done = !states.todoList[index].done;
38 }
39
40 provide('todoList', computed(() => states.todoList))
41 provide('actions', { addTodo, deleteTodo, toggleDone, updateTodo })
42 </script>
```

2. 1단계 예제 작성

❖ 예제 12-05 : src/components/Header.vue

```
1 <template>
2   <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
3     <span class="navbar-brand ps-2">ToDoList App</span>
4     <button class="navbar-toggler" type="button" @click="isNavShow = !isNavShow">
5       <span class="navbar-toggler-icon"></span>
6     </button>
7     <div :class="isNavShow ? 'collapse navbar-collapse show' : 'collapse navbar-collapse'">
8       <ul class="navbar-nav">
9         <li class="nav-item">
10           <router-link class="nav-link" to="/">
11             Home
12           </router-link>
13         </li>
14         <li class="nav-item">
15           <router-link class="nav-link" to="/about">
16             About
17           </router-link>
18         </li>
```

```
19       <li class="nav-item">
20         <router-link class="nav-link" to="/todos">
21           ToDoList
22         </router-link>
23       </li>
24     </ul>
25   </div>
26 </nav>
27 </template>
28
29 <script setup>
30 import { ref } from 'vue';
31
32 const isNavShow = ref(false);
33 </script>
```

❖ 예제 12-06, 07

예제 12-06 : src/pages/Home.vue, src/pages/About.vue 작성

```
01: <template>
02:   <div class="card card-body">
03:     <h2>Home</h2>
04:   </div>
05: </template>
```

예제 12-07 : src/pages/NotFound.vue 작성

```
01: <template>
02:   <div class="m-3">
03:     <h3>존재하지 않는 경로</h3>
04:     <p>요청 경로 : {{currentRoute.path}}</p>
05:   </div>
06: </template>
07:
08: <script setup>
09: import { useRoute } from 'vue-router'
10: const currentRoute = useRoute();
11: </script>
```

2. 1단계 예제 작성

❖ 예제 12-08 : src/pages/ToDoList.vue

```
1  <template>
2    <div class="row">
3      <div class="col p-3">
4        <router-link class="btn btn-primary" to="/todos/add">
5          할일 추가
6        </router-link>
7      </div>
8    </div>
9    <div class="row">
10     <div class="col">
11       <ul class="list-group">
12         <ToDoItem v-for="todoItem in todoList" :key="todoItem.id" :todoItem="todoItem" />
13       </ul>
14     </div>
15   </div>
16 </template>
17
18 <script setup>
19 import {inject} from 'vue';
20 import ToDoItem from '@pages/ToDoItem.vue'
21
22 const todoList = inject('todoList');
23 </script>
```

2. 1단계 예제 작성

❖ 예제 12-09 : src/pages/TodoItem.vue

```
1 <template>
2   <li :class="todoItem.done ? 'list-group-item list-group-item-success' : 'list-group-item'">
3     <span :class="todoItem.done ? 'todo-done pointer' : 'pointer'"
4       @click="toggleDone(todoItem.id)">
5       {{todoItem.todo}}
6       {{todoItem.done ? '(완료)' : '' }}
7     </span>
8     <span class="float-end badge bg-secondary pointer m-1"
9       @click="router.push(`/todos/edit/${todoItem.id}`)">
10      편집</span>
11     <span class="float-end badge bg-secondary pointer m-1"
12       @click="deleteTodo(todoItem.id)">
13      삭제</span>
14   </li>
15 </template>

16
17 <script setup>
18 import { useRouter } from 'vue-router';
19 import { inject } from 'vue';
20
21 defineProps({
22   todoItem: { Type: Object, required:true }
23 })
24
25 const router = useRouter();
26 const { deleteTodo, toggleDone } = inject('actions');
27 </script>
```

2. 1단계 예제 작성

❖예제 12-10 : src/pages/AddTodo.vue

