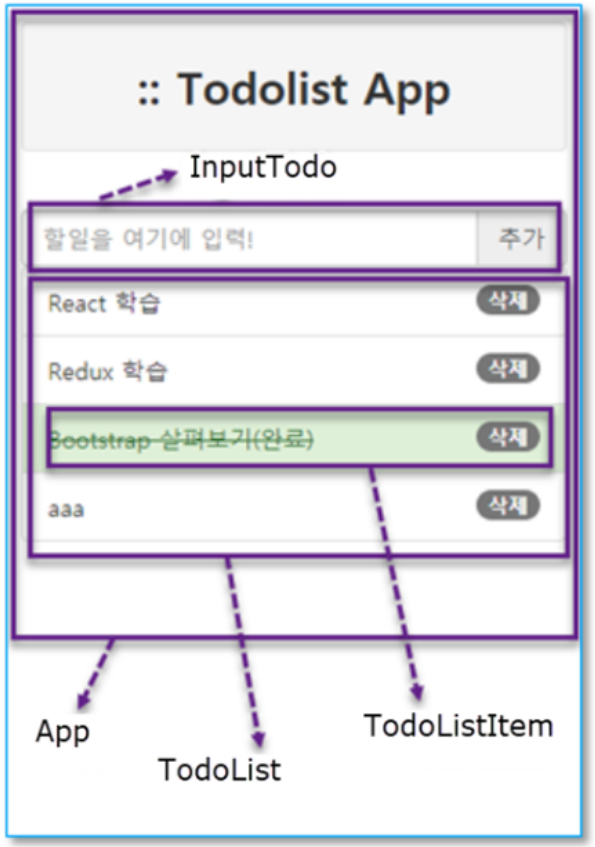
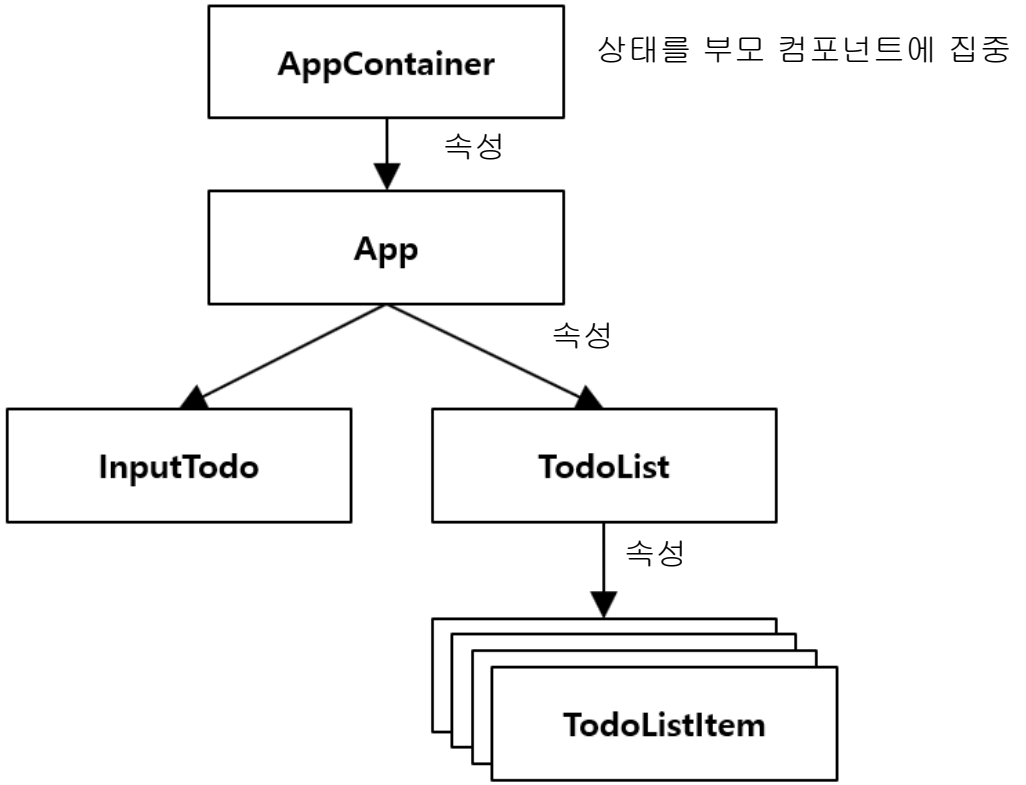
- 리팩토링 1,2

