원쌤의 Vue.js 퀵스타트

13. pinia를 이용한 상태 관리

1. pinia란?

❖pinia

- Composition API 방식으로 Vue 애플리케이션을 위한 중앙집중화된 상태 관리 기능을 제공하도록 설계된 라이브러리
- Vuex --〉 Pinia

What is Vuex?

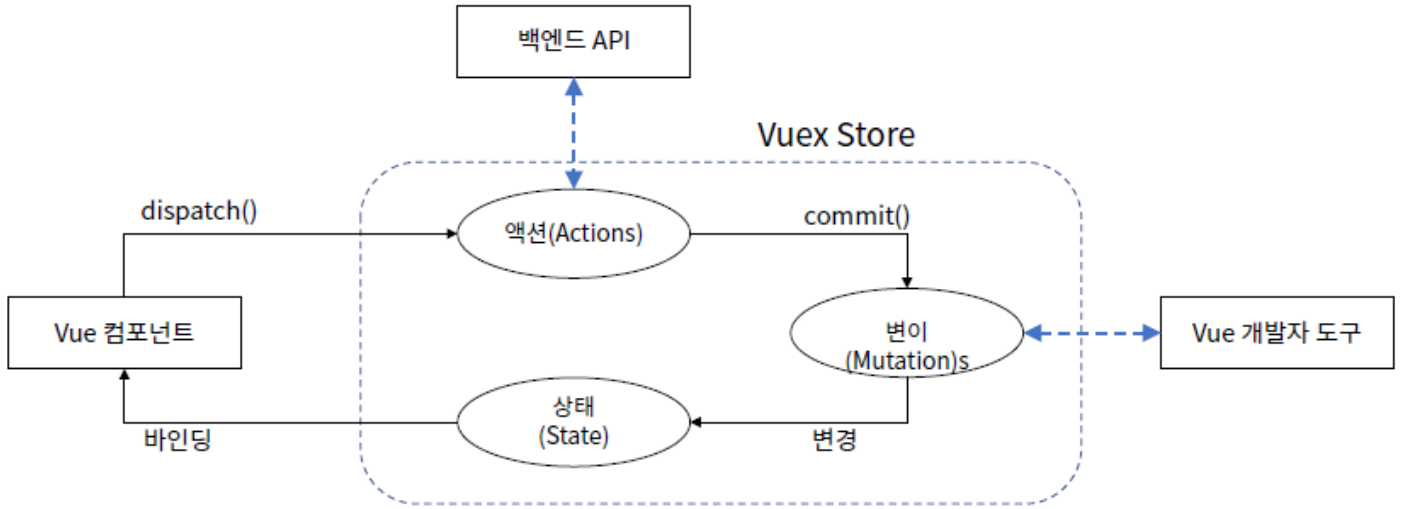
> **Pinia is now the new default**
>
> The official state management library for Vue has changed to Pinia. Pinia has almost the exact same or enhanced API as Vuex 5, described in Vuex 5 RFC. You could simply consider Pinia as Vuex 5 with a different name. Pinia also works with Vue 2.x as well.
>
> Vuex 3 and 4 will still be maintained. However, it's unlikely to add new functionalities to it. Vuex and Pinia can be installed in the same project. If you're migrating existing Vuex app to Pinia, it might be a suitable option. However, if you're planning to start a new project, we highly recommend using Pinia instead.

1. pinia란?

❖ pinia의 장점

- pinia는 vuex보다 더 간단한 구조
  - 스토어 내부에 액션, 상태만을 정의하면 됨
  - 액션 메서드를 직접 호출할 수 있음
  - vuex에서 상태 변경을 위해...

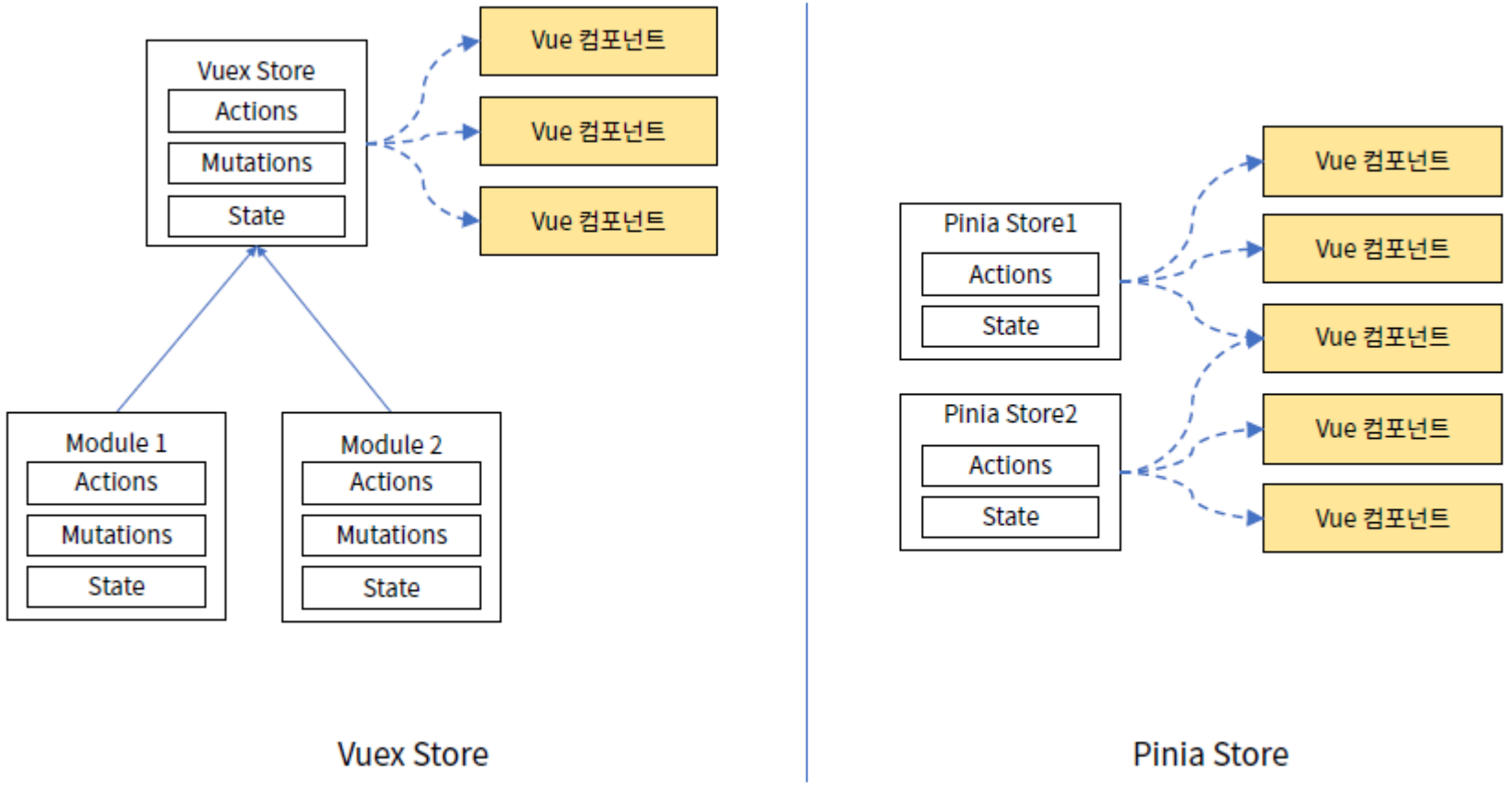
```
this.$store.dispatch("addTodo", { todo: this.todo, done: false })
```