```
1 <template>
2   <div class="row">
3     <div class="col p-3">
4       <h2>할일 추가</h2>
5     </div>
6   </div>
7   <div class="row">
8     <div class="col">
9       <div class="form-group">
10        <label htmlFor="todo">할일 :</label>
11        <input type="text" class="form-control" id="todo" v-model="todoItem.todo" />
12      </div>
13      <div class="form-group">
14        <label htmlFor="desc">설명 :</label>
15        <textarea class="form-control" rows="3" id="desc" v-model="todoItem.desc"></textarea>
16      </div>
17      <div class="form-group">
18        <button type="button" class="btn btn-primary m-1" @click="addTodoHandler">
19          추 가
20        </button>
21        <button type="button" class="btn btn-primary m-1" @click="router.push('/todos')">
22          취 소
23        </button>
24      </div>
25    </div>
26  </div>
27 </template>
```

```
28
29 <script setup>
30 import { inject, reactive } from 'vue';
31 import { useRouter } from 'vue-router';
32
33 const router = useRouter();
34 const { addTodo } = inject('actions');
35 const todoItem = reactive({ todo:"", desc:"" })
36
37 const addTodoHandler = () => {
38   let { todo } = todoItem;
39   if (!todo || todo.trim()=== "") {
40     alert('할일은 반드시 입력해야 합니다');
41     return;
42   }
43   addTodo({ ...todoItem });
44   router.push('/todos')
45 }
46 </script>
```


❖ EditTodo 컴포넌트 작성

- 현재 라우트(currentRoute) 정보를 이용해 /todos/edit/{id}에 해당하는 동적 파라미터 params.id 값을 받아냅니다.
- todoList 전체에서 params.id와 일치하는 id를 가진 할 일 한 건(matchedTodoItem)을 찾아냅니다.
- todoItem 한 건으로 reactive()를 이용해 상태 데이터로 설정합니다. 이것은 v-model 디렉티브로 양방향 데이터 바인딩하여 사용자가 직접 변경할 수 있도록 합니다.

todoList 데이터 중에서 id가 일치하는 것을 찾아 그 데이터를 화면에 보여주고 수정할 수 있도록 해야 하므로 47행과 같이 reactive() 함수를 이용해 상태 데이터로 지정합니다. 이때 기존 todoItem이 직접 변경되지 않도록 전개 연산자(Spread Operator)를 사용해 새로운 객체를 만들어야 한다는 점에 주의하세요. 만일 새로운 객체를 상태 데이터로 설정하지 않으면 편집 화면에서 값을 변경한 뒤 취소 버튼을 클릭해도 todoList 상태 데이터가 바뀔 것입니다.

❖예제 12-11 : src/pages/EditTodo.vue

```

1 <template>
2   <div class="row">
3     <div class="col p-3">
4       <h2>할일 수정</h2>
5     </div>
6   </div>
7   <div class="row">
8     <div class="col">
9       <div class="form-group">
10        <label htmlFor="todo">할일:</label>
11        <input type="text" class="form-control" id="todo" v-model="todoItem.todo" />
12      </div>
13      <div class="form-group">
14        <label htmlFor="desc">설명:</label>
15        <textarea class="form-control" rows="3" id="desc" v-model="todoItem.desc"></textarea>
16      </div>
17      <div class="form-group">
18        <label htmlFor="done">완료여부 : </label>&nbsp;  
19        <input type="checkbox" v-model="todoItem.done" />
20      </div>
21      <div class="form-group">
22        <button type="button" class="btn btn-primary m-1" @click="updateTodoHandler">
23          수정
24        </button>
25        <button type="button" class="btn btn-primary m-1" @click="router.push('/todos')">
26          취소
27        </button>
28      </div>
29    </div>
30  </div>
31 </template>

```


❖예제 12-11 : src/pages/EditTodo.vue (이어서)