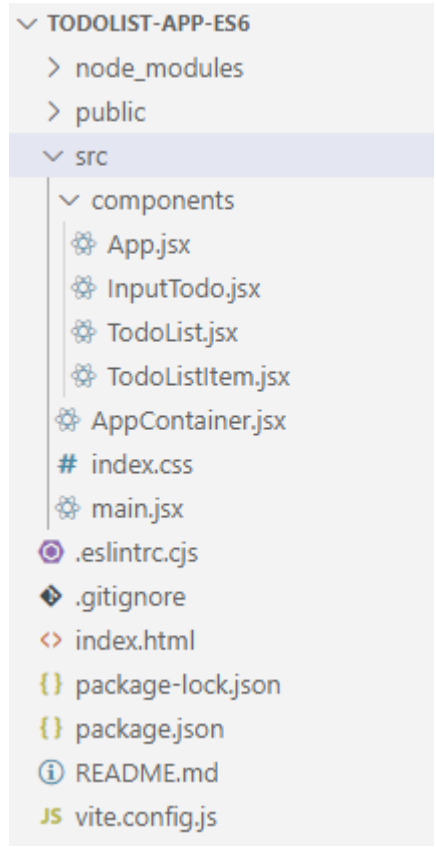
1. TodoList앱 리팩토링1

❖ES6 기반의 TodoList 리액트 앱을 Typescript 기반의 코드로 변경함

- 기존 TodoList 앱 : todolist-app-es6
- 실행 여부 확인







1.1 기존 프로젝트 변경

❖기존 ES6 프로젝트를 Typescript 프로젝트로 변경하려면? (Vite 기반 예시)

- 필요 패키지 추가
  - npm install -D @typescript-eslint/eslint-plugin @typescript-eslint/parser typescript
- tsconfig.json, tsconfig.node.json 추가
  - tsconfig.node.json : Node.js 빌드 환경에 대한 typescript 설정
  - tsconfig.json : React 앱 빌드 환경에 대한 typescript 설정

** tsconfig.node.json

```
{
  "compilerOptions": {
    "composite": true,
    "skipLibCheck": true,
    "module": "ESNext",
    "moduleResolution": "bundler",
    "allowSyntheticDefaultImports": true
  },
  "include": ["vite.config.ts"]
}
```

** tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,

    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",
```

** tsconfig.json(이어서)

```
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["src"],
  "references": [{ "path":
"./tsconfig.node.json" }]
}
```

1.1 기존 프로젝트 변경

- vite.config.js 파일의 확장자를 .ts로 변경
- 모든 자바스크립트 코드의 확장자를 다음과 같이 변경
  - .jsx : .tsx로 변경
  - .js : .ts로 변경
- src/main.tsx 파일에서 다음 부분을 찾아 변경 : ! 추가

```
ReactDOM.createRoot(document.getElementById('root')!).render( ……)
```

- index.html 파일에서 script 태그의 경로에서 main.jsx 경로를 main.tsx로 변경
- 각 컴포넌트 내부를 Typescript에 맞게 변경
  - 이 내용은 다음 페이지부터의 내용을 참조

1.2 프로젝트 생성

❖ 새로운 프로젝트 생성하여 작성하려면...
- npm init vite todolist-app-ts -- --template react-ts
- cd todolist-app-ts
- npm install immer prop-types bootstrap

❖ 주요 변경 포인트
- 파일명 변경
  - 컴포넌트 파일 확장자 : .tsx
  - 모듈 파일 확장자 : .ts
- 타입 추가 : interface 또는 type
  - 데이터의 타입이 필요한 경우
  - 데이터가 생성되는 곳 또는 참조되는 모듈에서 type 추가
  - 만일 다른 모듈에서도 같은 type이 사용된다면 export 할 것
- useState : Generic으로 타입을 지정하여 생성
- 컴포넌트 속성 : interface 또는 type
- EventHandler 제네릭 타입

1.3 파일 이동, 삭제

❖불필요한 파일 삭제
- App.css, App.tsx 삭제
- assets 폴더 삭제

❖index.css 파일 이동
- css는 동일한 것을 사용함.

❖다음 파일을 es6 프로젝트에서 복사한 후 확장자를 .tsx로 변경
- src/AppContainer.jsx
- src/components/App.jsx
- src/components/InputTodo.jsx
- src/components/TodoList.jsx
- src/components/TodoListItem.jsx

1.4 AppContainer 변경

❖src/AppContainer.tsx 변경

- 상태를 사용하는 컴포넌트
- 상태의 타입을 정의 : interface 또는 type
  - 다른 모듈에서 재사용하고, 확장될 수 있는 타입이라면 interface 사용
  - 해당 모듈에서만 사용하며 재사용되지 않는다면 type을 사용해도 됨
  - 정의된 타입이 다른 모듈에서 사용된다면 export 할 것
- useState<Type>() 과 같이 제네릭 사용하여 타입 지정
- 반드시 타입을 지정해야만 하는 것은 아님
  - 타입 추론을 활용해도 디버깅할 때 지장이 없다면 문제 없음

1.4 AppContainer 변경

- src/AppContainer.tsx