1. pinia란?

❖ pinia의 장점

- pinia는 다중 스토어 지원 – vuex는 단일 스토어



1. pinia란?

❖ pinia의 장점

- Composition API, 옵션 API 모두 지원함 – Vuex는 옵션 API 만을 지원

[Vuex의 옵션 API 방식의 스토어 생성]

```
const store = createStore({
  state: {  count: 0  },
  mutations: {
    increment (state, { num }) {
      state.count += payload.num;
    }
  },
  actions: {
    increment (context, { num }) {
      context.commit('increment', { num : num } )
    }
  }
})
```

1. pinia란?

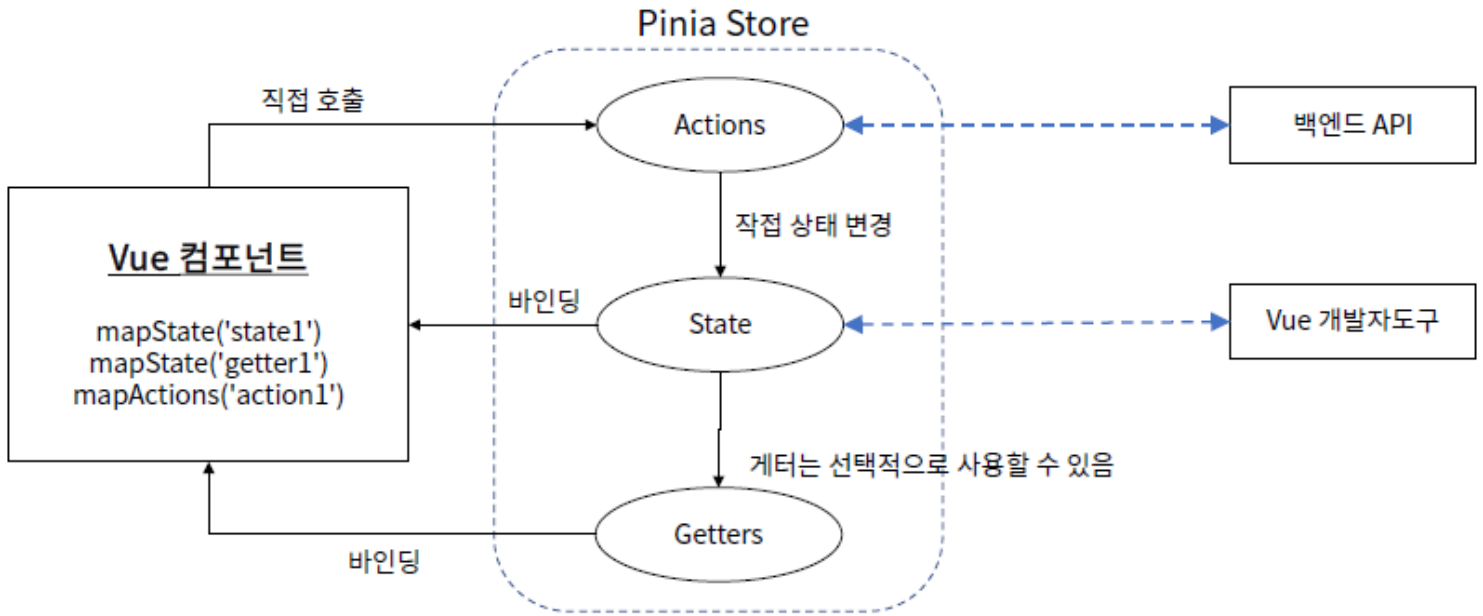
❖ pinia의 장점

- 타입스크립트 지원이 vuex보다 강력함
  - Vuex는 모듈을 이용해 확장
  - Vuex 단일 스토어에서 모듈이 제공하는 분산된 상태, 액션을 모두 이용함.
  - 이와 같은 경우 타입 추론을 어렵게 함.
  - 이 책에서는 타입 스크립트를 사용하지 않지만 최근의 트렌드임
  - 반면 pinia은 다중 스토어를 지원하고, 각 스토어마다 독자적인 상태, 액션을 정의하므로 타입 추론이 쉬움
- pinia는 vuex에 비해 가벼움
  - pinia 라이브러리의 크기 : 〈 6KB
  - vuex의 1/3

