**2. UI 구현하기**

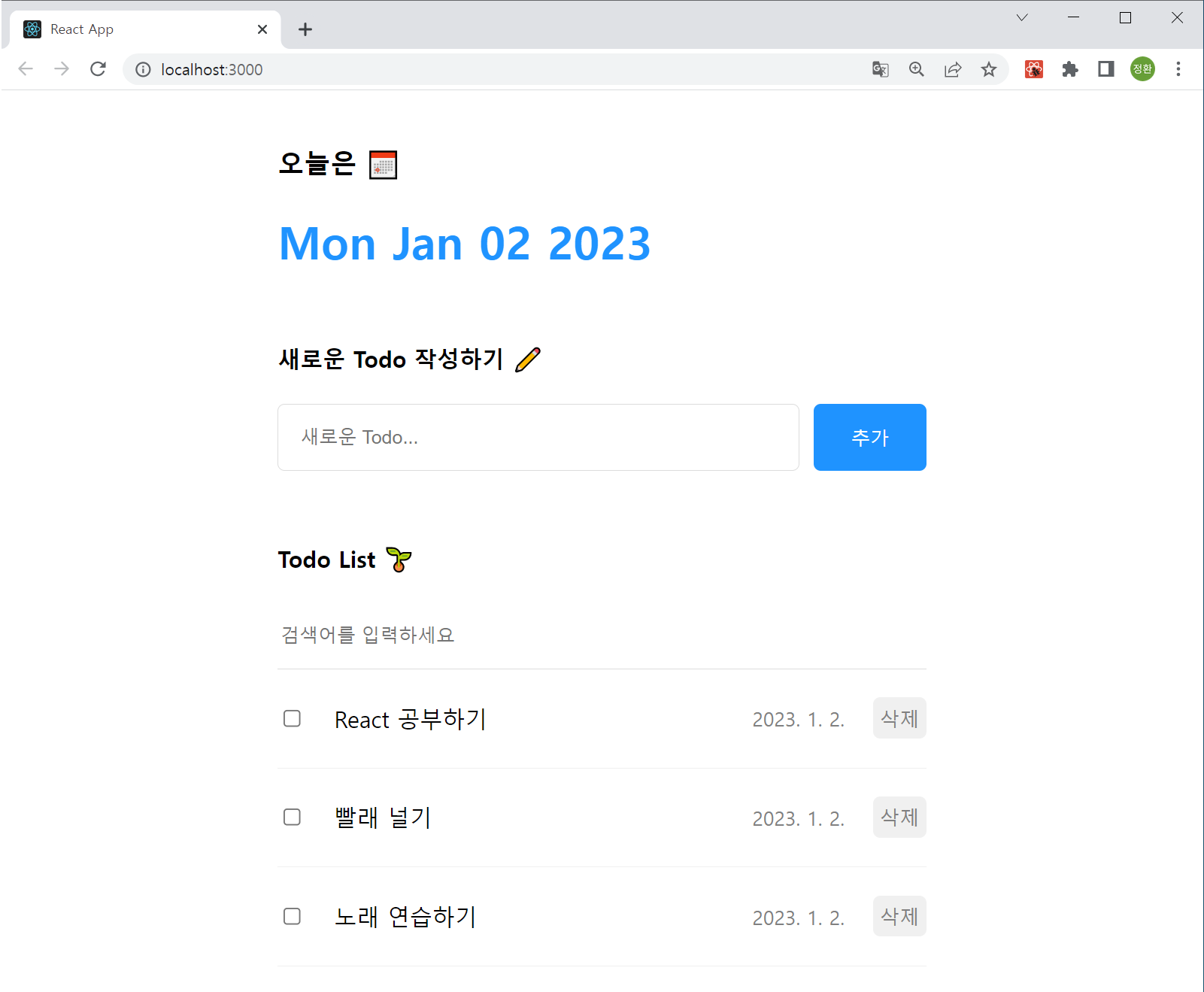
📄

**Table Of Contents**

프로젝트 준비를 모두 끝마쳤다면 UI를 구현하겠습니다. [할 일 관리] 앱의 UI 구현은 페이지의 전체 레이아웃부터 먼저 만들고, 세부 요소는 순서에 따라 차근차근 만들 예정입니다.

**페이지 레이아웃 만들기**

다음 그림은 이 프로젝트에서 구현할 [할 일 관리] 앱의 최종 형태를 UI 관점에서 보여 줍니다.