```
import { useState } from "react";
import App from "./components/App";
import { produce } from "immer";

export interface ITodoItem {
  no: number; todo:string; done: boolean;
}

const AppContainer = () => {
  const [todoList, setTodoList] = useState<ITodoItem[]>([
    { no: 1, todo: "React학습1", done: false },
    { no: 2, todo: "React학습2", done: false },
    { no: 3, todo: "React학습3", done: true },
    { no: 4, todo: "React학습4", done: false },
  ]);

  const addTodo = (todo:string) => {
    let newTodoList = produce(todoList, (draft) => {
      draft.push({ no: new Date().getTime(), todo: todo, done: false });
    });
    setTodoList(newTodoList);
  };
```

1.4 AppContainer 변경

- src/AppContainer.tsx(이어서)

```
  const deleteTodo = (no:number) => {
    let newTodoList = todoList.filter((item) => item.no !== no);
    setTodoList(newTodoList);
  };

  const toggleDone = (no:number) => {
    let index = todoList.findIndex((item) => item.no === no);
    let newTodoList = produce(todoList, (draft) => {
      draft[index].done = !draft[index].done;
    });
    setTodoList(newTodoList);
  };

  return (
    <App todoList={todoList} addTodo={addTodo}
      deleteTodo={deleteTodo} toggleDone={toggleDone}  />
  );
};

export default AppContainer;
```

1.5 App 변경

❖ 이 컴포넌트의 특징

- 자체적인 UI 기능 없음
- 부모로부터 속성을 받아 자식 컴포넌트에 속성으로 다시 전달함
- src/components/App.tsx

```
import TodoList from './TodoList';
import InputTodo from './InputTodo';
import { ITodoItem } from '../AppContainer';

type AppType = {
  addTodo: (todo:string)=>void;
  deleteTodo: (no:number)=>void;
  toggleDone: (no:number)=>void;
  todoList: ITodoItem[];
}

const App = ({ todoList, addTodo, deleteTodo, toggleDone }: AppType) => {
  ……(생략)
};

export default App;
```

1.6 InputTodo 변경

❖이 컴포넌트의 특징

- 이벤트 핸들러가 있음 : 핸들러 타입 지정
- 함수(메서드)를 속성으로 전달받음, 속성의 유효성 검사는 필요 없음
- src/components/InputTodo.tsx

```
import { KeyboardEvent, useState } from 'react';                    //임포트!!

type InputTodoType = {
  addTodo: (todo:string)=>void;
}
const InputTodo = ({ addTodo }: InputTodoType) => {
  const [todo, setTodo] = useState<string>("");

  const addHandler = () => {
    addTodo(todo);
    setTodo("");
  }
  // 아래 타입은 어떻게 지정할까?
  const enterInput = (e: KeyboardEvent<HTMLInputElement>) => {
    if (e.key === "Enter") {
        addHandler();
    }
  }
```

1.6 InputTodo 변경

- src/components/InputTodo.tsx (이어서)