2. pinia 아키텍처와 구성 요소

❖ pinia 아키텍처

- 스토어, 액션, 상태, 게터
- 게터는 필수 아님
- Vue 컴포넌트에서 사용할 때는...
  - 옵션 API : mapState, mapActions와 같은 유틸리티 함수 사용
  - Composition API : use~ 로 시작하는 사용자 정의 훅을 사용



2. pinia 아키텍처와 구성 요소

❖ 스토어 정의

- defineStore() 함수 사용

```
import { defineStore } from 'pinia'
import { reactive, computed } from 'vue'

// [ 옵션 API 방법 적용 ]
export const useCount1Store = defineStore('count1', {
  state: () => ({
    count: 0
  }),
  actions: {
    increment({ num }) {
      this.count += num;
    },
  }
})
```

```
// [ 컴포지션 API 방법 적용 ]
export const useCount2Store = defineStore('count2', ()=>{
    const state = reactive({ count : 0 });
    const increment = ({ num }) => {
        state.count +=num;
    }
    const count = computed(()=>state.count);

    return { count, increment };
})
```

만일 Vue 2.x를 지원해야 하거나 pinia를 처음 접한다면 옵션 API로 시작하시는 것이 좋습니다. 옵션 API로 개발하는 것이 더 뛰어나거나 바람직하다기보다는 더 많은 개발자들이 옵션 API를 사용하고 있으며 하위 호환성도 잘 지원하기 때문입니다. 또한 Vue 개발자 도구에서도 옵션 API를 사용할 때만 상태를 초기화하거나 액션을 추적할 수 있는 기능을 제공하고 있기 때문에 디버깅할 때도 더 편리합니다. (2023년 3월 기준)

2. pinia 아키텍처와 구성 요소

❖ pinia를 사용하도록 Vue 애플리케이션 인스턴스 설정

- createPinia() 를 호출하여 pinia 객체 생성
- pinia 객체를 Vue 애플리케이션 인스턴스의 use 메서드의 인자로 전달하여 호출

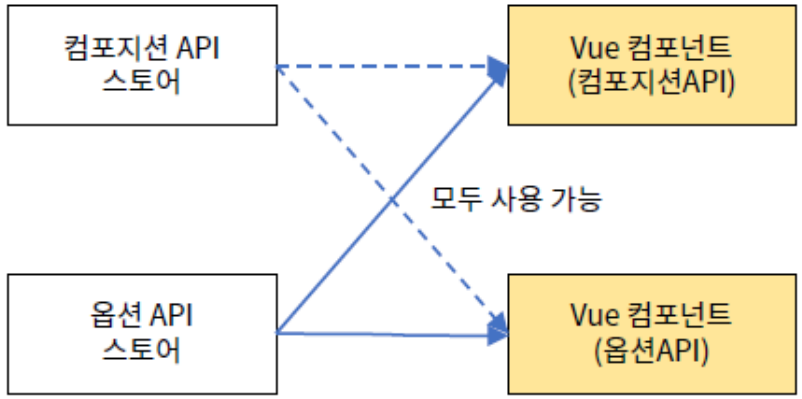
```
import { createApp } from 'vue'
import App from './App.vue'
import './assets/main.css'
import { createPinia } from 'pinia'

const pinia = createPinia()
const app = createApp(App)

app.use(pinia)
app.mount('#app')
```

2. pinia 아키텍처와 구성 요소

❖ 컴포넌트에서 스토어 사용



- 옵션 API 사용 컴포넌트
  - mapState : 스토어의 상태, 게터를 컴포넌트의 계산된 속성에 지정
  - mapActions : 스토어의 액션을 컴포넌트의 메서드에 지정
  - 기존의 메서드, 계산된 속성과 결합해야 할 때는 전개 연산자를 활용함

[옵션 API를 사용한 컴포넌트에서의 스토어 사용]

```
<script>
import { useCount1Store} from '@/store/counter.js'
import { mapState, mapActions } from 'pinia';

export default {
  name:"App",
  computed : {
    ...mapState(useCount1Store, ['count'])
  },
  methods : {
    ...mapActions(useCount1Store, ['increment'])
  }
}
</script>
```

2. pinia 아키텍처와 구성 요소

- Composition API를 사용하는 컴포넌트
  - defineStore() 함수 호출하여 리턴된 훅 형태의 함수를 호출해 스토어 객체를 받아내어 이용함
  - 상태 데이터를 이용할 때 computed() 를 이용해 반응성을 잃지 않도록 해야 함

[컴포지션 API를 사용한 컴포넌트에서의 스토어 사용]

```
<script>
import { useCount1Store} from '@/store/counter.js'
import { computed } from 'vue';

export default {
  setup() {
    const store = useCount1Store();
    const count =  computed(()=>store.count);
    const increment = store.increment;

    return { count, increment };
  }
}
</script>
```

3. 간단한 pinia 예제 작성

❖ 프로젝트 생성, 초기화

```
npm init vue pinia-test-app
cd pinia-test-app
npm install
npm install pinia
```

- src/components, src/assets 디렉터리 내의 파일, src/App.vue 삭제
- 다음 디렉터리, 파일 생성
  - src/App1.vue
  - src/App2.vue
  - src/stores/todoList.js

