KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

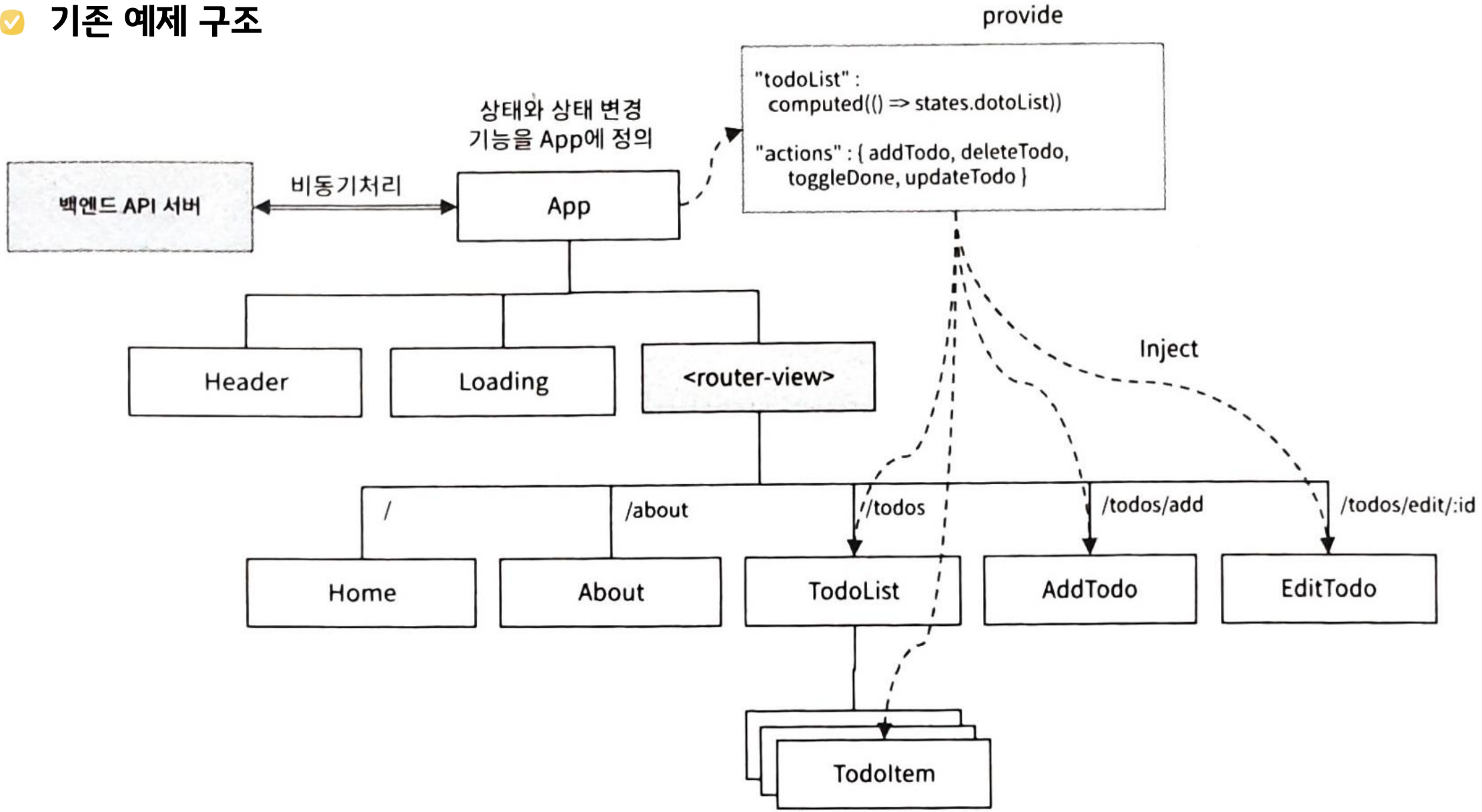
todolist 예제에 pinia 적용하기

[KB] IT's Your Life

KB 국민은행

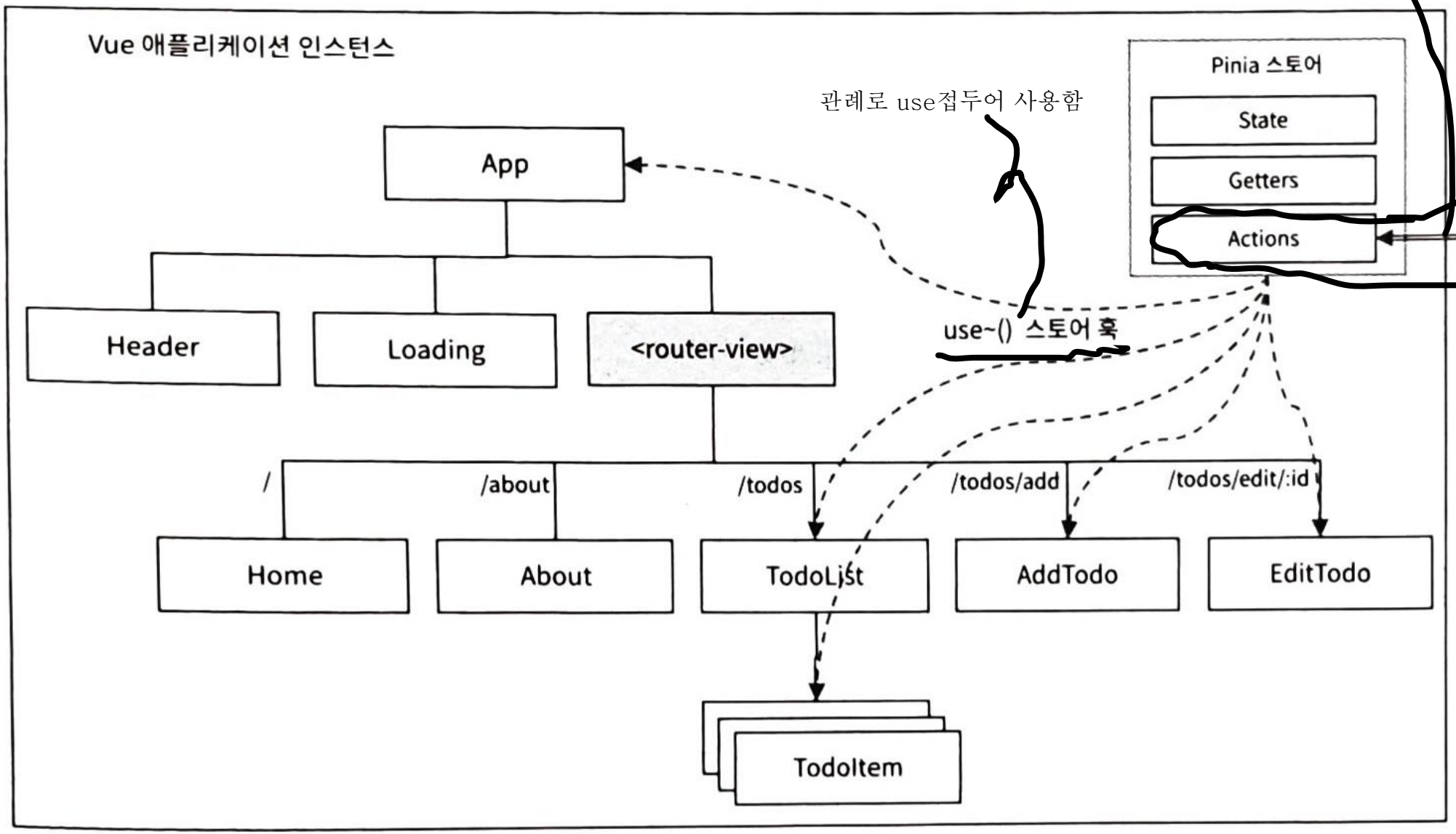
provide/inject 활용한 구조

기존 예제 구조



1 기존 예제 구조 검토

새롭게 변경할 pinia를 적용한 예제 구조



매번 필요할 때마다
임포트해서 초기화해서
사용해야 하냐

지금 공통으로
같은 스토어를 사용하는
컴포넌트들의
가상 더미 부모를 두면
된다!

더미 부모는 router-view

뒤에 구조

1 기존 예제 구조 검토

중첩 라우터 기법

pinia 추가

추가 설치`

- npm i pinia

인증정보처럼 앱 전체에서 상시 필요한 정보는
스토어를 통해서 관리해야하는 대표적인 정보다.

또 다른 이슈 새로고침.
스토어의 상태가 업데이트가 많이 된 상태에서 새로고침이 되면
스토어의 상태도 리셋 되어버린다. 이건 provide/inject방식도 그럼. 어칼까?