```
  return (
    <div className="row">
      <div className="col">
        <div className="input-group">
          <input
            id="msg"
            type="text"
            className="form-control"
            name="msg"
            placeholder="할일을 여기에 입력!"
            value={todo}
            onChange={ (e)=> setTodo(e.target.value) }
            onKeyUp={enterInput}
          />
          <span className="btn btn-primary input-group-addon"
              onClick={addHandler}>추가</span>
        </div>
      </div>
    </div>
  );
};

export default InputTodo;
```

** 비교
-왜 onChange 이벤트 핸들러 함수의 인자 e에 type를 지정하지 않아도 될까?

onChange 핸들러에 지정된 함수이니까!!

1.6 InputTodo 변경

❖이벤트 핸들러 함수의 인자(e)의 타입은 어떻게 지정하는가?

- on~ 핸들러에 마우스를 가져다대보면 React.KeyboardEventHandler<...>과 같이 툴팁이 나타남
- 여기서 Handler를 빼고 인자로 사용하면 됨
  - 예시 : React.KeyboardEvent<HTMLInputElement>

```
<div className="row">
  <div className="col">
    <div className="input-group">
      <input
        id="msg"
        type="text"
        className="form-control"
        name="msg"
        placeholder="할일을 여기에 입력!"
        onKeyUp={enterInput}
      />
      <span className="btn btn-primary input-group-addon"
          onClick={addHandler}>추가</span>
    </div>
  </div>
</div>
```

```
(property) React.DOMAttributes<HTMLInputElement>.onKeyUp?:
React.KeyboardEventHandler<HTMLInputElement> | undefined
```

```
const enterInput = (e: KeyboardEvent<HTMLInputElement>) => {
  if (e.key === "Enter") {
      addHandler();
  }
}
```

1.7 TodoList 변경

❖ 이 컴포넌트의 특징

- 속성을 전달받아 자식으로 속성 전달
- src/components/TodoList.tsx

```
import { ITodoItem } from "../AppContainer";
import TodoListItem from "./TodoListItem";

type TodoListType = {
  todoList: ITodoItem[];
  deleteTodo: (no:number)=>void;
  toggleDone: (no:number)=>void;
}

const TodoList = ({ todoList, deleteTodo, toggleDone }: TodoListType) => {
  let items = todoList.map((item) => {
    return <TodoListItem key={item.no} todoItem={item}
              deleteTodo={deleteTodo} toggleDone={toggleDone} />;
  });
   ......(생략)
};

export default TodoList;
```

1.8 TodoListItem 변경

❖ 이 컴포넌트의 특징

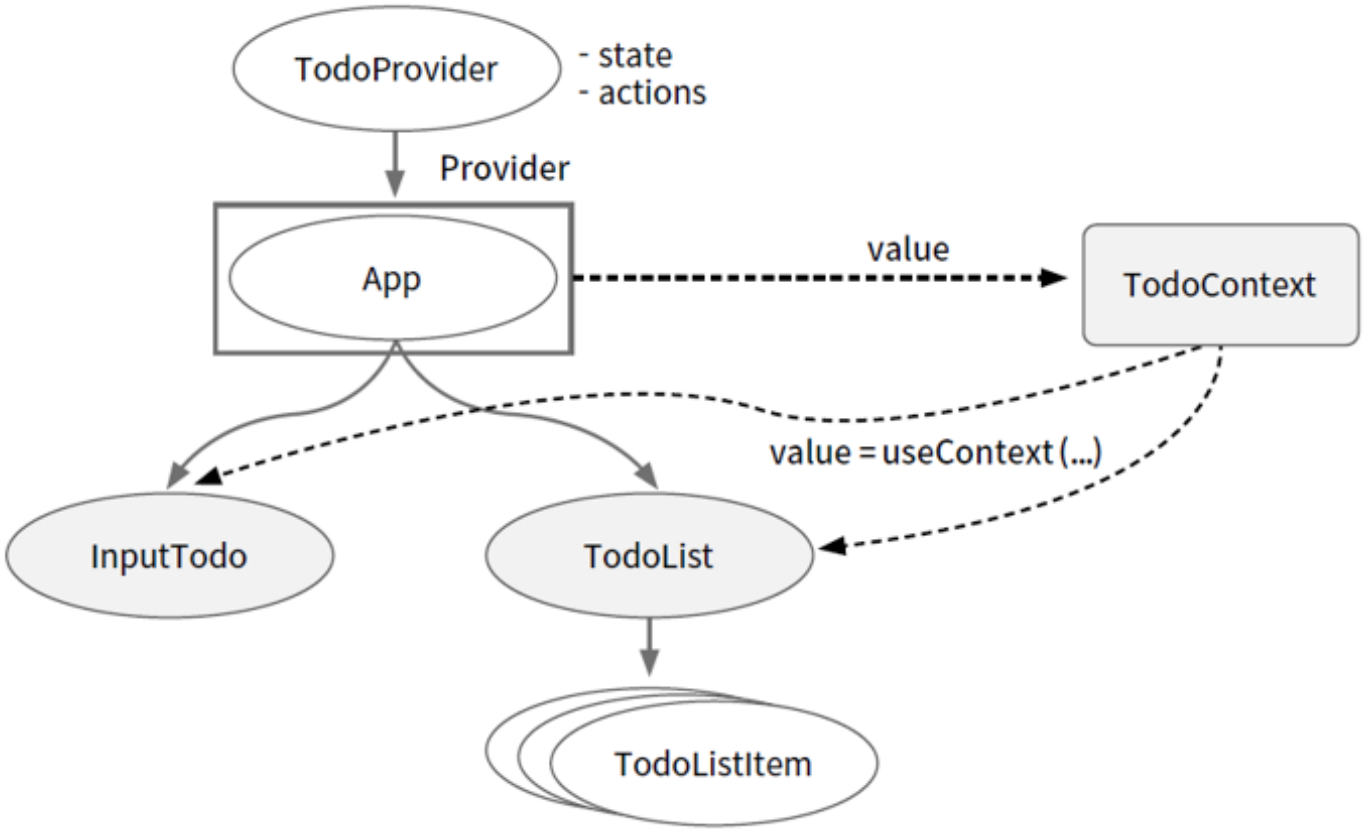
- 속성을 전달받아 렌더링하는 전형적인 표현 컴포넌트
- src/components/TodoListItem.tsx

