Web Programming

React programming

TODO list

Todo list

- npx create-react-app mashup-todolist
- cd mashup-todolist
- yarn add react-icons styled-components
- yarn add styled-components

TODO list

만들어야 할 컴포넌트 확인하기

- TodoTemplate
  - 투두리스트의 레이아웃을 설정하는 컴포넌트. 페이지의 중앙에 그림자가 적용된 흰색 박스를 보여준다.
- TodoHead
  - 오늘의 날짜와 요일을 보여주고, 앞으로 해야 할 일이 몇개 남았는지 보여준다.
- TodoList
  - 할 일에 대한 정보가 들어있는 todos 배열을 내장함수 map을 사용하여 여러 개의 TodoItem 컴포넌트를 렌더링해준다.
- TodoItem
  - 각 할 일에 대한 정보를 렌더링해주는 컴포넌트. 좌측에 있는 원을 누르면 할 일의 완료 여부를 toggle 할 수 있다. 할 일이 완료됐을 땐 좌측에 체크가 나타나고 텍스트의 색상이 연해진다. 그리고, 마우스를 올리면 휴지통 아이콘이 나타나고 이를 누르면 항목이 삭제된다.
- TodoCreate
  - 새로운 할 일을 등록할 수 있게 해주는 컴포넌트. TodoTemplate 의 하단부에 초록색 원 버튼을 렌더링해주고, 이를 클릭하면 할 일을 입력 할 수 있는 폼이 나타난다. 버튼을 다시 누르면 폼이 사라진다.

TODO list

페이지에 회색 배경색상 적용

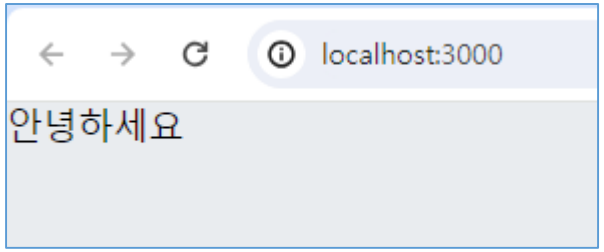
- styled-components에서 특정 컴포넌트를 만들어서 스타일링 하는게 아니라 글로벌 스타일을 추가하고 싶을 땐 createGlobalStyle이라는 것을 사용한다. 이 함수를 사용하면 컴포넌트가 만들어지는데, 이 컴포넌트를 렌더링하면 된다.

```
import React from 'react';
import { createGlobalStyle } from 'styled-components';

const GlobalStyle = createGlobalStyle`
  body {
    background: #e9ecef;
  }
`;

function App() {
  return (
    <>
      <GlobalStyle />
      <div>안녕하세요</div>
    </>
  );
}

export default App;
```



TODO list

TodoTemplate 만들기

- TodoTemplate 컴포넌트를 만들어서 중앙에 정렬된 흰색 박스를 보여준다. src 디렉터리에 components 디렉터리를 만들고, 그 안에 TodoTemplate.js 를 만든다. 앞으로 만들 컴포넌트들은 모두 components 디렉터리에 만들도록 하겠다.

```
import React from 'react';
import styled from 'styled-components';

const TodoTemplateBlock = styled.div`
  width: 512px;
  height: 768px;

  position: relative; /* 추후 박스 하단에 추가 버튼을 위치시키기 위한 설정 */
  background: white;
  border-radius: 16px;
  box-shadow: 0 0 8px 0 rgba(0, 0, 0, 0.04);
  margin: 0 auto; /* 페이지 중앙에 나타나도록 설정 */

  margin-top: 96px;
  margin-bottom: 32px;
  display: flex;
  flex-direction: column;
`;

function TodoTemplate({ children }) {
  return <TodoTemplateBlock>{children}</TodoTemplateBlock>;
}

export default TodoTemplate;
```

TODO list

TodoTemplate 만들기

- 컴포넌트를 App에서 렌더링

```
import React from 'react';
import { createGlobalStyle } from 'styled-components';
import TodoTemplate from './components/TodoTemplate';

const GlobalStyle = createGlobalStyle`
  body {
    background: #e9ecef;
  }
`;

function App() {
  return (
    <>
      <GlobalStyle />
      <TodoTemplate>안녕하세요</TodoTemplate>
    </>
  );
}

export default App;
```

안녕하세요

TODO list

TodoHead 만들기

- 오늘의 날짜, 요일, 그리고 남은 할 일 개수를 보여준다.

```
import React from 'react';
import styled from 'styled-components';

const TodoHeadBlock = styled.div`
  padding-top: 48px;
  padding-left: 32px;
  padding-right: 32px;
  padding-bottom: 24px;
  border-bottom: 1px solid #e9ecef;
  h1 {
    margin: 0;
    font-size: 36px;
    color: #343a40;
  }
  .day {
    margin-top: 4px;
    color: #868e96;
    font-size: 21px;
  }
```

```
  .tasks-left {
    color: #20c997;
    font-size: 18px;
    margin-top: 40px;
    font-weight: bold;
  }
`;

function TodoHead() {
  return (
    <TodoHeadBlock>
      <h1>2024년 5월 17일</h1>
      <div className="day">금요일</div>
      <div className="tasks-left">할 일 2개 남음</div>
    </TodoHeadBlock>
  );
}

export default TodoHead;
```

TODO list

TodoHead 만들기

```
import React from 'react';
import { createGlobalStyle } from 'styled-components';
import TodoTemplate from './components/TodoTemplate';
import TodoHead from './components/TodoHead';

const GlobalStyle = createGlobalStyle`
  body {
    background: #e9ecef;
  }
`;

function App() {
  return (
    <>
      <GlobalStyle />
      <TodoTemplate>
        <TodoHead />
      </TodoTemplate>
    </>
  );
}

export default App;
```

