KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

vue-router와 axios를 사용한 예제

[KB] IT's Your Life

기획 -> 화면 설계 (스토리 보드) -> 컴포넌트 트리 구조 나와야함 -> 컴포넌트들 구현 분업

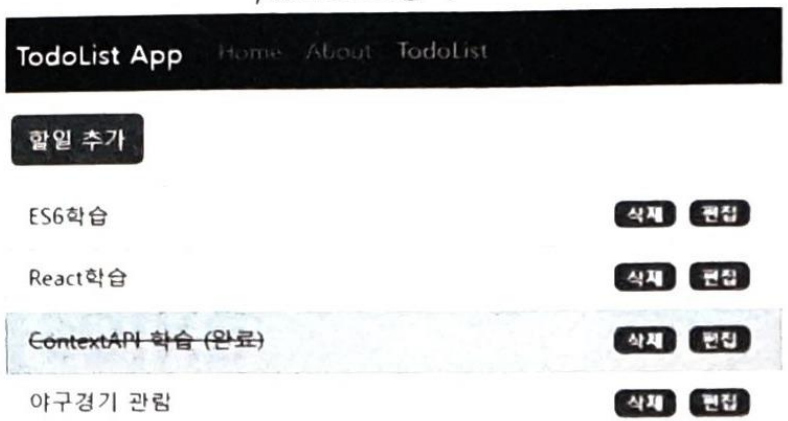
-> 데이터 운영 어떻게 할것이냐를 정해야해. provide/injection axios json-server 등등...