[할 일 관리] 앱의 최종 모습

[할 일 관리] 앱의 UI 요소는 마치 핸드폰을 웹 브라우저 위에 올려놓은 것처럼 좌우 여백이 넓으며 페이지의 정중앙에 자리 잡고 있습니다.

먼저 App.js에서 다음과 같이 <h2> 태그를 추가합니다.

**src/App.js**

**Copy**

import "./App.css";

function App() {

return (

<div className="App">

<h2>헬로 리액트</h2>

</div>

);

}

export default App;

다음에는 index.css에 작성된 스타일 규칙은 모두 삭제하고 다음과 같이 작성합니다.

**Copy**

body {

margin: 0px; ①

}

**①** <body> 태그의 margin을 0px로 설정합니다. margin은 여백이라는 뜻으로 0px로 설정하면 페이지의 외부 여백이 전부 사라집니다.

다음에는 App.css에 작성된 스타일 규칙은 모두 삭제하고 다음과 같이 작성합니다.

**Copy**

.App {

max-width: 500px; ①

width: 100%; ②

margin: 0 auto; ③

box-sizing: border-box; ④

padding: 20px; ⑤

border: 1px solid gray; ⑥

}

**①** [할 일 관리] 앱 페이지의 최대 너비를 500px로 고정합니다

**②** 페이지 너비를 브라우저의 100%로 설정합니다. 그러나 **①**에서 정한 규칙 때문에 500px 이상으로 페이지의 너비가 늘어나지 않습니다. 따라서 **①②** 규칙에 따라 [할 일 관리] 앱 페이지는 최대 500px의 너비를 갖습니다. 만약 브라우저의 너비가 500px보다 작아지면 페이지는 브라우저의 너비가 됩니다.

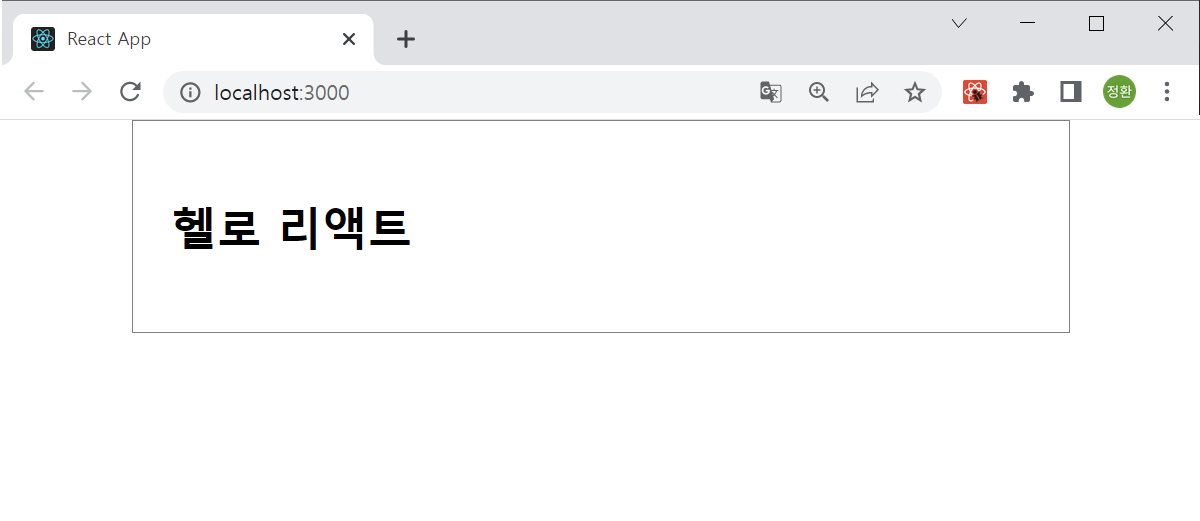
**③** 여백을 위아래는 0, 좌우는 자동으로 설정합니다. 좌우 여백을 자동으로 설정하면, UI 요소를 브라우저 가운데에 배치하기 위해 여백이 자동으로 조절됩니다. 예를 들어 현재 브라우저의 너비가 700px면 좌우로 100px의 여백이 자동으로 설정되면서 페이지 UI가 브라우저 정중앙에 자리 잡습니다. 만약 브라우저의 너비가 500px 이하라면 좌우 여백은 자동으로 0이 됩니다.