TODO list

TodoList 만들기

- 여러 개의 할 일 항목을 보여주게 될 TodoList.
- 지금은 특별한 내용을 보여주지 않고, 사이즈에 관련한 설정만 해줌.
- flex: 1 스타일을 설정함으로써 자신이 차지 할 수 있는 영역을 꽉 채우도록 설정을 했는데, 이게 잘 작동했는지 확인하기 위하여 임시적으로 회색 배경을 설정.

```
import React from 'react';
import styled from 'styled-components';

const TodoListBlock = styled.div`
  flex: 1;
  padding: 20px 32px;
  padding-bottom: 48px;
  overflow-y: auto;
  background: gray; /* 사이즈 조정이 잘 되고 있는지 확인하기 위한 임시 스타일 */
`;

function TodoList() {
  return <TodoListBlock>TodoList</TodoListBlock>;
}

export default TodoList;
```

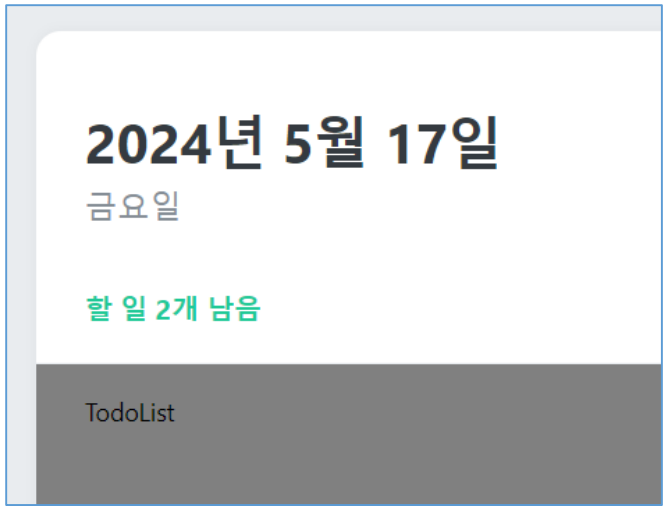
TODO list

TodoList 만들기

```
import React from 'react';
import { createGlobalStyle } from 'styled-components';
import TodoTemplate from './components/TodoTemplate';
import TodoHead from './components/TodoHead';
import TodoList from './components/TodoList';

const GlobalStyle = createGlobalStyle`
  body {
    background: #e9ecef;
  }
`;

function App() {
  return (
    <>
      <GlobalStyle />
      <TodoTemplate>
        <TodoHead />
        <TodoList />
      </TodoTemplate>
    </>
  );
}
export default App;
```



TODO list

TodoItem 만들기

- 각 할 일 항목들을 보여주는 TodoItem 컴포넌트를 만든다. 이 컴포넌트에서는 react-icons에서 MdDone 과 MdDelete 아이콘을 사용한다.
- 여기서 사용된 기능은 Component Selector 라는 기능이다. 이 스타일은 TodoItemBlock 위에 커서가 있을 때, Remove 컴포넌트를 보여주라는 의미를 가지고 있다.

```
import React from 'react';
import styled, { css } from 'styled-components';
import { MdDone, MdDelete } from 'react-icons/md';

const Remove = styled.div`
  display: flex;
  align-items: center;
  justify-content: center;
  color: #dee2e6;
  font-size: 24px;
  cursor: pointer;
  &:hover {
    color: #ff6b6b;
  }
  display: none;
`;
```

```
const TodoItemBlock = styled.div`
  display: flex;
  align-items: center;
  padding-top: 12px;
  padding-bottom: 12px;
  &:hover {
    ${Remove} {
      display: initial;
    }
  }
`;
```

TODO list

TodoItem 만들기

```
const CheckCircle = styled.div`
  width: 32px;
  height: 32px;
  border-radius: 16px;
  border: 1px solid #ced4da;
  font-size: 24px;
  display: flex;
  align-items: center;
  justify-content: center;
  margin-right: 20px;
  cursor: pointer;
  ${props =>
    props.done &&
    css`
      border: 1px solid #38d9a9;
      color: #38d9a9;
    `}
`;
```

```
const Text = styled.div`
  flex: 1;
  font-size: 21px;
  color: #495057;
  ${props =>
    props.done &&
    css`
      color: #ced4da;
    `}
`;

function TodoItem({ id, done, text }) {
  return (
    <TodoItemBlock>
      <CheckCircle done={done}>{done && <MdDone />}</CheckCircle>
      <Text done={done}>{text}</Text>
      <Remove>
        <MdDelete />
      </Remove>
    </TodoItemBlock>
  );
}

export default TodoItem;
```

TODO list

TodoItem 만들기

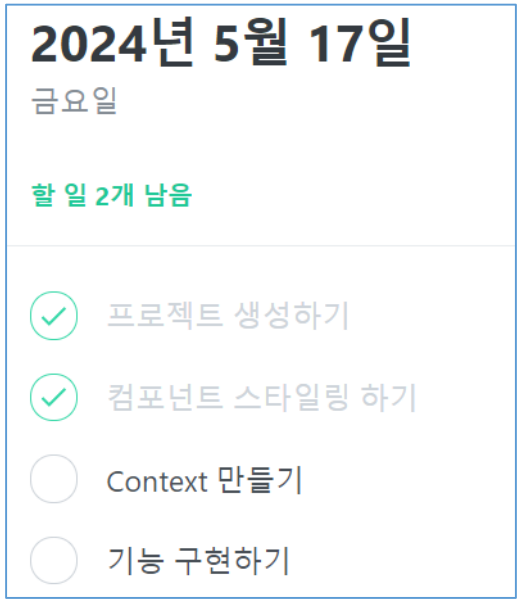
- 컴포넌트를 TodoList에서 렌더링

```
import React from 'react';
import styled from 'styled-components';
import TodoItem from './TodoItem';

const TodoListBlock = styled.div`
  flex: 1;
  padding: 20px 32px;
  padding-bottom: 48px;
  overflow-y: auto;
`;

function TodoList() {
  return (
    <TodoListBlock>
      <TodoItem text="프로젝트 생성하기" done={true} />
      <TodoItem text="컴포넌트 스타일링 하기" done={true} />
      <TodoItem text="Context 만들기" done={false} />
      <TodoItem text="기능 구현하기" done={false} />
    </TodoListBlock>
  );
}

export default TodoList;
```