3. 간단한 pinia 예제 작성

❖ 예제 13-01 : src/stores/todoList.js

```
1  import { defineStore } from "pinia";
2  import { reactive, computed } from 'vue';
3
4  // 옵션 API 방식의 todoList1 스토어
5  export const useTodoList1Store = defineStore("todoList1", {
6     state : () => ({
7        todoList : [
8           { id: 1, todo: "ES6학습",  done: false },
9           { id: 2, todo: "React학습",  done: false },
10          { id: 3, todo: "ContextAPI 학습",  done: true },
11          { id: 4, todo: "야구경기 관람", done: false },
12       ]
13    }),
14    getters : {
15       doneCount : (state)=> {
16          return state.todoList.filter((todoItem)=>todoItem.done === true).length;
17       }
18    },
19    actions : {
20       addTodo(todo) {
21          this.todoList.push({ id: new Date().getTime(), todo, done: false })
22       },
23       deleteTodo(id) {
24          let index = this.todoList.findIndex((todo) => todo.id === id);
25          this.todoList.splice(index, 1);
26       },
27       toggleDone(id) {
28          let index = this.todoList.findIndex((todo) => todo.id === id);
29          this.todoList[index].done = !this.todoList[index].done;
30       }
31    }
32 })
```

```
34 // 컴포지션 API 방식의 todoList2 스토어
35 export const useTodoList2Store = defineStore("todoList2", ()=> {
36    const state = reactive({
37       todoList : [
38          { id: 1, todo: "ES6학습",  done: false },
39          { id: 2, todo: "React학습",  done: false },
40          { id: 3, todo: "ContextAPI 학습",  done: true },
41          { id: 4, todo: "야구경기 관람", done: false },
42       ]
43    })
44
45    const addTodo = (todo) => {
46       state.todoList.push({ id: new Date().getTime(), todo, done: false })
47    }
48
49    const deleteTodo = (id) => {
50       let index = state.todoList.findIndex((todo) => todo.id === id);
51       state.todoList.splice(index, 1);
52    }
53
54    const toggleDone =  (id) => {
55       let index = state.todoList.findIndex((todo) => todo.id === id);
56       state.todoList[index].done = !state.todoList[index].done;
57    }
58
59    const doneCount = computed(()=> {
60       return state.todoList.filter((todoItem)=>todoItem.done === true).length;
61    })
62
63    const todoList = computed(()=> state.todoList);
64
65    return { todoList, doneCount, addTodo, deleteTodo, toggleDone };
66 })
```

3. 간단한 pinia 예제 작성

❖ 예제 13-02 : src/App1.vue – 옵션 API 컴포넌트

```vue
<template>
  <div>
    <h2>TodoList 테스트(Option API)</h2>
    <hr />
    할일 추가 :
    <input type="text" v-model="todo" />
    <button @click="addTodoHandler">추가</button>
    <hr />
    <ul>
      <li v-for="todoItem in todoList">
        <span style="cursor:pointer" @click="toggleDone(todoItem.id)">
        {{ todoItem.todo}} {{ todoItem.done ? "(완료)" : "" }}
        </span>
           
        <button @click="deleteTodo(todoItem.id)">삭제</button>
      </li>
    </ul>
    <div>완료된 할일 수 : {{doneCount}}</div>
  </div>
</template>
```

```vue
<script>
import { useTodoList1Store } from '@/stores/todoList.js'
import { mapState, mapActions } from 'pinia';

export default {
  name:"App1",
  data : ()=>({ todo:"" }),
  computed : {
    ...mapState(useTodoList1Store, ['todoList', 'doneCount' ])
  },
  methods : {
    ...mapActions(useTodoList1Store,  ['addTodo', 'deleteTodo', 'toggleDone']),
    addTodoHandler() {
      this.addTodo(this.todo);
      this.todo = "";
    }
  }
}
</script>
```

3. 간단한 pinia 예제 작성

❖ 예제 13-03 : src/App2.vue – Composition API 컴포넌트

```vue
<template>
  <div>
    <h2>TodoList 테스트(Composition API)</h2>
    <hr />
    할일 추가 :
    <input type="text" v-model="todo" />
    <button @click="addTodoHandler">추가</button>
    <hr />
    <ul>
      <li v-for="todoItem in todoList">
        <span style="cursor:pointer" @click="toggleDone(todoItem.id)">
        {{ todoItem.todo}} {{ todoItem.done ? "(완료)" : "" }}
        </span>
           
        <button @click="deleteTodo(todoItem.id)">삭제</button>
      </li>
    </ul>
    <div>완료된 할일 수 : {{doneCount}}</div>
  </div>
</template>
```

```vue
<script>
import { useTodoList2Store } from '@/stores/todoList.js'
import { ref, computed } from 'vue';

export default {
  name:"App2",
  setup() {
    const todo = ref("");

    const todoListStore = useTodoList2Store();
    const { todoList, addTodo, deleteTodo, toggleDone } = todoListStore;
    const doneCount = computed(()=>todoListStore.doneCount);

    const addTodoHandler = () => {
      addTodo(todo.value);
      todo.value = "";
    }
    return { todo, todoList, doneCount, addTodoHandler, deleteTodo, toggleDone }
  }
}
</script>
```