**④** box-sizing은 요소의 크기를 어떤 것을 기준으로 계산할지 정하는 속성입니다. box-sizing 속성을 border-box로 설정해 내부 여백이 요소의 크기에 영향을 미치지 않도록 설정합니다.

**⑤** 내부 여백을 20px로 설정합니다.

**⑥** 경계선을 1px 두께의 회색 실선으로 표시합니다. 이 경계선은 컴포넌트의 경계를 표시하기 위해 사용합니다. 이 경계선은 나중에 모두 삭제할 예정입니다.

저장하고 레이아웃이 잘 만들어졌는지 확인합니다.

새롭게 CSS 스타일 규칙 지정

App에는 3개의 자식 컴포넌트 Header, TodoEditor, TodoList를 각각 세로로 배치할 예정입니다. App에 배치할 자식 컴포넌트를 아직 구현하지 않았으므로, 임시 요소를 만들어 대신 배치하겠습니다.

App.js를 다음과 같이 수정합니다.

**Copy**

import "./App.css";

function App() {

return (

<div className="App">

<div>Header</div>

<div>Todo Editor</div>

<div>Todo List</div>

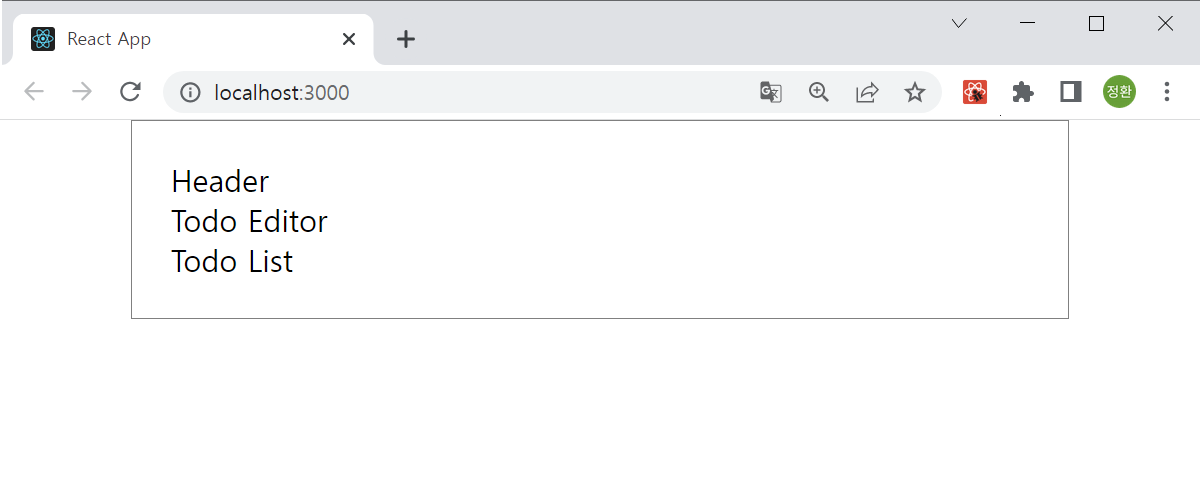
</div>

);

}

export default App;

저장하고 렌더링 결과를 확인합니다.

3개의 임시 자식 요소 렌더링

페이지의 요소를 모두 세로로 배치했지만, 요소 사이에 간격이 없어 답답해 보입니다. 이때는 App 컴포넌트의 display 속성을 이용하면, 요소의 배치간격을 좀 더 보기 좋게 만들 수 있습니다.

다음과 같이 App.css를 수정합니다.

**Copy**

.App {

(...)

display: flex; ①

flex-direction: column; ②

gap: 30px;③

}

**①** display는 페이지의 요소를 브라우저에서 어떻게 보여줄지 결정하는 속성으로, 기본값은 block 입니다. block은 배치 요소들을 한 줄로 꽉 채우며 세로로 배치합니다. display 속성을 flex로 설정하면 요소를 수직이 아닌 수평으로 배치합니다. 그리고 gap과 같이 요소의 간격을 조정하는 속성을 추가로 사용할 수 있습니다. display 속성을 flex로 적용한 요소는 다른 요소를 감싸는 컨테이너라고 하여, 특별히 ‘플렉스 컨테이너(Flex Container)’라고 합니다.