TODO list

TodoCreate 만들기

- 이번에는 새로운 항목을 등록 할 수 있는 컴포넌트 만든다.
- 이 컴포넌트에서는 react-icons의 MdAdd를 사용합니다.
- 이 컴포넌트에서는 useState를 사용하여 토글할 수 있는 open 값을 관리하며, 이 값이 true일 때에는 아이콘을 45도 돌려서 X 모양이 보여지게 한 후, 버튼 색상을 빨간색으로 바꿔준다. 그리고, 할 일을 입력 할 수 있는 폼도 보여준다.

```
import React, { useState } from 'react';
import styled, { css } from 'styled-components';
import { MdAdd } from 'react-icons/md';

const CircleButton = styled.button`
  background: #38d9a9;
  &:hover {
    background: #63e6be;
  }
  &:active {
    background: #20c997;
  }
```

```
  z-index: 5;
  cursor: pointer;
  width: 80px;
  height: 80px;
  display: block;
  align-items: center;
  justify-content: center;
  font-size: 60px;
  position: absolute;
  left: 50%;
  bottom: 0px;
```

```
  transform: translate(-50%, 50%);
  color: white;
  border-radius: 50%;
  border: none;
  outline: none;
  display: flex;
  align-items: center;
  justify-content: center;
```

TODO list

TodoCreate 만들기

```
  transition: 0.125s all ease-in;
  ${props =>
    props.open &&
    css`
      background: #ff6b6b;
      &:hover {
        background: #ff8787;
      }
      &:active {
        background: #fa5252;
      }
      transform: translate(-50%, 50%) rotate(45deg);
    `}
`;

const InsertFormPositioner = styled.div`
  width: 100%;
  bottom: 0;
  left: 0;
  position: absolute;
`;
```

```
const InsertForm = styled.form`
  background: #f8f9fa;
  padding-left: 32px;
  padding-top: 32px;
  padding-right: 32px;
  padding-bottom: 72px;

  border-bottom-left-radius: 16px;
  border-bottom-right-radius: 16px;
  border-top: 1px solid #e9ecef;
`;

const Input = styled.input`
  padding: 12px;
  border-radius: 4px;
  border: 1px solid #dee2e6;
  width: 100%;
  outline: none;
  font-size: 18px;
  box-sizing: border-box;
`;
```

TODO list

TodoCreate 만들기

```
function TodoCreate() {
  const [open, setOpen] = useState(false);

  const onToggle = () => setOpen(!open);

  return (
    <>
      {open && (
        <InsertFormPositioner>
          <InsertForm>
            <Input autoFocus placeholder="할 일을 입력 후, Enter 를 누르세요" />
          </InsertForm>
        </InsertFormPositioner>
      )}
      <CircleButton onClick={onToggle} open={open}>
        <MdAdd />
      </CircleButton>
    </>
  );
}

export default TodoCreate;
```

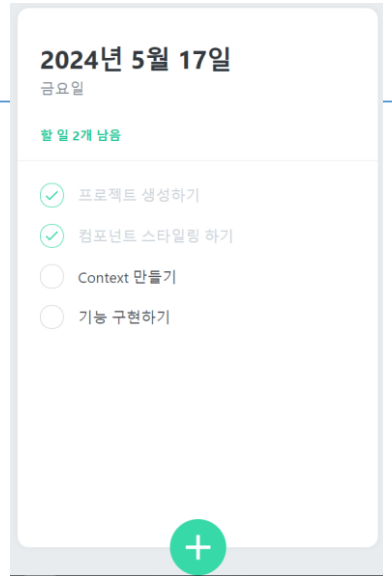
TODO list

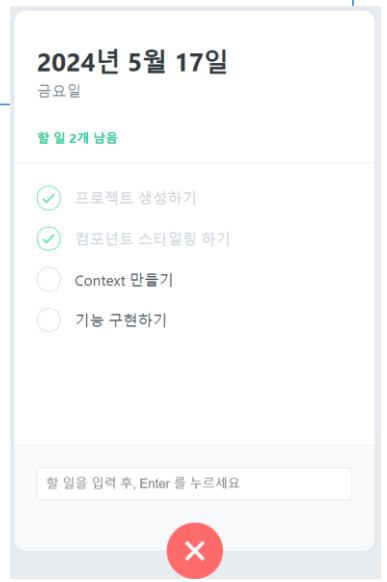
TodoCreate 만들기

- 컴포넌트를 App에서 렌더링

```
import React from 'react';
import { createGlobalStyle } from 'styled-components';
import TodoTemplate from './components/TodoTemplate';
import TodoHead from './components/TodoHead';
import TodoList from './components/TodoList';
import TodoCreate from './components/TodoCreate';

const GlobalStyle = createGlobalStyle`
  body {
    background: #e9ecef;
  }
`;
```