KB 국민은행

작성할 화면

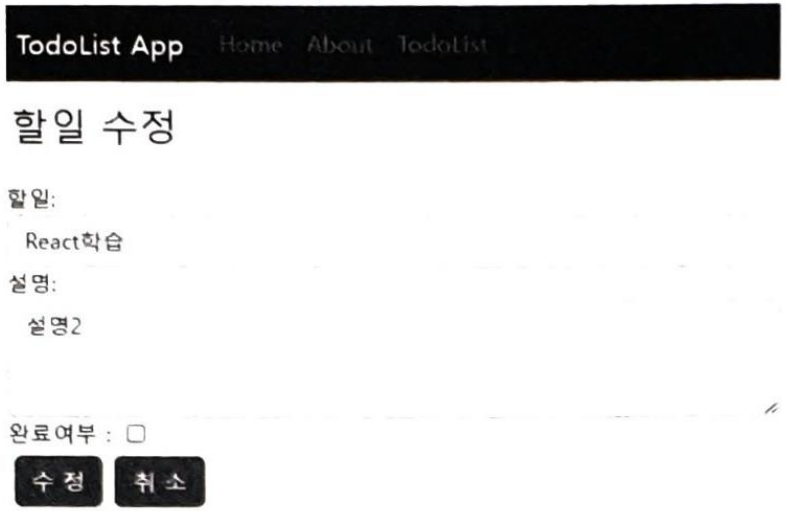
/home, /about 요청시

/todos 요청시



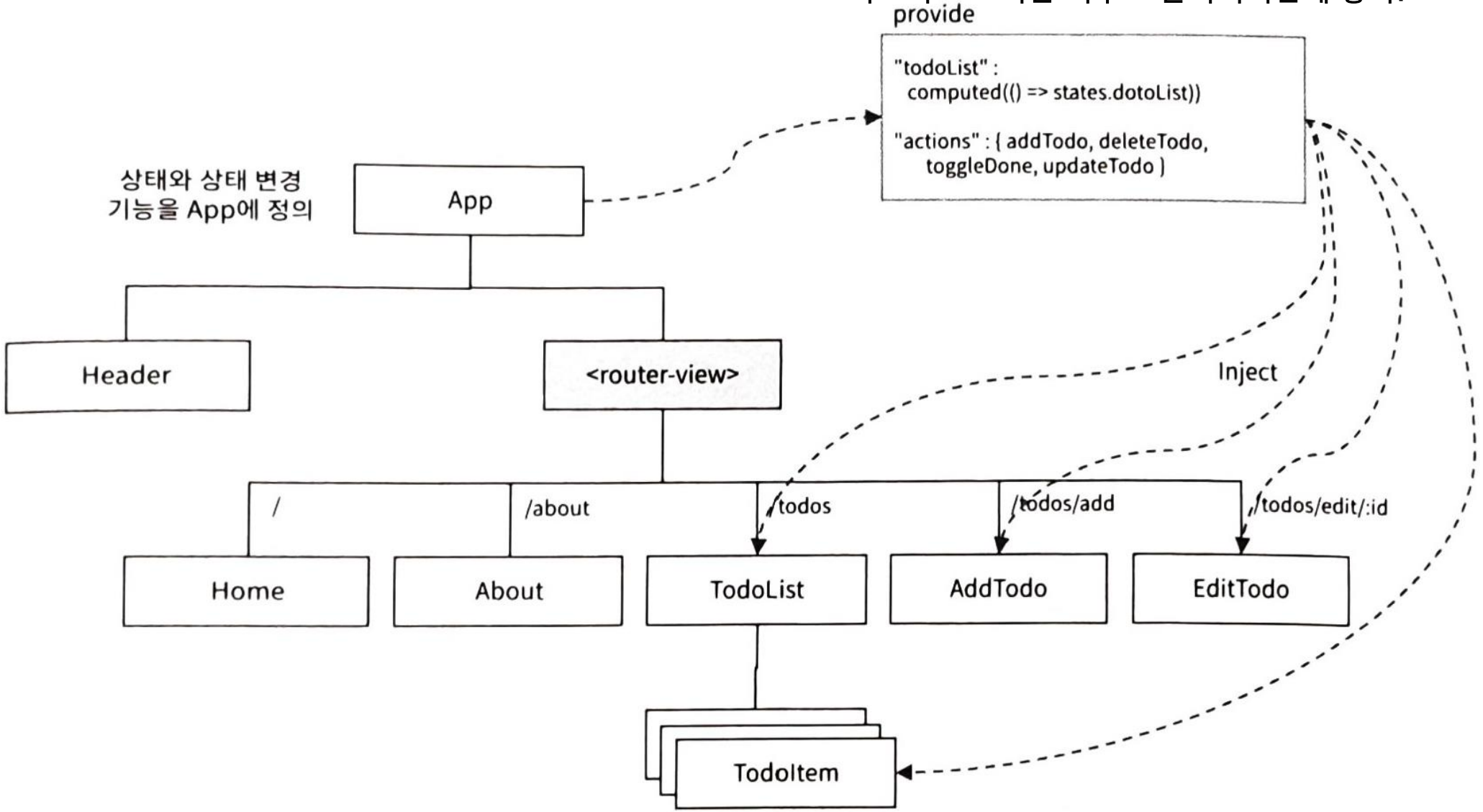
/todos/add 요청시

/todos/edit/:id 요청시



데이터 파트와
메서드 파트(비즈니스파트 or 데이터가공 파트)
데이터는 데이터를 갖고 잇는 곳에서 가공.
각 페이지에서 가공하면 안돼
각 컴포넌트는 출력에만 신경쓰게끔. 가공X.
비즈니스 로직은 외부로 분리시키는게 좋다.

컴포넌트 계층 구조



첫번째 예제는 axios없이

상태 데이터

[상태 데이터]