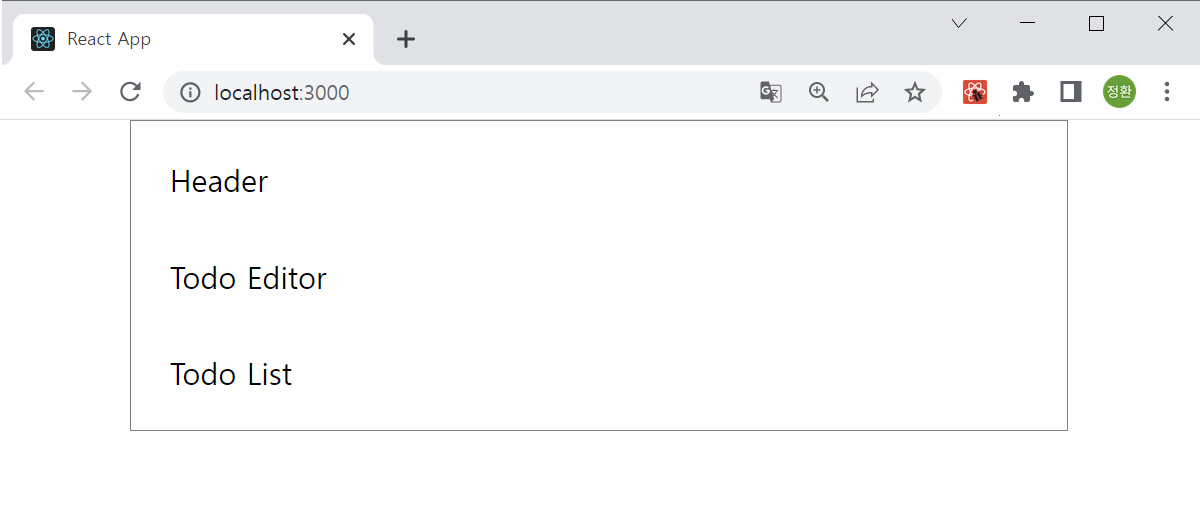
**②** flex-direction은 플렉스 컨테이너에 있는 요소들의 배치 방향을 조절하는 속성입니다. 기본값은 row로 요소를 수평으로 배치합니다. 이 속성을 column으로 변경하면, App 컴포넌트의 요소를 수직으로 배치합니다.

**③** gap은 App 컴포넌트에서 자식 요소 간의 여백을 조절하는 속성입니다.

✋🏼

CSS의 flex 속성은 실무에서 자주 사용하는 매우 유용한 기능입니다. flex에 대한 자세한 설명은 구글에서 ‘CSS Flexible Box Layout’이라고 검색하면 상세히 나옵니다. 꼭 익혀두길 바랍니다.

요소의 간격이 잘 설정되었는지 렌더링 결과를 확인합니다.

flex 기능을 이용해 요소 사이의 간격 조정하기

요소의 간격이 적절히 떨어져 있음을 알 수 있습니다. 이렇게 App 컴포넌트의 스타일링을 모두 마무리하였습니다.