3. 간단한 pinia 예제 작성

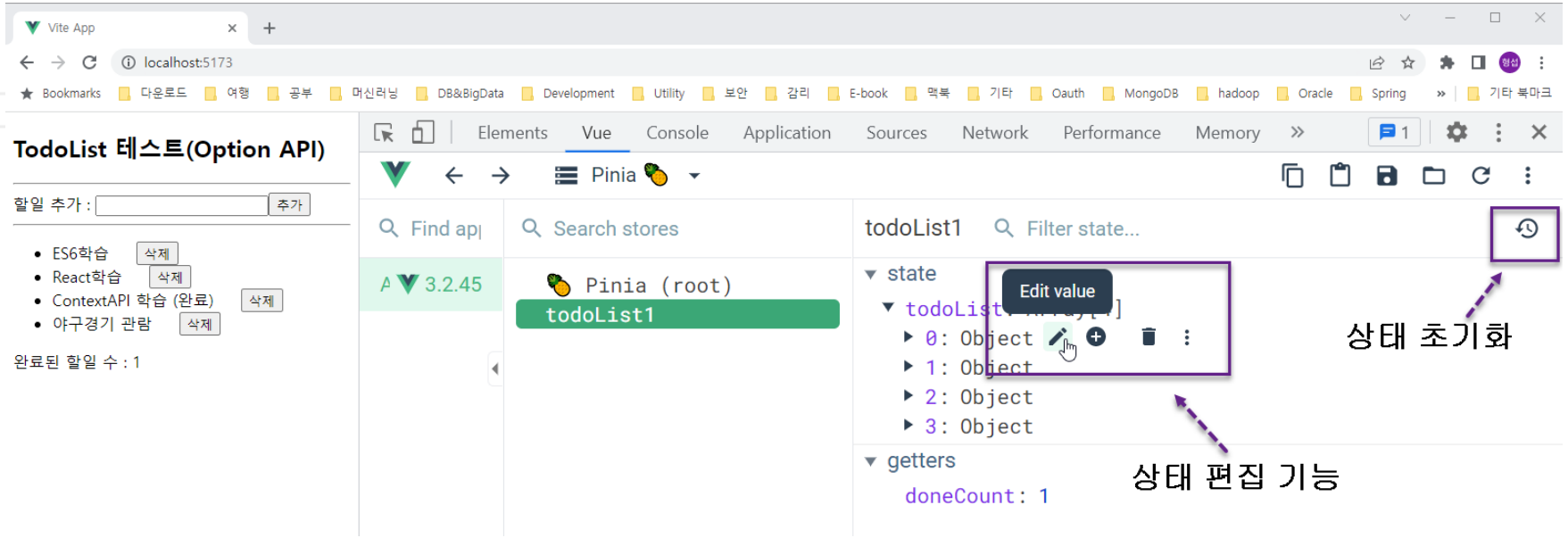
❖ 예제 13-04 : src/main.js 변경

- 3,4행의 주석을 바꿔가면서 테스트해 봄

```
1  import { createApp } from 'vue'
2  import { createPinia } from 'pinia'
3  import App from './App1.vue'
4  //import App from './App2.vue'
5
6  const pinia = createPinia()
7
8  const app = createApp(App)
9
10 app.use(pinia)
11 app.mount('#app')
```

❖실행결과

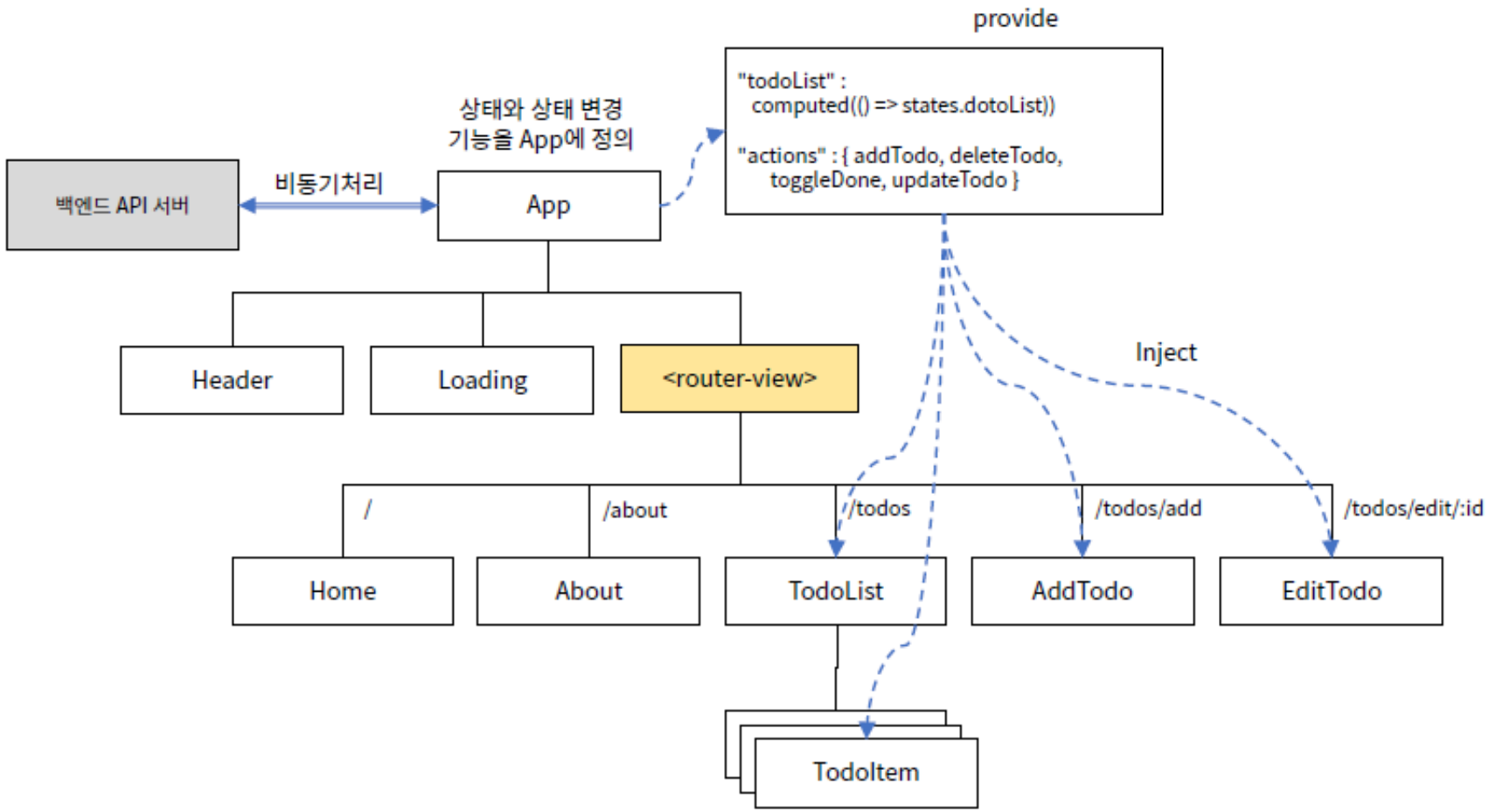
- Vue Devtools에서...