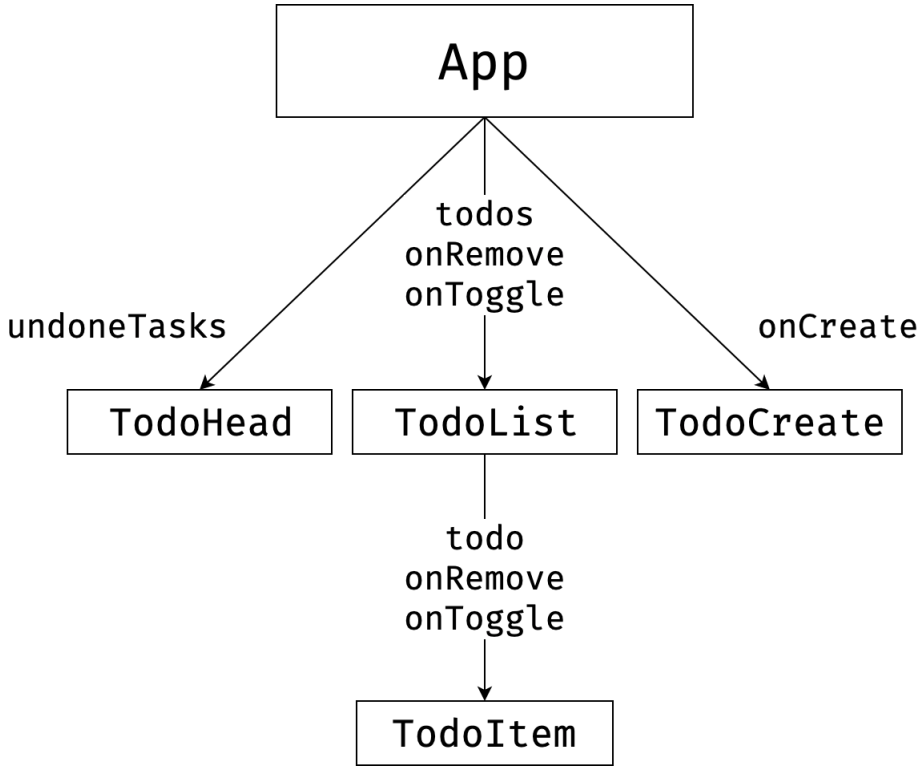

```
function App() {
  return (
    <>
      <GlobalStyle />
      <TodoTemplate>
        <TodoHead />
        <TodoList />
        <TodoCreate />
      </TodoTemplate>
    </>
  );
}

export default App;
```

TODO list

Context API 를 활용한 상태 관리

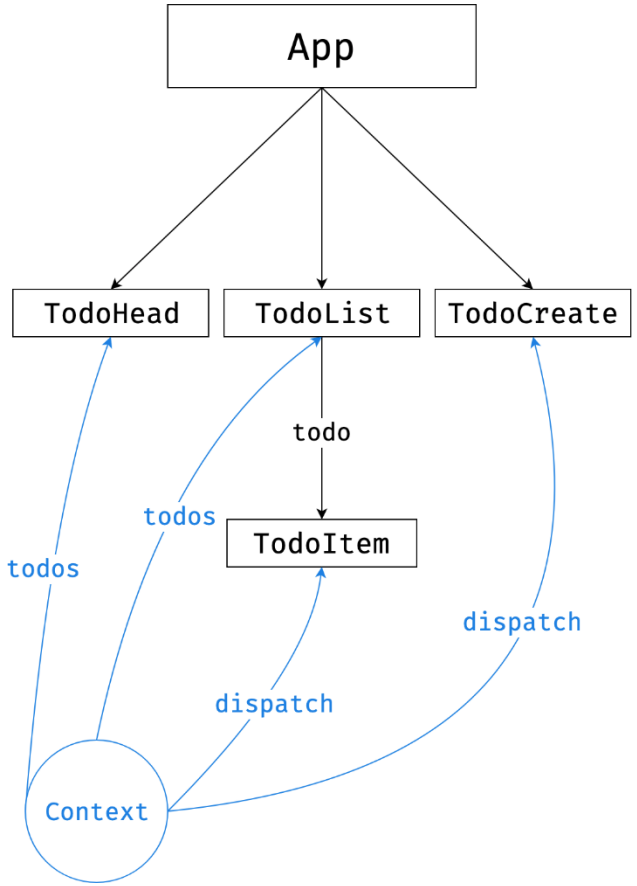
- 만약 상태 관리를 한다면 다음과 같은 구조로 구현할 수 있다.
- App에서 todos 상태와, onToggle, onRemove, onCreate 함수를 지니고 있게 하고, 해당 값들을 props를 사용해서 자식 컴포넌트들에게 전달해주는 방식으로 구현 할 수 있다.
- 이렇게 구현하는 것도 큰 문제는 없다. 이 프로젝트는 정말 작고 단순하기 때문.



TODO list

Context API 를 활용한 상태 관리

- 프로젝트의 규모가 커지게 된다면 최상위 컴포넌트인 App 에서 모든 상태 관리를 하기엔 App 컴포넌트의 코드가 너무 복잡해질 수도 있고, props 를 전달해줘야 하는 컴포넌트가 너무 깊숙히 있을 수도 있다. 만약 Context API를 활용한다면 다음과 같이 구현 할 수 있다.