```
32
33 <script setup>
34 import { inject, reactive } from 'vue';
35 import { useRouter, useRoute } from 'vue-router';
36
37 const todoList = inject('todoList');
38 const { updateTodo } = inject('actions');
39 const router = useRouter();
40 const currentRoute = useRoute();
41
42 const matchedTodoItem = todoList.value.find((item) => item.id === parseInt(currentRoute.params.id))
43 if (!matchedTodoItem) {
44   router.push('/todos');
45 }
46 const todoItem = reactive({ ...matchedTodoItem })
47
48 const updateTodoHandler = () => {
49   let { todo } = todoItem;
50   if (!todo || todo.trim() === "") {
51     alert('할일은 반드시 입력해야 합니다');
52     return;
53   }
54   updateTodo({ ...todoItem });
55   router.push('/todos');
56 }
57 </script>
```

2. 1단계 예제 작성

❖ 1단계 실행 결과

The screenshot displays a web browser at `localhost:5173/todos` showing a 'TodoList App' interface. The app has a dark header with 'Home', 'About', and 'TodoList' links. Below the header is a blue button labeled '할일 추가'. The main content area lists four items: 'ES6학습 (완료)', 'React학습', 'ContextAPI 학습 (완료)', and '야구경기 관람'. Each item has '삭제' and '편집' buttons. The 'ContextAPI 학습 (완료)' item is highlighted in green.

Below the browser window, the Vue DevTools interface is shown. The 'Elements' panel displays the component tree:

```
<App>
  <Header>
    <RouterLink> /
    <RouterLink> /about
    <RouterLink> /todos
  <RouterView> 12.7 ms /todos
    <TodoList> fragment 11.9 ms
```

The 'Components' panel shows the props for the selected `<TodoList>` component:

```
<TodoList>
  1: Object
  2: Object
  3: Object
  provided
    Symbol(router view depth): undefined (Computed)
    Symbol(router view location matched): Object (Computed)
    Symbol(router view location): Object (Computed)
  setup (other)
    TodoItem: TodoItem
  Routing
    $route: /todos
```

3. 2단계 axios 적용

❖ 백엔드 API 실행과 프록시 설정

- 11장에서 학습할 때 사용했던 todosvc
 - <https://github.com/stepanowon/todosvc>
- 다음 명령어로 백엔드 API 실행

```
npm install
npm run start:dev
```

❖ vite.config.js 변경

❖ axios 설치

- `npm install --save axios`

```
1  import { fileURLToPath, URL } from 'node:url'
2
3  import { defineConfig } from 'vite'
4  import vue from '@vitejs/plugin-vue'
5
6  // https://vitejs.dev/config/
7  export default defineConfig({
8    plugins: [vue()],
9    resolve: {
10     alias: {
11       '@': fileURLToPath(new URL('./src', import.meta.url))
12     }
13   },
14   server: {
15     proxy: {
16       '/api': {
17         target: 'http://localhost:8000',
18         changeOrigin: true,
19         rewrite: (path) => path.replace(/^\/api/, ''),
20       },
21     },
22   },
23 })
```

3. 2단계 axios 적용

❖ 예제 12-13 : src/App.vue 변경 - axios를 사용해 백엔드 API를 호출하도록 변경

```
1 <template>
2   <div class="container">
3     <Header />
4     <router-view />
5   </div>
6 </template>
7
8 <script setup>
9 import { reactive, provide, computed } from 'vue'
10 import Header from '@components/Header.vue'
11 import axios from 'axios';
12
13 const owner = "gdhong";
14 //의도적 지연 시간을 발생시키는 /todolist_long 이용
15 const BASEURI = "/api/todolist_long";
16 const states = reactive({ todoList:[] })
17 //TodoList 목록을 조회합니다.
18 const fetchTodoList = async () => {
19   try {
20     const response = await axios.get(BASEURI + `/${owner}`);
21     if (response.status === 200) {
22       states.todoList = response.data;
23     } else {
24       alert('데이터 조회 실패');
25     }
26   } catch(error) {
27     alert('에러발생 : ' + error);
28   }
29 }
```

```
30 // 새로운 TodoItem을 추가합니다.
31 const addTodo = async ({ todo, desc }, successCallback) => {
32   try {
33     const payload = { todo, desc };
34     const response = await axios.post(BASEURI + `/${owner}`, payload)
35     if (response.data.status === "success") {
36       states.todoList.push({ id: response.data.item.id, todo, desc, done: false })
37       successCallback();
38     } else {
39       alert('Todo 추가 실패 : ' + response.data.message);
40     }
41   } catch(error) {
42     alert('에러발생 : ' + error);
43   }
44 }
45 // 기존 TodoItem을 변경합니다.
46 const updateTodo = async ({ id, todo, desc, done }, successCallback) => {
47   try {
48     const payload = { todo, desc, done };
49     const response = await axios.put(BASEURI + `/${owner}/${id}`, payload)
50     if (response.data.status === "success") {
51       let index = states.todoList.findIndex((todo) => todo.id === id);
52       states.todoList[index] = { id, todo, desc, done };
53       successCallback();
54     } else {
55       alert('Todo 변경 실패 : ' + response.data.message);
56     }
57   } catch(error) {
58     alert('에러발생 : ' + error);
59   }
60 }
```

3. 2단계 axios 적용