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

[상태 변경 기능] 함수 구조 정의

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

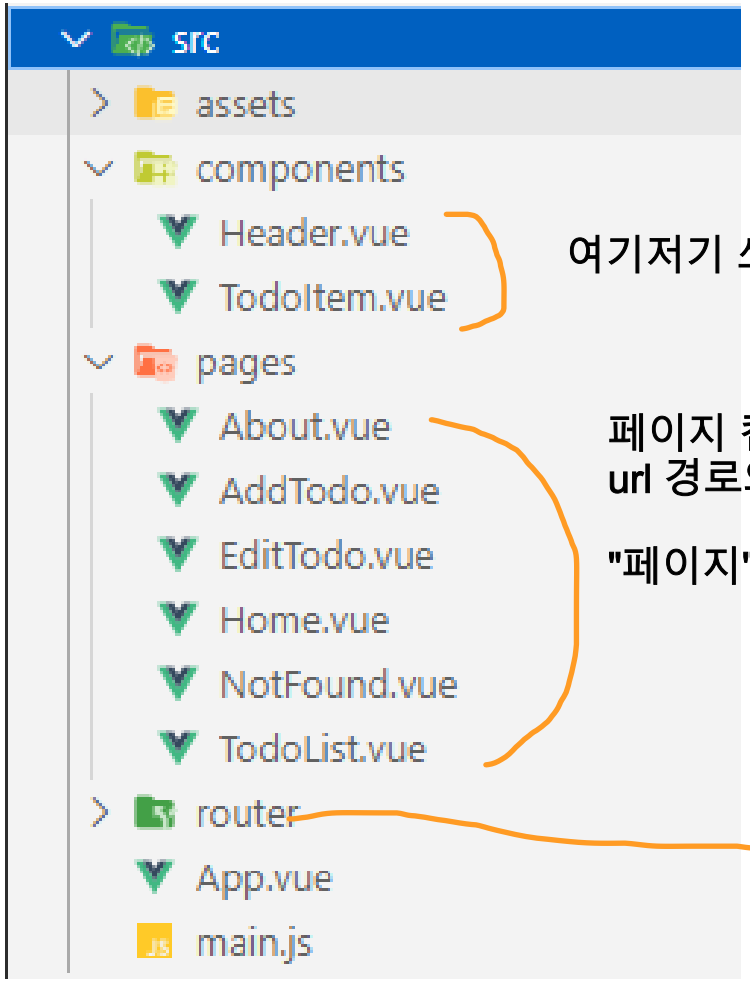
대상

프로젝트 생성

npm init vue todolist-app-router
→ 라우트 기능 설정
cd todolist-app-router
npm install
npm install bootstrap@5 axios

디렉토리및 파일 구조

일반 컴포넌트도 계층화와 카테고리화 하는 전략
componenets - common, board, todos, home
쓰이는 곳으로 분리하는 전략이 좋음.



여기저기 쓰이는 컴포넌트

페이지 컴포넌트 이름과 디렉토리 정책은
url 경로와 일치시키는 전략이 좋다.

"페이지" 컴포넌트

router 폴더 안에는 index.js가 들어있다.
임포트할 때 router폴더가 모듈이 된다. 기억나죠?

src/pages/Home.vue

```
<template>
    <div class="card card-body">
        <h2>Home</h2>
    </div>
</template>
```

○ 나머지 페이지 컴포넌트도 같은 방식으로 작성

src/pages/NotFound.vue

```
<template>
    <div class="m-3">
        <h3>존재하지 않는 경로</h3>
        <p>요청 경로 : {{currentRoute.path}}</p>
    </div>
</template>

<script setup>
import { useRoute } from 'vue-router'
const currentRoute = useRoute();
</script>
```

현재 경로 얻음

src/main.js

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

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

const app = createApp(App)

app.use(router)

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

src/assets/main.css

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

사용할 CSS 정의

src/components/Header.vue

```
<template>
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
      <span class="navbar-brand ps-2">TodoList App</span>
      <button class="navbar-toggler" type="button" @click="isNavShow = !isNavShow">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div :class="isNavShow ? 'collapse navbar-collapse show' : 'collapse navbar-collapse'">
        <ul class="navbar-nav">
          <li class="nav-item">
            <router-link class="nav-link" to="/">Home</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/about">About</router-link>
          </li>
          <li class="nav-item">
            <router-link class="nav-link" to="/todos">TodoList</router-link>
          </li>
        </ul>
      </div>
    </nav>
</template>
```

재사용성을 높이려면
지금 다 정해져 있는데
이 부분들을 다 변수처리하여
일반화 할 필요가 있다.