```
import { ITodoItem } from "../AppContainer";

type TodoItemType = {
  todoItem: ITodoItem;
  deleteTodo: (no:number)=>void;
  toggleDone: (no:number)=>void;
}

const TodoListItem = ({ todoItem, deleteTodo, toggleDone }: TodoItemType) => {
  ……(생략)
};

export default TodoListItem;
```

2. TodoList앱 리팩토링2

❖기존 앱

- todolist-app-context-es6 프로젝트
- Context API를 사용하는 TodoList 예제



2.1 새로운 프로젝트 생성

❖프로젝트 생성
- npm init vite todolist-app-context-ts -- --template react-ts

❖관련 패키지 설치
- npm install bootstrap immer

❖불필요한 파일 삭제
- src/assets 폴더 삭제
- src/App.tsx, src/App.css 삭제

❖src/index.css 변경
- ES6 프로젝트의 index.css 내용으로 변경

2.2 컴포넌트 파일 복사, main 변경

❖ 다음 파일 복사후 확장자를 .tsx로 변경

- src/TodoContext.jsx
- src/components/App.jsx
- src/components/InputTodo.jsx
- src/components/TodoList.jsx
- src/components/TodoListItem.jsx

❖ src/main.tsx 복사 후 변경

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { TodoProvider } from './TodoContext';
import App from './components/App'
import "bootstrap/dist/css/bootstrap.css";
import "./index.css";

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <TodoProvider>
      <App />
    </TodoProvider>
  </React.StrictMode>,
)
```

2.3 TodoContext 변경

❖이 컴포넌트의 특징

- Context 컴포넌트 생성
- Context Provider를 이용해 value 객체를 설정하기 위한 TodoProvider Wrapper 컴포넌트 생성
- Type 설정에 주의할 것
- src/TodoContext.tsx

```
import React, { useState } from "react";
import { produce } from "immer";

//TodoListItem  한건의 타입 선언
export interface TodoItemType {
  no: number;
  todo: string;
  done: boolean;
}
```

2.3 TodoContext 변경

- src/TodoContext.tsx (이어서)

```
//Context Provider의 value 값에 대한 타입 선언
//상태 데이터와 상태 변경 메서드를 구분하여 state, actions 속성으로 구분
export interface TodoContextValueType {
  state: { todoList: TodoItemType[] };
  actions: {
    addTodo: (todo: string) => void;
    deleteTodo: (no: number) => void;
    toggleDone: (no: number) => void;
  };
}

//TodoProvider의 자식 컴포넌트 속성 type 선언
type PropsType = {
  children: JSX.Element | JSX.Element[];
};

//초기값 null을 부여할 수 있어야 하므로 null을 union
const TodoContext = React.createContext<TodoContextValueType | null>(null);
```