```
61 //기존 TodoItem을 삭제합니다.
62 const deleteTodo = async (id) => {
63   try {
64     const response = await axios.delete(BASEURI + `/${owner}/${id}`)
65     if (response.data.status === "success") {
66       let index = states.todoList.findIndex((todo) => todo.id === id);
67       states.todoList.splice(index, 1);
68     } else {
69       alert('Todo 삭제 실패 : ' + response.data.message);
70     }
71   } catch(error) {
72     alert('에러발생 : ' + error);
73   }
74 }
75 //기존 TodoItem의 완료여부(done) 값을 토글합니다.
76 const toggleDone = async (id) => {
77   try {
78     const response = await axios.put(BASEURI + `/${owner}/${id}/done`)
79     if (response.data.status === "success") {
80       let index = states.todoList.findIndex((todo) => todo.id === id);
81       states.todoList[index].done = !states.todoList[index].done;
82     } else {
83       alert('Todo 완료 변경 실패 : ' + response.data.message);
84     }
85   } catch(error) {
86     alert('에러발생 : ' + error);
87   }
88 }
```

```
89
90 provide('todoList', computed(() => states.todoList));
91 provide('actions', { addTodo, deleteTodo, toggleDone, updateTodo, fetchTodoList })
92
93 fetchTodoList();
94 </script>
```

이 예제에서 addTodo, updateTodo 함수를 살펴보면 기존과 달리 successCallback이라는 인자를 추가하고 있음을 알 수 있습니다. 백엔드 API의 호출은 네트워크 상태, 서버의 상태에 따라 약간의 지연이 발생할 수 있는데, 이 예제에서는 15행에서 1초의 의도적인 지연 시간 후에 응답하는 /todolist_long API를 사용합니다. 이 경우 추가, 수정하는 화면에서 다시 할 일을 조회하는 화면으로의 전환은 추가, 수정이 완료된 이후여야 합니다. successCallback은 바로 비동기 처리가 완료된 후 수행할 작업을 전달하기 위해 사용하는 것입니다. 예를 들어 AddTodo 컴포넌트에서 새로운 TodoItem을 추가할 때 백엔드에서의 비동기 처리가 완료된 이후에 successCallback 함수를 호출하여 화면을 전환합니다.

3. 2단계 axios 적용

❖ 예제 12-14,15 : src/pages/AddTodo.vue, EditTodo.vue 변경

```
1 > <template> ...
27 </template>
28
29 <script setup>
30 import { inject, reactive } from 'vue';
31 import { useRouter } from 'vue-router';
32
33 const router = useRouter();
34 const { addTodo } = inject('actions');
35 const todoItem = reactive({ todo:"", desc:"" })
36
37 const addTodoHandler = () => {
38   let { todo } = todoItem;
39   if (!todo || todo.trim() === "") {
40     alert('할일은 반드시 입력해야 합니다');
41     return;
42   }
43   addTodo({ ...todoItem }, ()=>{
44     router.push('/todos')
45   });
46 }
47 </script>
```

```
1 > <template> ...
31 </template>
32
33 <script setup>
34 import { inject, reactive } from 'vue';
35 import { useRouter, useRoute } from 'vue-router';
36
37 const todoList = inject('todoList');
38 const { updateTodo } = inject('actions');
39 const router = useRouter();
40 const currentRoute = useRoute();
41
42 const matchedTodoItem = todoList.value.find((item)=> item.id === parseInt(currentRoute.params.id))
43 if (!matchedTodoItem) {
44   router.push('/todos');
45 }
46 const todoItem = reactive({ ...matchedTodoItem })
47
48 const updateTodoHandler = () => {
49   let { todo } = todoItem;
50   if (!todo || todo.trim() === "") {
51     alert('할일은 반드시 입력해야 합니다');
52     return;
53   }
54   updateTodo({ ...todoItem }, ()=>{
55     router.push('/todos');
56   });
57 }
58 </script>
```

3. 2단계 axios 적용

❖예제 12-16 : src/pages/ToDoList.vue 변경