TODO list

리듀서 만들기

- **src 디렉터리에 TodoContext.js 파일을 생성**하고, 그 안에 useReducer를 사용하여 상태를 관리하는 TodoProvider라는 컴포넌트를 만든다.

```
import { useReducer } from 'react';

const initialTodos = [
  {
    id: 1,
    text: '프로젝트 생성하기',
    done: true
  },
  {
    id: 2,
    text: '컴포넌트 스타일링하기',
    done: true
  },
  {
    id: 3,
    text: 'Context 만들기',
    done: false
  },
  {
    id: 4,
    text: '기능 구현하기',
    done: false
  }
];
```

src/TodoContext.js

TODO list

리듀서 만들기

```
function todoReducer(state, action) {
  switch (action.type) {
    case 'CREATE':
      return state.concat(action.todo);
    case 'TOGGLE':
      return state.map(todo =>
        todo.id === action.id ? { ...todo, done: !todo.done } : todo
      );
    case 'REMOVE':
      return state.filter(todo => todo.id !== action.id);
    default:
      throw new Error(`Unhandled action type: ${action.type}`);
  }
}
export function TodoProvider({ children }) {
  const [state, dispatch] = useReducer(todoReducer, initialTodos);
  return children;
}
```

TODO list

Context 만들기

- state 와 dispatch 를 Context 통하여 다른 컴포넌트에서 바로 사용 할 수 있게 한다.
- 하나의 Context 를 만들어서 state 와 dispatch 를 함께 넣어주는 대신에, 두개의 Context를 만들어서 따로 넣어줄 것이다. 이렇게 하면 dispatch만 필요한 컴포넌트에서 불필요한 렌더링을 방지할 수 있다. 추가적으로, 사용하게 되는 과정에서 더욱 편리하다.

```
import React, { useReducer, createContext } from 'react';

const initialTodos = [
  {
    id: 1,
    text: '프로젝트 생성하기',
    done: true
  },
  {
    id: 2,
    text: '컴포넌트 스타일링하기',
    done: true
  },
```

```
  {
    id: 3,
    text: 'Context 만들기',
    done: false
  },
  {
    id: 4,
    text: '기능 구현하기',
    done: false
  }
];
```

TODO list

Context 만들기

```
function todoReducer(state, action) {
  switch (action.type) {
    case 'CREATE':
      return state.concat(action.todo);
    case 'TOGGLE':
      return state.map(todo =>
        todo.id === action.id ? { ...todo, done: !todo.done } : todo
      );
    case 'REMOVE':
      return state.filter(todo => todo.id !== action.id);
    default:
      throw new Error(`Unhandled action type: ${action.type}`);
  }
}
```

```
const TodoStateContext = createContext();
const TodoDispatchContext = createContext();

export function TodoProvider({ children }) {
  const [state, dispatch] = useReducer(todoReducer, initialTodos);
  return (
    <TodoStateContext.Provider value={state}>
      <TodoDispatchContext.Provider value={dispatch}>
        {children}
      </TodoDispatchContext.Provider>
    </TodoStateContext.Provider>
  );
}
```

TODO list

Context 만들기

- Context에서 사용 할 값을 지정 할 때에는 앞에서와 같이 Provider 컴포넌트를 렌더링 하고 value 를 설정해주면 된다. 그리고, props로 받아온 children 값을 내부에 렌더링해주면 된다.
- 이렇게 하면 다른 컴포넌트에서 state 나 dispatch를 사용하고 싶을 때 다음과 같이 할 수 있다.

```
import React, { useContext } from 'react';
import { TodoStateContext, TodoDispatchContext } from '../TodoContext';

function Sample() {
  const state = useContext(TodoStateContext);
  const dispatch = useContext(TodoDispatchContext);
  return <div>Sample</div>;
}
```

TODO list

커스텀 Hook 만들기

- 컴포넌트에서 useContext를 직접 사용하는 대신에, useContext를 사용하는 커스텀 Hook을 만들어서 내 보낸다.

```
import React, { useReducer, createContext, useContext } from 'react';

const initialTodos = [
  {
    id: 1,
    text: '프로젝트 생성하기',
    done: true
  },
  {
    id: 2,
    text: '컴포넌트 스타일링하기',
    done: true
  },
  {
    id: 3,
    text: 'Context 만들기',
    done: false
  },
  {
    id: 4,
    text: '기능 구현하기',
    done: false
  }
];

function todoReducer(state, action) {
  switch (action.type) {
    case 'CREATE':
      return state.concat(action.todo);
    case 'TOGGLE':
      return state.map(todo =>
        todo.id === action.id ? { ...todo, done: !todo.done } : todo
      );
    case 'REMOVE':
      return state.filter(todo => todo.id !== action.id);
    default:
      throw new Error(`Unhandled action type: ${action.type}`);
  }
}
```

TODO list

커스텀 Hook 만들기

```
const TodoStateContext = createContext();
const TodoDispatchContext = createContext();

export function TodoProvider({ children }) {
  const [state, dispatch] = useReducer(todoReducer, initialTodos);
  return (
    <TodoStateContext.Provider value={state}>
      <TodoDispatchContext.Provider value={dispatch}>
        {children}
      </TodoDispatchContext.Provider>
    </TodoStateContext.Provider>
  );
}

export function useTodoState() {
  return useContext(TodoStateContext);
}

export function useTodoDispatch() {
  return useContext(TodoDispatchContext);
}
```

TODO list

커스텀 Hook 만들기

- 이렇게 하면 나중에 아래와 같이 사용 할 수 있다.

```
import React from 'react';
import { useTodoState, useTodoDispatch } from '../TodoContext';

function Sample() {
  const state = useTodoState();
  const dispatch = useTodoDispatch();
  return <div>Sample</div>;
}
```

- 이렇게 하면 조금 더 편리하게 사용할 수 있다. 하지만, 취향에 따라 useContext를 컴포넌트에서 바로 사용해도 상관은 없다.