변수에 들어갈 내용들은

설정 파일로 따로 빼서 설정객체
에 기록해 놨다가 사용

config : {
  title:~, subtitile : ~ , menus : [..],
}

패스로

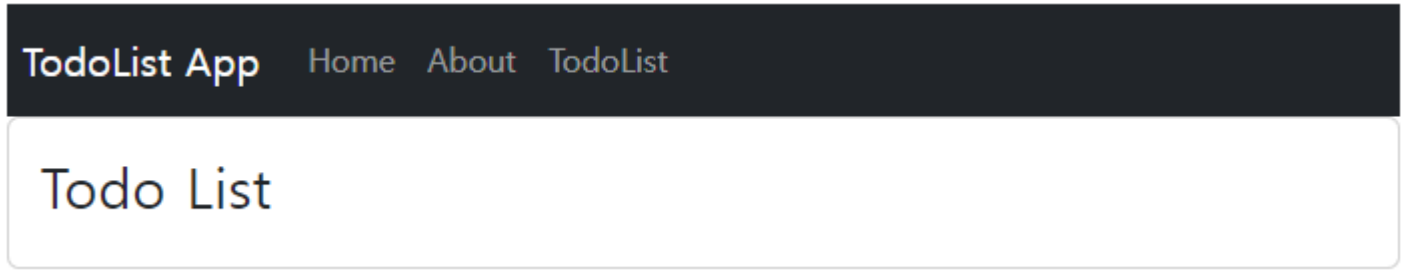
a태그 대신 router-link태그
href속성값 내용을 to속성에

패스가 아닌 name으로 하려면
:to="~~"
바인딩 해야한다.

src/components/Header.vue

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

const isNavShow = ref(false);
</script>
```



src/App.vue

```
<template>
    <div class="container">
        <Header />
        <router-view />
    </div>
</template>

<script setup>
import { reactive, computed, provide } from 'vue'
import Header from '@/components/Header.vue'

const states = reactive({
  todoList : [
    { id: 1, todo: "ES6학습", desc: "설명1", done: false },
    { id: 2, todo: "React학습", desc: "설명2", done: false },
    { id: 3, todo: "ContextAPI 학습", desc: "설명3", done: true },
    { id: 4, todo: "야구경기 관람", desc: "설명4", done: false },
  ]
})
```

배열 전체 교체를
위해서
객체로 한번 더 래핑

src/App.vue

탑레벨 함수처럼 보이지만 setup 함수의 내부함수다. 즉 클로저에 다 포함된다.

```
const addTodo = ({ todo, desc }) => {
  states.todoList.push({ id: new Date().getTime(), todo, desc, done: false })
};

const updateTodo = ({ id, todo, desc, done }) => {
  let index = states.todoList.findIndex((todo) => todo.id === id);
  states.todoList[index] = { ...states.todoList[index], todo, desc, done };
};

const deleteTodo = (id) => {
  let index = states.todoList.findIndex((todo) => todo.id === id);
  states.todoList.splice(index, 1);
}

const toggleDone = (id) => {
  let index = states.todoList.findIndex((todo) => todo.id === id);
  states.todoList[index].done = !states.todoList[index].done;
}

provide('todoList', computed(()=>states.todoList))
provide('actions', { addTodo, deleteTodo, toggleDone, updateTodo })
</script>
```

js에서는 이렇게 간편하게 표현가능하다. 기존에 있는 이름이면 그 값으로 업데이트 새로운 이름이면 추가.

Array.prototype 객체에 해당 역할을 하는 메소드removeAt()를 추가하여 쉽게 사용할 수 있게금 할 수 있다.

배열을 반환하는 반응형을 todoList이름으로 provide

데이터 가공을 위한 메소드들을 모아서 provide

왜 이렇게 했지?
provide('todoList',state.todoList)
이렇게 해도 되잖아
위는 컴포넌트에서 쓰기도 허용된다.
정책에 위배된다. 본문에 나온것 처럼
해야 읽기 전용으로 사용할 수 있게
제공할 수 있다.
데이터 가공은 데이터를 가진 공간의
메소드를 통해서만 가능하게끔
컴포넌트는 출력에만 신경쓰고

src/components/TodoItem.vue