```
1 <template>
2   <div class="row">
3     <div class="col p-3">
4       <router-link class="btn btn-primary" to="/todos/add">
5         할일 추가
6       </router-link>
7       <button class="btn btn-primary ms-1" @click="fetchToDoList">
8         새로 고침
9       </button>
10    </div>
11  </div>
12  <div class="row">
13    <div class="col p-3">
14      <ul class="list-group">
15        <ToDoItem v-for="todoItem in todoList" :key="todoItem.id" :todoItem="todoItem" />
16      </ul>
17    </div>
18  </div>
19 </template>
20
21 <script setup>
22 import {inject} from 'vue';
23 import ToDoItem from '@/pages/ToDoItem.vue'
24
25 const todoList = inject('todoList');
26 const { fetchToDoList } = inject('actions');
27 </script>
```

3. 2단계 axios 적용

❖ 2단계 실행 결과

Vite App

localhost:5173/todos

Bookmark Downloads 여행 공부 머신러닝 DB&BigData Development Utility 보안 감리 E-book 맥북 기타 OAuth MongoDB hadoop Oracle Spring 낚시 기타 북마크

TodoList App Home About TodoList

할일 추가 새로 고침

ES6 공부	삭제 편집
Vue 학습	삭제 편집
놀기 (완료)	삭제 편집
야구장 (완료)	삭제 편집
햄스터 먹이주기	삭제 편집

Find components...

<Header>

- <RouterLink> /
- <RouterLink> /about
- <RouterLink> /todos

<RouterView> /todos

<TodoList> fragment

- <RouterLink> /todos/add
- <TodoItem key=123456789>
- <TodoItem key=1671940103853>
- <TodoItem key=1671940103854>
- <TodoItem key=1671940103855>
- <TodoItem key=1671944388233>

<TodoList> Filter state...

- setup
- provided
 - Symbol(router view depth): undefined (Computed)
 - Symbol(router view location matched): Object (Computed)
 - Symbol(router view location): Object (Computed)
- setup (other)
 - TodoItem: TodoItem
 - fetchTodoList: function fetchTodoList()
- Routing
 - \$route: /todos

4. 3단계 지연 시간에 대한 스피너 UI 구현

❖ 이전까지의 예제

- 지연 시간동안 화면이 정지한 느낌
- 비동기 처리 동안 시각적으로 작업이 처리중임을 알리는 것이 좋은

❖ 패키지 설치

- `npm install --save vue-cssspin`

❖ 예제 12-17 : src/components/Loading.vue 추가

```
1 <template>
2   |   <VueCssspin message="Loading" spin-style="cp-flip" />
3 </template>
4
5 <script setup>
6 import { VueCssspin } from 'vue-cssspin'
7 import 'vue-cssspin/dist/vue-cssspin.css'
8 </script>
```

4. 3단계 지연 시간에 대한 스피너 UI 구현

❖ 예제 12-18 : src/App.vue 변경

- 상태 추가 : isLoading
 - 비동기 처리 시작 -> isLoading:true
 - 비동기 처리 완료 -> isLoading:false
- isLoading 을 이용해 Loading 컴포넌트를 보여줄 지 여부 결정

```
1 <template>
2   <div class="container">
3     <Header />
4     <router-view />
5     <Loading v-if="states.isLoading" />
6   </div>
7 </template>
8
9 <script setup>
10 import { reactive, provide, computed } from 'vue'
11 import Header from '@/components/Header.vue'
12 import Loading from '@/components/Loading.vue'
13 import axios from 'axios';
14
15 const owner = "gdhong";
16 const BASEURI = "/api/todolist_long";
17
18 const states = reactive({ todoList:[], isLoading:false })
19
```

4. 3단계 지연 시간에 대한 스피너 UI 구현