TODO list

nextId 값 관리하기

- state 를 위한 Context 와 dispatch 를 위한 Context 를 만들었는데, 추가적으로 nextId 값을 위한 Context 를 만들어주겠다. 여기서 nextId 가 의미하는 값은 새로운 항목을 추가할 때 사용할 고유 ID 이다. 이 값은, useRef를 사용하여 관리해주도록 하겠다.
- nextId 값을 위한 Context를 만들 때도 useTodoNextId 라는 커스텀 Hook을 따로 만들어주었다.

```
import React, { useReducer, createContext, useContext, useRef } from 'react';

const initialTodos = [
  {
    id: 1,
    text: '프로젝트 생성하기',
    done: true
  },
  {
    id: 2,
    text: '컴포넌트 스타일링하기',
    done: true
  },
```

```
  {
    id: 3,
    text: 'Context 만들기',
    done: false
  },
  {
    id: 4,
    text: '기능 구현하기',
    done: false
  }
];
```

TODO list

nextId 값 관리하기

```
function todoReducer(state, action) {
  switch (action.type) {
    case 'CREATE':
      return state.concat(action.todo);
    case 'TOGGLE':
      return state.map(todo =>
        todo.id === action.id ? { ...todo, done: !todo.done } : todo
      );
    case 'REMOVE':
      return state.filter(todo => todo.id !== action.id);
    default:
      throw new Error(`Unhandled action type: ${action.type}`);
  }
}

const TodoStateContext = createContext();
const TodoDispatchContext = createContext();
const TodoNextIdContext = createContext();
export function TodoProvider({ children }) {
  const [state, dispatch] = useReducer(todoReducer, initialTodos);
  const nextId = useRef(5);
```

```
  return (
    <TodoStateContext.Provider value={state}>
      <TodoDispatchContext.Provider value={dispatch}>
        <TodoNextIdContext.Provider value={nextId}>
          {children}
        </TodoNextIdContext.Provider>
      </TodoDispatchContext.Provider>
    </TodoStateContext.Provider>
  );
}

export function useTodoState() {
  return useContext(TodoStateContext);
}

export function useTodoDispatch() {
  return useContext(TodoDispatchContext);
}

export function useTodoNextId() {
  return useContext(TodoNextIdContext);
}
```

TODO list

커스텀 Hook 에서 에러 처리

- useTodoState, useTodoDispatch, useTodoNextId Hook을 사용하려면, 해당 컴포넌트가 TodoProvider 컴포넌트 내부에 렌더링되어 있어야 한다 (예: App 컴포넌트에서 모든 내용을 TodoProvider로 감싸기).
- 만약 TodoProvider 로 감싸져 있지 않다면 에러를 발생시키도록 커스텀 Hook 을 수정해보겠다.
- 꼭 이렇게 해줄 필요는 없지만, Context 사용을 위한 커스텀 Hook 을 만들 때 이렇게 에러 처리를 해준다면, 나중에 실수를 하게 됐을 때 문제점을 빨리 발견 할 수 있다.

TODO list

커스텀 Hook 에서 에러 처리

```
import React, { useReducer, createContext, useContext, useRef } from
'react';

const initialTodos = [
  {
    id: 1,
    text: '프로젝트 생성하기',
    done: true
  },
  {
    id: 2,
    text: '컴포넌트 스타일링하기',
    done: true
  },
  {
    id: 3,
    text: 'Context 만들기',
    done: false
  },
  {
    id: 4,
    text: '기능 구현하기',
    done: false
  }
];
```

```
function todoReducer(state, action) {
  switch (action.type) {
    case 'CREATE':
      return state.concat(action.todo);
    case 'TOGGLE':
      return state.map(todo =>
        todo.id === action.id ? { ...todo, done: !todo.done } : todo
      );
    case 'REMOVE':
      return state.filter(todo => todo.id !== action.id);
    default:
      throw new Error(`Unhandled action type: ${action.type}`);
  }
}

const TodoStateContext = createContext();
const TodoDispatchContext = createContext();
const TodoNextIdContext = createContext();

export function TodoProvider({ children }) {
  const [state, dispatch] = useReducer(todoReducer, initialTodos);
  const nextId = useRef(5);
```

TODO list

커스텀 Hook 에서 에러 처리

```
  return (
    <TodoStateContext.Provider value={state}>
      <TodoDispatchContext.Provider value={dispatch}>
        <TodoNextIdContext.Provider value={nextId}>
          {children}
        </TodoNextIdContext.Provider>
      </TodoDispatchContext.Provider>
    </TodoStateContext.Provider>
  );
}

export function useTodoState() {
  const context = useContext(TodoStateContext);
  if (!context) {
    throw new Error('Cannot find TodoProvider');
  }
  return context;
}
```

```
export function useTodoDispatch() {
  const context = useContext(TodoDispatchContext);
  if (!context) {
    throw new Error('Cannot find TodoProvider');
  }
  return context;
}

export function useTodoNextId() {
  const context = useContext(TodoNextIdContext);
  if (!context) {
    throw new Error('Cannot find TodoProvider');
  }
  return context;
}
```

TODO list

컴포넌트 TodoProvider로 감싸기

- 프로젝트 모든 곳에서 Todo 관련 Context들을 사용할 수 있도록, App 컴포넌트에서 TodoProvider를 불러와서 모든 내용을 TodoProvider 로 감싸주겠다.

```
import React from 'react';
import { createGlobalStyle } from 'styled-components';
import TodoTemplate from './components/TodoTemplate';
import TodoHead from './components/TodoHead';
import TodoList from './components/TodoList';
import TodoCreate from './components/TodoCreate';
import { TodoProvider } from './TodoContext';

const GlobalStyle = createGlobalStyle`
  body {
    background: #e9ecef;
  }
`;
```

```
function App() {
  return (
    <TodoProvider>
      <GlobalStyle />
      <TodoTemplate>
        <TodoHead />
        <TodoList />
        <TodoCreate />
      </TodoTemplate>
    </TodoProvider>
  );
}

export default App;
```