4. todolist-app-router 예제에 pinia 적용하기

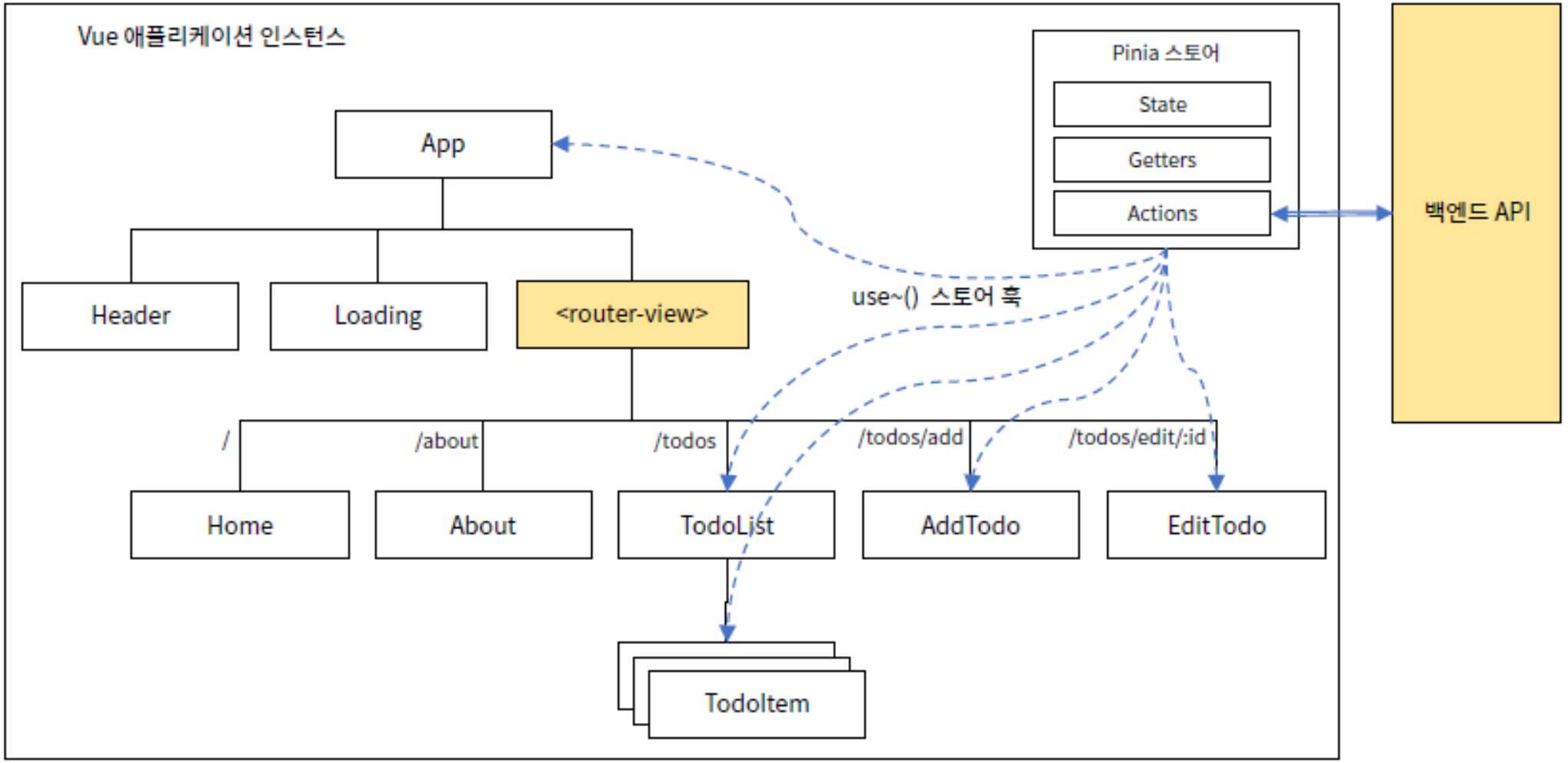
❖ 12장에서 작성했던 예제에 pinia 적용하도록 함

❖ 기존 예제 구조



4.1 예제 구조 검토

❖ pinia를 적용하여 변경할 예제 구조



4.2 백엔드 API 서버 실행

❖ 11,12장에서 사용했던 todosvc 이용

- https://github.com/stepanowon/todosvc 에서 다운로드 후 다음 명령 실행

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

4.3 기초 작업

- 12장에서 작성한 todolist-app-router 예제를 이어서 작업함
  - 만일 12장을 학습하지 않았다면 다음 github의 예제 중 ch12/todolist-app-router4 예제를 기반으로 시작하면 됨
  - https://github.com/stepanowon/vue3-quickstart
- 패키지 설치
  - npm install --save pinia
- 예제 13-05 : src/main.js 변경

```
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   import { createPinia } from 'pinia';
7
8   const pinia = createPinia();
9   const app = createApp(App);
10  app.use(pinia);
11  app.use(router);
12  app.mount('#app')
```