```
20 //TodoList 목록을 조회합니다.
21 const fetchTodoList = async () => {
22   states.isLoading = true;
23   try {
24     const response = await axios.get(BASEURI + `/${owner}`);
25     if (response.status === 200) {
26       states.todoList = response.data;
27     } else {
28       alert('데이터 조회 실패');
29     }
30   } catch(error) {
31     alert('에러발생 : ' + error);
32   }
33   states.isLoading = false;
34 }
35
36 // 새로운 TodoItem을 추가합니다.
37 const addTodo = async ({ todo, desc }, successCallback) => {
38   states.isLoading = true;
39   try {
40     const payload = { todo, desc };
41     const response = await axios.post(BASEURI + `/${owner}`, payload)
42     if (response.data.status === "success") {
43       states.todoList.push({ id: response.data.item.id, todo, desc, done: false })
44       successCallback();
45     } else {
46       alert('Todo 추가 실패 : ' + response.data.message);
47     }
48   } catch(error) {
49     alert('에러발생 : ' + error);
50   }
51   states.isLoading = false;
52 }
```

```
53 // 기존 TodoItem을 변경합니다.
54 const updateTodo = async ({ id, todo, desc, done }, successCallback) => {
55   states.isLoading = true;
56   try {
57     const payload = { todo, desc, done };
58     const response = await axios.put(BASEURI + `/${owner}/${id}`, payload)
59     if (response.data.status === "success") {
60       let index = states.todoList.findIndex((todo) => todo.id === id);
61       states.todoList[index] = { id, todo, desc, done };
62       successCallback();
63     } else {
64       alert('Todo 변경 실패 : ' + response.data.message);
65     }
66   } catch(error) {
67     alert('에러발생 : ' + error);
68   }
69   states.isLoading = false;
70 }
71 //기존 TodoItem을 삭제합니다.
72 const deleteTodo = async (id) => {
73   states.isLoading = true;
74   try {
75     const response = await axios.delete(BASEURI + `/${owner}/${id}`)
76     if (response.data.status === "success") {
77       let index = states.todoList.findIndex((todo) => todo.id === id);
78       states.todoList.splice(index, 1);
79     } else {
80       alert('Todo 삭제 실패 : ' + response.data.message);
81     }
82   } catch(error) {
83     alert('에러발생 : ' + error);
84   }
85   states.isLoading = false;
86 }
```

4. 3단계 지연 시간에 대한 스피너 UI 구현

```
87 //기존 TodoItem의 완료여부(done) 값을 토글합니다.
88 const toggleDone = async (id) => {
89   states.isLoading = true;
90   try {
91     const response = await axios.put(BASEURI + `/${owner}/${id}/done`)
92     if (response.data.status === "success") {
93       let index = states.todoList.findIndex((todo) => todo.id === id);
94       states.todoList[index].done = !states.todoList[index].done;
95     } else {
96       alert('Todo 완료 변경 실패 : ' + response.data.message);
97     }
98   } catch(error) {
99     alert('에러발생 : ' + error);
100   }
101   states.isLoading = false;
102 }
103
104 provide('todoList', computed(()=>states.todoList));
105 provide('actions', { addTodo, deleteTodo, toggleDone, updateTodo, fetchTodoList })
106
107 fetchTodoList();
108 </script>
```

4. 3단계 지연 시간에 대한 스피너 UI 구현

❖ 3단계 실행 결과

The screenshot shows a web browser window with the address bar displaying 'localhost:5173/todos'. The page title is 'Vite App'. The browser's developer tools are open, showing the component tree. The tree structure is as follows:

- <App> (15.6 ms)
 - <Header>
 - <RouterLink> (/)
 - <RouterLink> (/about)
 - <RouterLink> (/todos)
 - <RouterView> (/todos)
 - <TodoList> (fragment)
 - <Loading key=0>
 - <VueCssspin>

10장과 11장에서 학습한 내용을 바탕으로 TodoList 애플리케이션을 vue-router와 axios를 이용하도록 다시 작성해보았습니다. 하지만 조금 아쉬운 부분이라면 상태관리를 provide/inject를 사용한 것입니다. provide/inject는 주로 읽기 전용의 데이터를 공유하기 위한 용도로 사용합니다. 애플리케이션 수준의 중앙 집중화된 상태 관리 용도로는 적합하지 않습니다. 그렇기 때문에 다음 장에서는 이 예제를 pinia라는 상태 관리 라이브러리를 적용하도록 변경해볼 것입니다.