```
<template>
    <li :class="todoItem.done ? 'list-group-item list-group-item-success' : 'list-group-item'">
      <span :class="todoItem.done ? 'todo-done pointer' : 'pointer'"
        @click="toggleDone(todoItem.id)">
        {{todoItem.todo}}
        {{todoItem.done ? '(완료)' : '' }}
      </span>
      <span class="float-end badge bg-secondary pointer m-1"
        @click="router.push(`/todos/edit/${todoItem.id}`)">
        편집</span>
      <span class="float-end badge bg-secondary pointer m-1"
        @click="deleteTodo(todoItem.id)">
        삭제</span>
    </li>
</template>
```

동적 파라미터를 사용

다른 방법
router.push(
{name: ~, params: {id} }
)

직접 데이터 가공 로직을 하지 않고

만들어진 것을 사용하고 있죠?

src/components/TodoItem.vue

```
<script setup>
import { useRouter } from 'vue-router';
import { inject } from 'vue';

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

const router = useRouter();
const { deleteTodo, toggleDone } = inject('actions');
</script>
```

todoitem컴포넌트가
태그가
부모로 부터 받을 매개변수 이름 todoitem

provide햇던 actions 중에서 2개 추출

페이지 컴포넌트

src/pages/TodoList.vue

데이터 출력만 신경쓰고 있죠?

```
<template>
    <div class="row">
        <div class="col p-3">
            <router-link class="btn btn-primary" to="/todos/add">
            할일 추가
            </router-link>
        </div>
    </div>
    <div class="row">
        <div class="col">
            <ul class="list-group">
                <TodoItem v-for="todoItem in todoList" :key="todoItem.id" :todoItem="todoItem" />
            </ul>
        </div>
    </div>
</template>

<script setup>
import {inject} from 'vue';
import TodoItem from '@/components/TodoItem.vue'

const todoList = inject('todoList');
</script>
```

이거 아까 읽기전용으로 provide했었죠?

페이지 컴포넌트

src/pages/AddTodo.vue

```
<template>
    <div class="row">
        <div class="col p-3">
            <h2>할일 추가</h2>
        </div>
    </div>
    <div class="row">
        <div class="col">
            <div class="form-group">
                <label htmlFor="todo">할일 :</label>
                <input type="text" class="form-control" id="todo" v-model="todoItem.todo" />
            </div>
            <div class="form-group">
                <label htmlFor="desc">설명 :</label>
                <textarea class="form-control" rows="3" id="desc" v-model="todoItem.desc"></textarea>
            </div>
            <div class="form-group">
                <button type="button" class="btn btn-primary m-1" @click="addTodoHandler"> 추 가
            </button>
                <button type="button" class="btn btn-primary m-1" @click="router.push('/todos')"> 취 소
            </button>
            </div>
        </div>
    </div>
</template>
```

src/pages/AddTodo.vue

```
<script setup>
import { inject, reactive } from 'vue';
import { useRouter } from 'vue-router';

const router = useRouter();
const { addTodo } = inject('actions');
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>
```

add(todoItem)말고 이렇게 한 이유는? 복사를 해야지.
todoItem 자체는 reactive객체 즉 proxy객체다. proxy객체가 넘어간다. 안된다.
순수한 데이터 객체를 넘기는 것이 목표다.

본문에 있는 표현이
전개 연산자를 이용하여 들어있는 내용들로 새로운 객체를 만들어
넘기는 연산이다.

src/pages/EditTodo.vue

```
<template>
    <div class="row">
        <div class="col p-3">
            <h2>할일 수정</h2>
        </div>
    </div>
    <div class="row">
        <div class="col">
            <div class="form-group">
                <label htmlFor="todo">할일:</label>
                <input type="text" class="form-control" id="todo" v-model="todoItem.todo" />
            </div>
            <div class="form-group">
                <label htmlFor="desc">설명:</label>
                <textarea class="form-control" rows="3" id="desc" v-model="todoItem.desc"></textarea>
            </div>
            <div class="form-group">
                <label htmlFor="done">완료여부 : </label> 
                <input type="checkbox" v-model="todoItem.done" />
            </div>
```

만약 해당 단위를 컴포넌트화 시키면
V-MODEL 부분을
사용자정의V-model로 처리를 해야한다.

src/pages/EditTodo.vue

번외 팁 필수항목 안채워질때 비활성화
:disabled=" 검증하는 js 코드 표현식 "