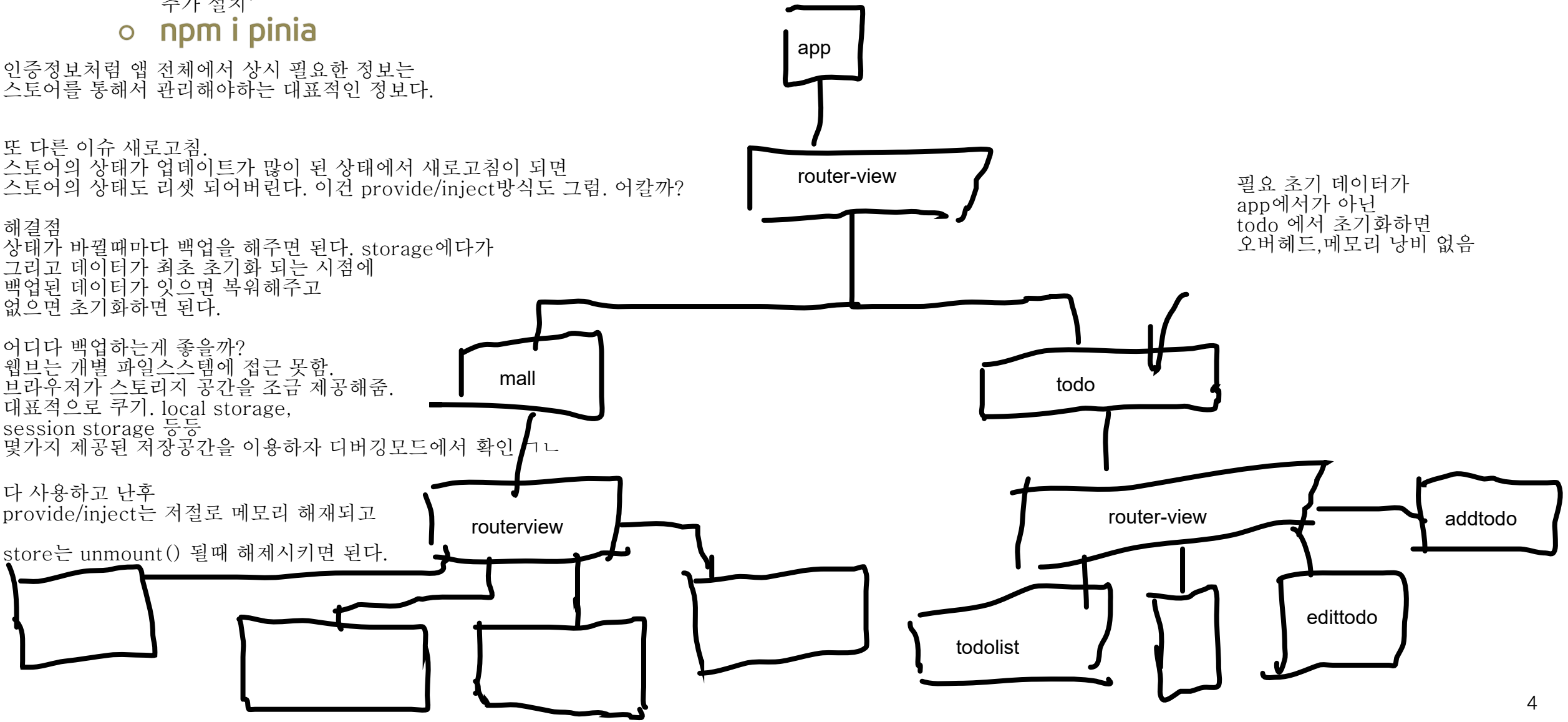
해결점
상태가 바뀔때마다 백업을 해주면 된다. storage에다가
그리고 데이터가 최초 초기화 되는 시점에
백업된 데이터가 잇으면 복워해주고
없으면 초기화하면 된다.

어디다 백업하는게 좋을까?
웹브는 개별 파일스스템에 접근 못함.
브라우저가 스토리지 공간을 조금 제공해줌.
대표적으로 쿠기. local storage,
session storage 등등
몇가지 제공된 저장공간을 이용하자 디버깅모드에서 확인ㄱㄴ

다 사용하고 난후
provide/inject는 저절로 메모리 해재되고

store는 unmount() 될때 해제시키면 된다.