TODO list

컴포넌트 TodoProvider로 감싸기

- TodoHead 컴포넌트에서 useTodoState 사용

```
import React from 'react';
import styled from 'styled-components';
import { useTodoState } from '../TodoContext';

const TodoHeadBlock = styled.div`
  padding-top: 48px;
  padding-left: 32px;
  padding-right: 32px;
  padding-bottom: 24px;
  border-bottom: 1px solid #e9ecef;
  h1 {
    margin: 0;
    font-size: 36px;
    color: #343a40;
  }
  .day {
    margin-top: 4px;
    color: #868e96;
    font-size: 21px;
  }
  .tasks-left {
    color: #20c997;
    font-size: 18px;
    margin-top: 40px;
    font-weight: bold;
  }
`;

function TodoHead() {
  const todos = useTodoState();
  console.log(todos);
  return (
    <TodoHeadBlock>
      <h1>2019년 7월 10일</h1>
      <div className="day">수요일</div>
      <div className="tasks-left">할 일 2개 남음</div>
    </TodoHeadBlock>
  );
}

export default TodoHead;
```

```
▼ Array(4) i
  ▶ 0: {id: 1, text: '프로젝트 생성하기', done: true}
  ▶ 1: {id: 2, text: '컴포넌트 스타일링하기', done: true}
  ▶ 2: {id: 3, text: 'Context 만들기', done: false}
  ▶ 3: {id: 4, text: '기능 구현하기', done: false}
    length: 4
```

TODO list

기능 구현하기

- Context 와 연동을 하여 기능을 구현
- Context 에 있는 state 를 받아와서 렌더링을 하고, 필요한 상황에는 특정 액션을 dispatch 하면 됨.

TODO list

TodoHead 완성하기

- Context 와 연동을 하여 기능을 구현
- TodoHead에서는 done 값이 false인 항목들의 개수를 화면에 보여줌

```
import React from 'react';
import styled from 'styled-components';
import { useTodoState } from '../TodoContext';

const TodoHeadBlock = styled.div`
  padding-top: 48px;
  padding-left: 32px;
  padding-right: 32px;
  padding-bottom: 24px;
  border-bottom: 1px solid #e9ecef;
  h1 {
    margin: 0;
    font-size: 36px;
    color: #343a40;
  }
  .day {
    margin-top: 4px;
    color: #868e96;
    font-size: 21px;
  }
```

```
  .tasks-left {
    color: #20c997;
    font-size: 18px;
    margin-top: 40px;
    font-weight: bold;
  }
`;
```

```
function TodoHead() {
  const todos = useTodoState();
  const undoneTasks = todos.filter(todo => !todo.done);

  return (
    <TodoHeadBlock>
      <h1>2024년 5월 17일</h1>
      <div className="day">금요일</div>
      <div className="tasks-left">할 일 {undoneTasks.length}개 남음
</div>
    </TodoHeadBlock>
  );
}

export default TodoHead;
```

TODO list

TodoHead 완성하기

- 날짜가 보여지는 부분을 작업
- 이 과정에서는 Date 의 toLocaleString 이라는 함수를 사용

Date.prototype.toLocaleString()

```
import React from 'react';
import styled from 'styled-components';
import { useTodoState } from '../TodoContext';

const TodoHeadBlock = styled.div`
  padding-top: 48px;
  padding-left: 32px;
  padding-right: 32px;
  padding-bottom: 24px;
  border-bottom: 1px solid #e9ecef;
  h1 {
    margin: 0;
    font-size: 36px;
    color: #343a40;
  }
```

```
  .day {
    margin-top: 4px;
    color: #868e96;
    font-size: 21px;
  }
  .tasks-left {
    color: #20c997;
    font-size: 18px;
    margin-top: 40px;
    font-weight: bold;
  }
`;

function TodoHead() {
  const todos = useTodoState();
  const undoneTasks = todos.filter(todo => !todo.done);
```

TODO list

TodoHead 완성하기

```
  const today = new Date();
  const dateString = today.toLocaleDateString('ko-KR', {
    year: 'numeric',
    month: 'long',
    day: 'numeric'
  });
  const dayName = today.toLocaleDateString('ko-KR', { weekday: 'long' });

  return (
    <TodoHeadBlock>
      <h1>{dateString}</h1>
      <div className="day">{dayName}</div>
      <div className="tasks-left">할 일 {undoneTasks.length}개 남음
</div>
    </TodoHeadBlock>
  );
}
```

TODO list

TodoList 완성하기

- TodoList 에서는 state를 조회하고 이를 렌더링

```
import React from 'react';
import styled from 'styled-components';
import TodoItem from './TodoItem';
import { useTodoState } from '../TodoContext';

const TodoListBlock = styled.div`
  flex: 1;
  padding: 20px 32px;
  padding-bottom: 48px;
  overflow-y: auto;
`;

function TodoList() {
  const todos = useTodoState();
```

```
  return (
    <TodoListBlock>
      {todos.map(todo => (
        <TodoItem
          key={todo.id}
          id={todo.id}
          text={todo.text}
          done={todo.done}
        />
      ))}
    </TodoListBlock>
  );
}

export default TodoList;
```