4.4 스토어 작성

❖ 예제 13-06 : src/stores/todoList.js 추가

- 가장 중요한 작업, 긴 코드

```
1  import { defineStore } from "pinia";
2  import axios from 'axios';
3
4  //1. BASEURI는 vite.config.js의 프록시 설정에 맞추어 지정합니다.
5  //2. owner는 샘플 백엔드 API 서버(https://sample.bmaster.kro.kr)의
6  //   문서를 참조하여 지정합니다. 기본값 gdhong 데이터는 존재합니다.
7  const owner = "gdhong";
8  //의도적 1초의 지연시간을 발생시키는 API 사용
9  const BASEURI = "/api/todolist_long";
10
11 //todoList1 스토어 정의
12 export const useTodoListStore = defineStore("todoList1", {
13    //상태 정의(todoList, isLoading)
14    //isLoading : Loading 컴포넌트를 보여줄지 여부 결정을 위한 상태
15    state : () => {
16       return {
17          todoList : [],
18          isLoading: false
19       }
20    },
21    //읽기 전용의 게터
22    //doneCount : 완료된 할일의 건수를 읽기 전용으로 제공
23    getters : {
24       doneCount : (state)=> {
25          const filtered = state.todoList.filter((todoItem)=>todoItem.done === true);
26          return filtered.length;
27       }
28    },
29    //액션
30    //비동기처리 시작에서 isLoading=true, 비동기 처리 완료 후 isLoading=false
31    actions : {
32       async fetchTodoList() {
33          this.isLoading = true;
34          try {
35             const response = await axios.get(BASEURI + `/${owner}`);
36             if (response.status === 200) {
37                this.todoList = response.data;
38             } else {
39                alert('데이터 조회 실패');
40             }
41             this.isLoading = false;
42          } catch(error) {
43             alert('에러발생 :' + error);
44             this.isLoading = false;
45          }
46       },
47       async addTodo({ todo, desc }, successCallback) {
48          if (!todo || todo.trim()==="") {
49             alert('할일은 반드시 입력해야 합니다');
50             return;
51          }
52          this.isLoading = true;
53          try {
54             const payload = { todo, desc };
55             const response = await axios.post(BASEURI + `/${owner}`, payload)
56             if (response.data.status === "success") {
57                this.todoList.push({ id: response.data.item.id, todo, desc, done: false })
58                successCallback();
59              } else {
60                alert('Todo 추가 실패 : ' + response.data.message);
61              }
62             this.isLoading = false;
63          } catch(error) {
64             alert('에러발생 :' + error);
65             this.isLoading = false;
66          }
67       },
```

4.4 스토어 작성

```
68      async updateTodo({ id, todo, desc, done }, successCallback) {
69        if (!todo || todo.trim()==="") {
70          alert('할일은 반드시 입력해야 합니다');
71          return;
72        }
73        this.isLoading = true;
74        try {
75          const payload = { todo, desc, done };
76          const response = await axios.put(BASEURI + `/${owner}/${id}`, payload)
77          if (response.data.status === "success") {
78            let index = this.todoList.findIndex((todo) => todo.id === id);
79            this.todoList[index] = { id, todo, desc, done };
80            successCallback();
81          } else {
82            alert('Todo 변경 실패 : ' + response.data.message);
83          }
84          this.isLoading = false;
85        } catch(error) {
86          alert('에러발생 :' + error);
87          this.isLoading = false;
88        }
89      },
90      async deleteTodo(id) {
91        this.isLoading = true;
92        try {
93          const response = await axios.delete(BASEURI + `/${owner}/${id}`)
94          if (response.data.status === "success") {
95            let index = this.todoList.findIndex((todo) => todo.id === id);
96            this.todoList.splice(index, 1);
97          } else {
98            alert('Todo 삭제 실패 : ' + response.data.message);
99          }
100         this.isLoading = false;
101       } catch(error) {
102         alert('에러발생 :' + error);
103         this.isLoading = false;
104       }
105     },
```

```
106     async toggleDone(id) {
107       this.isLoading = true;
108       try {
109         const response = await axios.put(BASEURI + `/${owner}/${id}/done`)
110         if (response.data.status === "success") {
111           let index = this.todoList.findIndex((todo) => todo.id === id);
112           this.todoList[index].done = !this.todoList[index].done;
113         } else {
114           alert('Todo 완료 변경 실패 : ' + response.data.message);
115         }
116         this.isLoading = false;
117       } catch(error) {
118         alert('에러발생 :' + error);
119         this.isLoading = false;
120       }
121     },
122   }
123 })
```

이 예제에서는 옵션 API 방식의 스토어를 작성하였습니다. 컴포지션 API 방식의 스토어 예제는 다음 github에서 예제를 다운로드하여 ch13/todolist-app-router5 예제의 src/stores/todoList2.js를 확인해보세요. 컴포지션 API 예제는 옵션 API를 대체해서 실행할 수 있도록 작성되었습니다. 참조하세요.

4.5 App 컴포넌트 변경

❖ 예제 13-07 : src/App.vue 변경

```
<template>
  <div class="container">
    <Header />
    <router-view />
    <Loading v-if="isLoading" />
  </div>
</template>

<script setup>
import { computed } from 'vue';
import Header from '@/components/Header.vue'
import { useTodoListStore } from '@/stores/todoList.js'
import Loading from '@/components/Loading.vue'

const todoListStore = useTodoListStore();
const isLoading = computed(()=>todoListStore.isLoading);
const fetchTodoList = todoListStore.fetchTodoList;
fetchTodoList();
</script>
```

4.6 TodoList, TodoItem 컴포넌트 변경

❖ 예제 13-08 : src/pages/TodoList.vue 변경

```
1   <template>
2     <div class="row"> ···
9     </div>
10    <div class="row">
11      <div class="col"> ···
15      </div>
16      <span>완료된 할일 : {{doneCount}}</span>
17    </div>
18  </template>
19
20  <script setup>
21  import { computed } from 'vue';
22  import { useTodoListStore } from '@/stores/todoList.js'
23  import TodoItem from '@/pages/TodoItem.vue'
24
25  const todoListStore = useTodoListStore();
26  const { fetchTodoList } = todoListStore;
27  const doneCount = computed(()=>todoListStore.doneCount);
28  const todoList = computed(()=>todoListStore.todoList);
29  </script>
```

4.6 TodoList, TodoItem 컴포넌트 변경

❖ 예제 13-09 : src/pages/TodoItem.vue 변경

```
 1 > <template> ···
15   </template>
16
17   <script setup>
18   import { useRouter } from 'vue-router';
19   import { useTodoListStore } from '@/stores/todoList.js'
20
21   defineProps({
22     todoItem: { Type: Object, required:true }
23   })
24
25   const router = useRouter();
26   const todoListStore = useTodoListStore();
27   const { deleteTodo, toggleDone } = todoListStore;
28   </script>
```