필요 초기 데이터가
app에서가 아닌
todo 에서 초기화하면
오버헤드,메모리 낭비 없음



src/main.js 변경

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

import { createApp } from 'vue';
import App from './App.vue';
import router from './router';
import 'bootstrap/dist/css/bootstrap.css';

import { createPinia } from 'pinia';

const pinia = createPinia();
const app = createApp(App);
app.use(pinia);
app.use(router);
app.mount('#app')
```

초기화 말고 추가 설치시에는
main.js에서
직접 써야함

2 스토어 작성

src/stores/todoList.js 추가

액시오스를 이용해 서버와 비동기 작업

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


export const useTodoListStore = defineStore('todoList', () => {
  const BASEURI = '/api/todos';
  const state = reactive({ todoList: [] });
```

그냥 초기화보다
백업할 데이터 있으면 백업하고 없으면
그때 초기화!

src/stores/todoList.js 추가

```
//TodoList 목록을 조회합니다.
const fetchTodoList = async () => {
  try {
    const response = await axios.get(BASEURI);
    if (response.status === 200) {
      state.todoList = response.data;
    } else {
      alert('데이터 조회 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

클로저 기반이기 때문에 상태정보가 항상 최신정보.

src/stores/todoList.js 추가

```
// 새로운 TodoItem을 추가합니다.
const addTodo = async ({ todo, desc }, successCallback) => {
  try {
    const payload = { todo, desc };
    const response = await axios.post(BASEURI, payload);
    if (response.status === 201) {
      state.todoList.push({ ...response.data, done: false });
      successCallback();
    } else {
      alert('Todo 추가 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

src/stores/todoList.js 추가

```
// 기존 TodoItem을 변경합니다.
const updateTodo = async ({ id, todo, desc, done }, successCallback) => {
  try {
    const payload = { id, todo, desc, done };
    const response = await axios.put(BASEURI + `/${id}`, payload);
    if (response.status === 200) {
      let index = state.todoList.findIndex((todo) => todo.id === id);
      state.todoList[index] = payload;
      successCallback();
    } else {
      alert('Todo 변경 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

src/stores/todoList.js 추가

```
//기존 TodoItem을 삭제합니다.
const deleteTodo = async (id) => {
  try {
    const response = await axios.delete(BASEURI + `/${id}`);

    if (response.status === 200) {
      let index = state.todoList.findIndex((todo) => todo.id === id);
      state.todoList.splice(index, 1);
    } else {
      alert('Todo 삭제 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

src/stores/todoList.js 추가

```
//기존 TodoItem의 완료여부(done) 값을 토글합니다.
const toggleDone = async (id) => {
  try {
    let todo = state.todoList.find((todo) => todo.id === id);
    let payload = { ...todo, done: !todo.done };
    const response = await axios.put(BASEURI + `/${id}`, payload);
    if (response.status === 200) {
      todo.done = payload.done;
    } else {
      alert('Todo 완료 변경 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

src/stores/todoList.js 추가

```
  const todoList = computed(() => state.todoList);
  const isLoading = computed(() => state.isLoading);
  const doneCount = computed(() => {
    const filtered = state.todoList.filter((todoItem) => todoItem.done === true);
    return filtered.length;
  });

  return { todoList, isLoading, doneCount, fetchTodoList, addTodo, deleteTodo, updateTodo, toggleDone };
});
```

읽기 전용 데이터로 직접 정의해야함

src/App.vue 변경

여기서 app.vue역할
초기 데이터 준비

```
<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>
```

3 TodoList, TodoItem 컴포넌트 변경

3 TodoList, TodoItem 컴포넌트 변경

src/pages/TodoList.vue 변경

여기선 pinia store사용!

```
<template>
    …
    <div class="row">
    …
        <span>완료된 할일 : {{doneCount}}</span>
    </div>
</template>

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

const todoListStore = useTodoListStore();
const { fetchTodoList } = todoListStore;
const doneCount = computed(()=>todoListStore.doneCount);
const todoList = computed(()=>todoListStore.todoList);
</script>
```

src/components/TodoItem.vue 변경 여기서도 pinia store 활용

```
<template>
  …
</template>

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

defineProps({
  todoItem: { Type: Object, required:true }
})

const router = useRouter();
const todoListStore = useTodoListStore();
const { deleteTodo, toggleDone } = todoListStore;
</script>
```

4 AddTodo, EditTodo 컴포넌트 변경

src/pages/AddTodo.vue 변경

```
…
<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>
```

src/pages/EditTodo.vue 변경

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

const router = useRouter();
const currentRoute = useRoute();

const { todoList, updateTodo, } = useTodoListStore();

const matchedTodoItem = todoList.find((item)=> item.id === currentRoute.params.id);
if (!matchedTodoItem)  {
    router.push('/todos');
}

const todoItem =  reactive({ ...matchedTodoItem });
```

src/pages/EditTodo.vue 변경

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