TODO list

TodoItem 완성하기

- dispatch를 사용해서 토글 기능과 삭제 기능을 구현

```
import React from 'react';
import styled, { css } from 'styled-components';
import { MdDone, MdDelete } from 'react-icons/md';
import { useTodoDispatch } from '../TodoContext';

const Remove = styled.div`
  display: flex;
  align-items: center;
  justify-content: center;
  color: #dee2e6;
  font-size: 24px;
  cursor: pointer;
  &:hover {
    color: #ff6b6b;
  }
 display: none;
`;
```

```
const TodoItemBlock = styled.div`
  display: flex;
  align-items: center;
  padding-top: 12px;
  padding-bottom: 12px;
  &:hover {
    ${Remove} {
      display: initial;
    }
  }
`;

const CheckCircle = styled.div`
  width: 32px;
  height: 32px;
  border-radius: 16px;
  border: 1px solid #ced4da;
  font-size: 24px;
```

```
 display: flex;
  align-items: center;
  justify-content: center;
  margin-right: 20px;
  cursor: pointer;
  ${props =>
    props.done &&
    css`
      border: 1px solid #38d9a9;
      color: #38d9a9;
    `}
`;
```

TODO list

TodoItem 완성하기

```
const Text = styled.div`
  flex: 1;
  font-size: 21px;
  color: #495057;
  ${props =>
    props.done &&
    css`
      color: #ced4da;
    `}
`;

function TodoItem({ id, done, text }) {
  const dispatch = useTodoDispatch();
  const onToggle = () => dispatch({ type: 'TOGGLE', id });
  const onRemove = () => dispatch({ type: 'REMOVE', id });
```

```
  return (
    <TodoItemBlock>
      <CheckCircle done={done} onClick={onToggle}>
        {done && <MdDone />}
      </CheckCircle>
      <Text done={done}>{text}</Text>
      <Remove onClick={onRemove}>
        <MdDelete />
      </Remove>
    </TodoItemBlock>
  );
}

export default React.memo(TodoItem);
```

마지막에
export default React.memo(TodoItem);

이렇게 하면, 다른 항목이 업데이트 될 때, 불필요한 리렌더링을 방지하게 되어 성능을 최적화 할 수 있음

TODO list

TodoCreate 완성하기

- 자체적으로 관리해야 할 input 상태도 같이 처리

```
import React, { useState } from 'react';
import styled, { css } from 'styled-components';
import { MdAdd } from 'react-icons/md';
import { useTodoDispatch, useTodoNextId } from '../TodoContext';

const CircleButton = styled.button`
  background: #38d9a9;
  &:hover {
    background: #63e6be;
  }
  &:active {
    background: #20c997;
  }

  z-index: 5;
  cursor: pointer;
  width: 80px;
  height: 80px;
  display: block;
  align-items: center;
  justify-content: center;
  font-size: 60px;
  position: absolute;
  left: 50%;
  bottom: 0px;
  transform: translate(-50%, 50%);
  color: white;
  border-radius: 50%;
  border: none;
  outline: none;
  display: flex;
  align-items: center;
  justify-content: center;

  transition: 0.125s all ease-in;
  ${props =>
    props.open &&
    css`
      background: #ff6b6b;
      &:hover {
        background: #ff8787;
      }
      &:active {
        background: #fa5252;
      }
      transform: translate(-50%, 50%) rotate(45deg);
    `}
`;
```

TODO list

TodoCreate 완성하기

```
const InsertFormPositioner = styled.div`
  width: 100%;
  bottom: 0;
  left: 0;
  position: absolute;
`;

const InsertForm = styled.form`
  background: #f8f9fa;
  padding-left: 32px;
  padding-top: 32px;
  padding-right: 32px;
  padding-bottom: 72px;

  border-bottom-left-radius: 16px;
  border-bottom-right-radius: 16px;
  border-top: 1px solid #e9ecef;
`;
```

```
const Input = styled.input`
  padding: 12px;
  border-radius: 4px;
  border: 1px solid #dee2e6;
  width: 100%;
  outline: none;
  font-size: 18px;
  box-sizing: border-box;
`;

function TodoCreate() {
  const [open, setOpen] = useState(false);
  const [value, setValue] = useState('');

  const dispatch = useTodoDispatch();
  const nextId = useTodoNextId();
```

```
  const onToggle = () => setOpen(!open);
  const onChange = e => setValue(e.target.value);
  const onSubmit = e => {
    e.preventDefault(); // 새로고침 방지
    dispatch({
      type: 'CREATE',
      todo: {
        id: nextId.current,
        text: value,
        done: false
      }
    });
    setValue('');
    setOpen(false);
    nextId.current += 1;
  };
```

TODO list

TodoCreate 완성하기

```
  return (
    <>
      {open && (
        <InsertFormPositioner>
          <InsertForm onSubmit={onSubmit}>
            <Input
              autoFocus
              placeholder="할 일을 입력 후, Enter 를 누르세요"
              onChange={onChange}
              value={value}
            />
          </InsertForm>
        </InsertFormPositioner>
      )}
      <CircleButton onClick={onToggle} open={open}>
        <MdAdd />
      </CircleButton>
    </>
  );
}

export default React.memo(TodoCreate);
```

레포트(optional)

HCJ, javascript, React 웹 사이트
TODO list App SWOT 분석

발표주제

톡방 공지

Reference

- https://react.vlpt.us/styling/01-sass.html
- https://react.vlpt.us/mashup-todolist/01-create-components.html
- https://wikidocs.net/197754
- https://blog.toycrane.xyz/css%EC%9D%98-%EC%A7%84%ED%99%94-%EA%B3%BC%EC%A0%95-f7c9b4310ff7

- https://www.youtube.com/watch?v=_JDeJgsU-bI

TL;DR