4.7 AddTodo, EditTodo 컴포넌트 변경

❖ 예제 13-10 : src/pages/AddTodo.vue 변경

```
<template> ···
</template>

<script setup>
import { reactive } from 'vue';
import { useRouter } from 'vue-router';
import { useTodoListStore } from '@/stores/todoList.js'

const router = useRouter();
const { addTodo } = useTodoListStore();
const todoItem =  reactive({ todo:"", desc:"" })

const addTodoHandler = () => {
    let { todo } = todoItem;
    if (!todo || todo.trim()==="") {
        alert('할일은 반드시 입력해야 합니다');
        return;
    }
    addTodo({ ...todoItem }, ()=>{
        router.push('/todos')
    });
}
</script>
```

4.7 AddTodo, EditTodo 컴포넌트 변경

❖ 예제 13-11 : src/pages/EditTodo.vue 변경

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

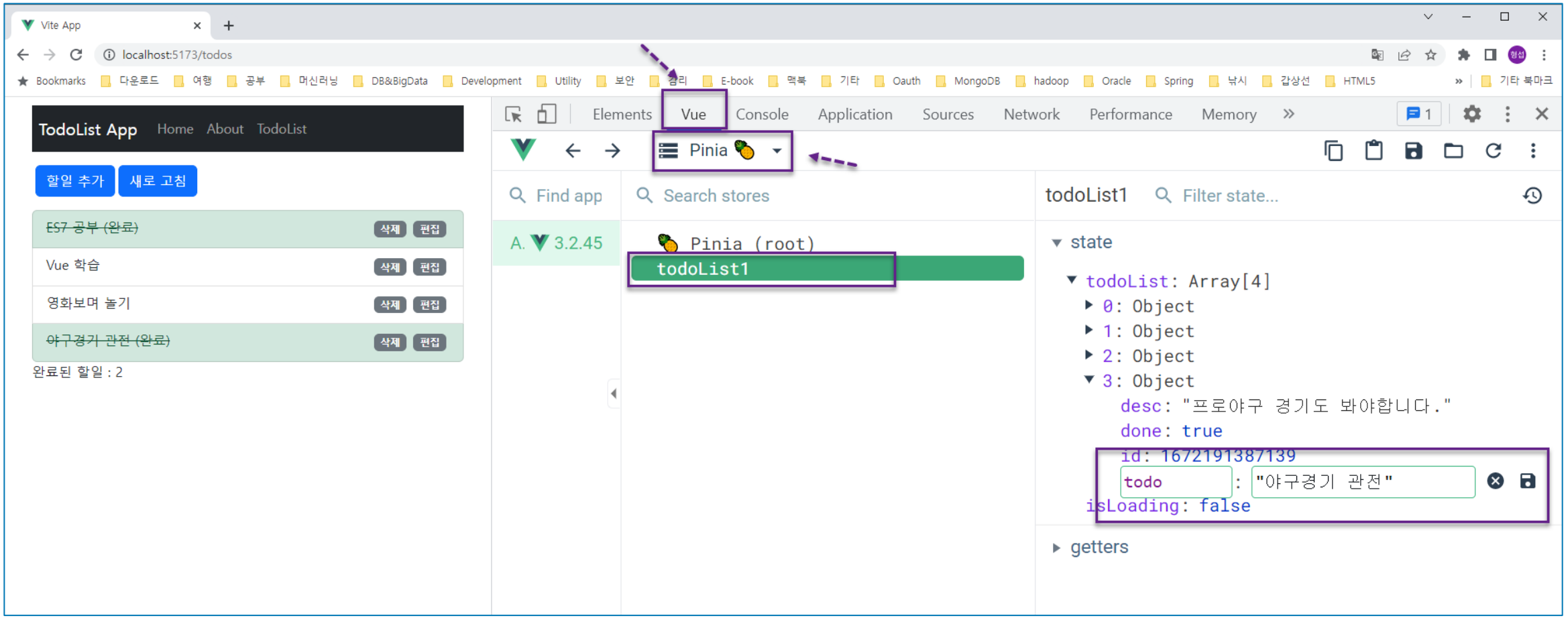
기존 provide/inject를 이용한 예제에서는 App 컴포넌트에서 todoList 상태를 provide()로 등록할 때 computed()를 이용해 반응성을 잃지 않도록 작성했기 때문에 데이터를 이용할 때 todoList.value로 접근해야 했지만 변경된 예제에서는 상태를 참조하기 때문에 예제 13-11의 16행과 같이 todoList를 직접 이용하도록 작성한다는 점에 주의하세요.

```
// 기존 예제 App 컴포넌트에서의 provide 호출 코드
provide('todoList', computed(()=>states.todoList));

// 기존 예제의 matchedTodoItem 변수 생성 코드
const matchedTodoItem =
    todoList.value.find((item)=> item.id === parseInt(currentRoute.params.id))
```

4.7 AddTodo, EditTodo 컴포넌트 변경

❖ 실행 결과 확인



5. 마무리

이제까지 차세대 Vue 애플리케이션을 위한 상태 관리 라이브러리인 pinia에 대해 살펴보았습니다. 사용 방법이 간단하고 적용하기 쉬워서 꼭 한번 실무에 적용해보시라고 추천드립니다. 특히 Vue나 React와 같은 프런트엔드 애플리케이션의 특징이 상태(State)가 바뀌면 UI를 렌더링하는 구조입니다. 애플리케이션을 디버깅할 때 가장 많이 참조하고 추적해야 하는 대상 데이터가 상태입니다. pinia와 같은 애플리케이션 수준의 상태 관리 라이브러리를 사용하면 상태의 추적이 더 편리해질 것입니다.