```
            <div class="form-group">
                <button type="button" class="btn btn-primary m-1" @click="updateTodoHandler">
                  수 정
                </button>
                <button type="button" class="btn btn-primary m-1" @click="router.push('/todos')">
                  취 소
                </button>
            </div>
        </div>
    </div>
</template>
```

src/pages/EditTodo.vue

```
<script setup>
import { inject, reactive } from 'vue';
import { useRouter, useRoute } from 'vue-router';

const todoList = inject('todoList');
const { updateTodo } = inject('actions');
const router = useRouter();
const currentRoute = useRoute();

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

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

번외
findindex메서드는 못찾으면 -1
find메서드는 못찾으면 null 혹은 undefined

ref도 아닌데 왜? computed감쌌잖아.
computed속성이잖아. ref랑 비슷하게 취급하면된다

이번엔 왜? reactive(matched)안함?
복사 안하고 원본데이터를 넘겨주면

원본과의 관계를 끊기 위해서

바로 원본이 수정되면 중간에
취소할 경우 못돌아감

그래서 복사

유효성
검사

router provide/inject 프론트까지

유효성 검사를 지원하는 모듈도 있다.

백엔드 API 실행과 프록시 설정

axios 설치

- npm install axios

vite.config.js 변경

접두어

```
…
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:3000',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, ''),
      },
    },
  },
})
```

db.json

JSON-SERVER 활용할 거야. 주의 id는 문자열로.
내부에서 ===연산자로 찾거든

```
{
  "todos": [
    { "id": "1", "todo": "ES6학습", "desc": "설명1", "done": false },
    { "id": "2", "todo": "React학습", "desc": "설명2", "done": false },
    { "id": "3", "todo": "ContextAPI 학습", "desc": "설명3", "done": true },
    { "id": "4", "todo": "야구경기 관람", "desc": "설명4", "done": false }
  ]
}
```

id라는 필드가 없다면?? 식별자는 어떻게 되는 걸까?
없더라도 실제로 없더라도 id필드와 값을 임시로 발급해줌.
하지만 파일에서는 나타나지 않는다.

데이터 서버 기동

- 새 터미널에서

  npx json-server db.json 요건 일회성 실행버전

src/App.vue

```
<template>
  <div class="container">
      <Header />
      <router-view />
  </div>
</template>

<script setup>
import { reactive, provide, computed } from 'vue'
import Header from '@/components/Header.vue'
import axios from 'axios';

const BASEURI = "/api/todos"; 엔드포인트에 대한 기본 uri
const states = reactive({ todoList:[]  })
```

src/App.vue

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

엔드포인트 기본 uri에 대한 response 받기

객체로 래핑했기에
한방에 교체 가능

src/App.vue

```
// 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) {
      states.todoList.push({ ...response.data, done: false });
      successCallback();
    } else {
      alert('Todo 추가 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

서버에서 추가했는데
왜 인메모리 배열에 또 추가하지?

목록을 얻어오는 통신 횟수를 줄이기 위해서
속도면에서 장점이 있지만

하지만 사용자가 여러명일 때
동기화 문제가 생길 수 있다.

src/App.vue

```
// 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 = states.todoList.findIndex((todo) => todo.id === id);
      states.todoList[index] = payload;
      successCallback();
    } else {
      alert('Todo 변경 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

src/App.vue

```
//기존 TodoItem을 삭제합니다.
const deleteTodo = async (id) => {
  try {
    const response = await axios.delete(BASEURI + `/${id}`);
    console.log(response.status, response.data);
    if (response.status === 200) {
      let index = states.todoList.findIndex((todo) => todo.id === id);
      states.todoList.splice(index, 1);
    } else {
      alert('Todo 삭제 실패');
    }
  } catch (error) {
    alert('에러발생 :' + error);
  }
};
```

src/App.vue

```
//기존 TodoItem의 완료여부(done) 값을 토글합니다.
const toggleDone = async (id) => {
  try {
    let todo = states.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);
  }
};

provide('todoList', computed(()=>states.todoList));
provide('actions', { addTodo, deleteTodo, toggleDone, updateTodo, fetchTodoList })

fetchTodoList();
</script>
```

src/pages/AddTodo.vue 변경

```
…

<script setup>
import { inject, reactive } from 'vue';
import { useRouter } from 'vue-router';

const router = useRouter();
const { addTodo } = inject('actions');
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>
```