2.3 TodoContext 변경

- src/TodoContext.tsx (이어서)

```
/** ## 다음과 같이 사용함
 * <TodoProvider>
 *    <App />      ---> children
 * </TodoProvider>
 ** ## 다음과 같이 렌더링됨
 * <Context.Provider value={values}>
 *    {props.children}
 * </Context.Provider>
 */
export const TodoProvider = (props: PropsType) => {
  const [todoList, setTodoList] = useState<TodoItemType[]>([
    { no: 1, todo: "React학습1", done: false },
    { no: 2, todo: "React학습2", done: false },
    { no: 3, todo: "React학습3", done: true },
    { no: 4, todo: "React학습4", done: false },
  ]);

  const addTodo = (todo:string) => {
      ...(생략)
  };
```

2.3 TodoContext 변경

- src/TodoContext.tsx (이어서)

```
  const deleteTodo = (no:number) => {
      ...(생략)
  };

  const toggleDone = (no:number) => {
      ...(생략)
  };

  //상태와 상태 변경 메서드를 state, actions 속성으로 구분하여 설정
  const values: TodoContextValueType = {
    state: { todoList },
    actions: { addTodo, deleteTodo, toggleDone },
  };

  return (
    <TodoContext.Provider value={values}>
        {props.children}
    </TodoContext.Provider>
  );
};

export default TodoContext;
```

2.4 컴포넌트 작성

❖src/components/InputTodo.tsx

```
import { ChangeEvent, KeyboardEvent, useContext, useState } from "react";
import TodoContext from "../TodoContext";

const InputTodo = () => {
  const value = useContext(TodoContext);
  const [todo, setTodo] = useState("");

  //value의 type이 TodoContextValueType | null임
  //value가 null일수도 있기 때문에 value?.actions.addTodo(todo)와 같이 ? 사용
  const addHandler = () => {
    value?.actions.addTodo(todo);
    setTodo("");
  };
  //이벤트 아규먼트 타입 주의
  const enterInput = (e: KeyboardEvent<HTMLInputElement>) => {
    if (e.key === "Enter") {
      addHandler();
    }
  };
```

2.4 컴포넌트 작성

❖src/components/InputTodo.tsx(이어서)

```
//이벤트 아규먼트 타입 주의
 const changeTodo = (e: ChangeEvent<HTMLInputElement>) => {
   setTodo(e.target.value);
 };

 return (
   <div className="row">
     <div className="col">
       <div className="input-group">
         <input id="msg" type="text" className="form-control" name="msg"
           placeholder="할일을 여기에 입력!" value={todo}
           onChange={changeTodo} onKeyUp={enterInput} />
         <span className="btn btn-primary input-group-addon"
            onClick={addHandler}>
           추가
         </span>
       </div>
     </div>
   </div>
 );
};

export default InputTodo;
```

2.4 컴포넌트 작성

❖src/components/TodoList.tsx

```
import { useContext } from "react";
import TodoListItem from "./TodoListItem";
import TodoContext from "../TodoContext";

const TodoList = () => {
  const value = useContext(TodoContext);

  let items = value?.state.todoList.map((item) => {
    return <TodoListItem key={item.no} todoItem={item}
            deleteTodo={value?.actions.deleteTodo}
            toggleDone={value?.actions.toggleDone} />;
  });

  return (
    <div className="row">
      {" "}
      <div className="col">
        <ul className="list-group">{items}</ul>
      </div>
    </div>
  );
};

export default TodoList;
```

2.4 컴포넌트 작성

❖src/components/TodoListItem.tsx

```
import { TodoItemType } from "../TodoContext";

type PropsType = {  todoItem: TodoItemType;  deleteTodo: (no:number)=>void;  toggleDone: (no:number)=>void;   }

const TodoListItem = ({ todoItem, deleteTodo, toggleDone }: PropsType) => {
  let itemClassName = "list-group-item";
  if (todoItem.done) itemClassName += " list-group-item-success";

  return (
    <li className={itemClassName}>
      <span className={todoItem.done ? "todo-done pointer" : "pointer"} onClick={() => toggleDone(todoItem.no)}>
        {todoItem.todo}
        {todoItem.done ? " (완료)" : ""}
      </span>
      <span className="float-end badge bg-secondary pointer" onClick={() => deleteTodo(todoItem.no)}>
        삭제
      </span>
    </li>
  );
};

export default TodoListItem;
```

Q&A