비동기

src/pages/EditTodo.vue 변경

```
…
<script setup>
import { inject, reactive } from 'vue';
import { useRouter, useRoute } from 'vue-router';

const todoList = inject('todoList');
const { updateTodo } = inject('actions');
const router = useRouter();
const currentRoute = useRoute();

const matchedTodoItem = todoList.value.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>
```

3 2단계 axios 적용

목록 가져오는 것을 앱에서 해야하느냐
목록 자체에서 해야하느냐.

여기서는 서버와의 통신량을 줄이기 위해
앱에서 했다! 수정 될때 서버에게 요구를 보내고
로컬에 있는 것들도 수정함.

src/pages/TodoList.vue 변경

```
<template>
    <div class="row">
        <div class="col p-3">
            <router-link class="btn btn-primary" to="/todos/add">
            할일 추가
            </router-link>
            <button class="btn btn-primary ms-1" @click="fetchTodoList">
            새로 고침
            </button>
        </div>
    </div>
    <div class="row">
        <div class="col p-3">
            <ul class="list-group">
                <TodoItem v-for="todoItem in todoList" :key="todoItem.id" :todoItem="todoItem" />
            </ul>
        </div>
    </div>
</template>
```

목록 자체에서 가져온다면
매번 다시 받아오는 오버헤드가
있지만
provide/inject와 부가적인 것을
할 필요가 없어짐.

2번째 방법이 더 일반적인 방법이긴해

src/pages/TodoList.vue 변경

```
<script setup>
import {inject} from 'vue';
import TodoItem from '@/components/TodoItem.vue'

const todoList = inject('todoList');
const { fetchTodoList } = inject('actions');
</script>
```

로딩 아이콘

모듈화되어있음 컴포넌트가 있다

- npm install vue-csspin

src/components/Loading.vue

```
<template>
    <VueCsspin message="Loading" spin-style="cp-flip" />
</template>

<script setup>
import { VueCsspin } from 'vue-csspin'
import 'vue-csspin/dist/vue-csspin.css'
</script>
```

src/App.vue 변경

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

<script setup>
import { reactive, provide, computed } from 'vue'
import Header from '@/components/Header.vue'
import Loading from '@/components/Loading.vue'
import axios from 'axios';
…
```

```
const states=reactive({todoList: [ ], isLoading:false });
```

src/App.vue 변경

```
//TodoList 목록을 조회합니다.
const fetchTodoList = async () => {
  states.isLoading = true;
  …
  states.isLoading = false;
}

// 새로운 TodoItem을 추가합니다.
const addTodo = async ({ todo, desc }, successCallback) => {
  states.isLoading = true;
  …
  states.isLoading = false;
}

// 기존 TodoItem을 변경합니다.
const updateTodo = async ({ id, todo, desc, done }, successCallback) => {
  states.isLoading = true;
  …
  states.isLoading = false;
}
```

이거 자동으로 할 일반적인 방법 없을까?? 고민해봐 decorate패턴을 적용하면 해결 가능

App모듈에 있는 이유
외부로 뺄 수 없는 이유

states 변수에 접근해야해 하는데 클로저이기 때문에

외부 모듈로 빼면 states에 접근 안돼 ㅜㅜ

src/App.vue 변경

```
//기존 TodoItem을 삭제합니다.
const deleteTodo = async (id) => {
  states.isLoading = true;
  …
  states.isLoading = false;
}

//기존 TodoItem의 완료여부(done) 값을 토글합니다.
const toggleDone = async (id) => {
  states.isLoading = true;
  …
  states.isLoading = false;
}

provide('todoList', computed(()=>states.todoList));
provide('actions', { addTodo, deleteTodo, toggleDone, updateTodo, fetchTodoList })

fetchTodoList();
</script>
```