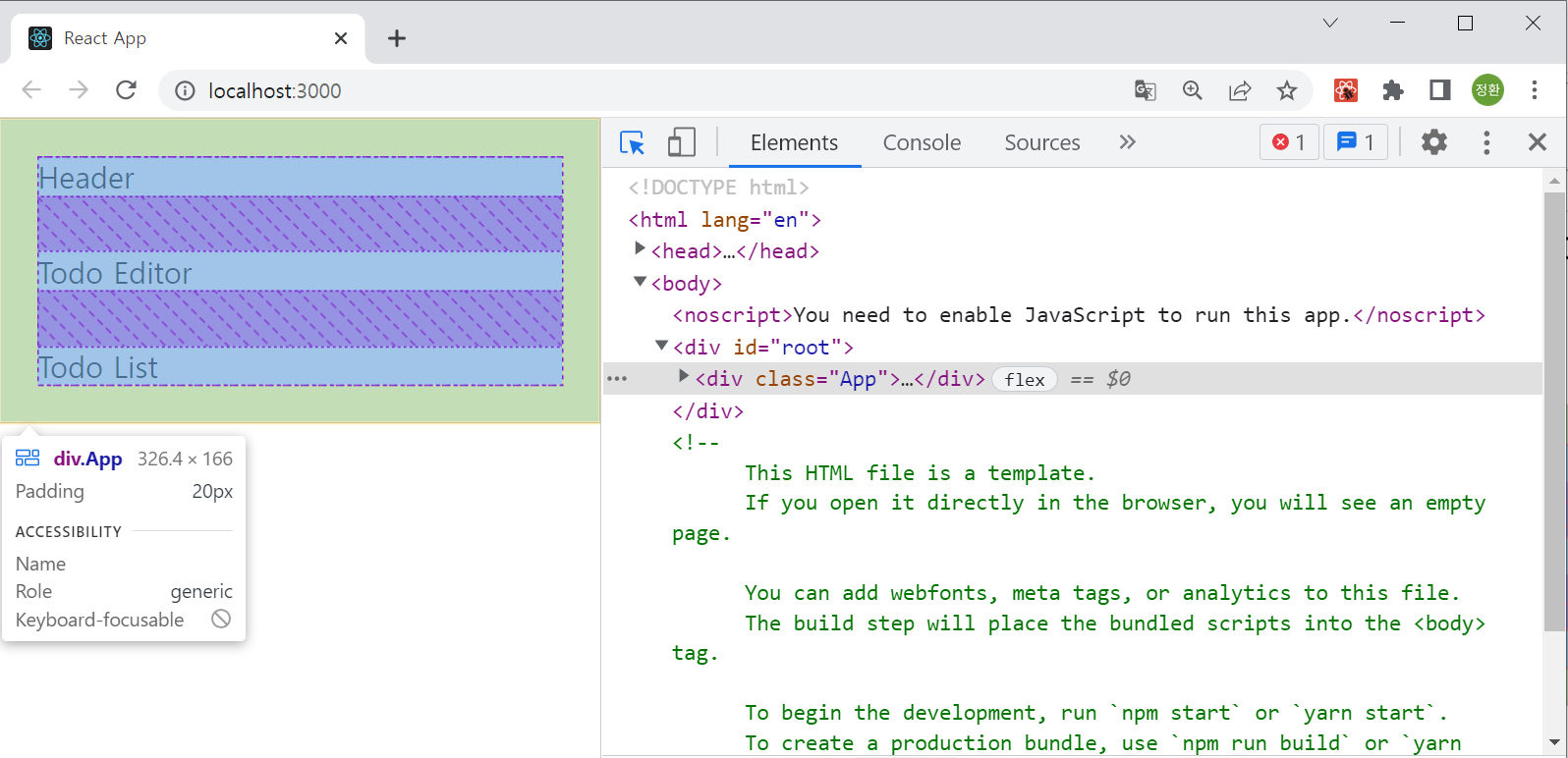
✋🏼

**개발자 도구에서 flex 적용 확인하기**

크롬 브라우저의 개발자 도구에서 요소의 배치 상황을 구체적으로 볼 수 있습니다. display 속성을 flex로 했을 때 요소의 배치가 어떻게 되는지 확인해 보겠습니다.

개발자 도구를 열고 [Elements] 탭을 클릭합니다. [Elements] 탭 옆에 있는 요소 선택 아이콘을 클릭한 다음, 지금 페이지에 렌더링한 박스를 클릭합니다.

다시 개발자 도구의 [Elements] 탭을 살펴보면, App 컴포넌트의 최상위 태그인 <div class='App'>에 flex가 적용되었음을 알리는 표시가 있습니다. 이 태그 위에 마우스 포인터를 올립니다.

개발자 도구에서 flex 속성일 때의 배치 확인

다시 페이지를 보면 앞서 설정한 gap 속성을 색으로 표현하고 있습니다.

**Header 컴포넌트 만들기**

이번에는 페이지 최상단에 위치할 Header 컴포넌트를 만들겠습니다.

src에 이 프로젝트의 컴포넌트 파일을 한곳에 모아 둘 component 폴더를 만듭니다. 계속해서 component 폴더에 Header.js를 생성하고 다음과 같이 작성합니다.

**src/component/Header.js**

**Copy**

const Header = () => {

return <div className="Header">Header Component</div>;

};

export default Header;

Header 컴포넌트를 페이지에 렌더링하려면 App의 자식으로 배치해야 합니다.

**src/App.js**

**Copy**

import "./App.css";

import Header from "./component/Header";

function App() {

return (

<div className="App">

<Header />

<div>Todo Editor</div>

<div>Todo List</div>

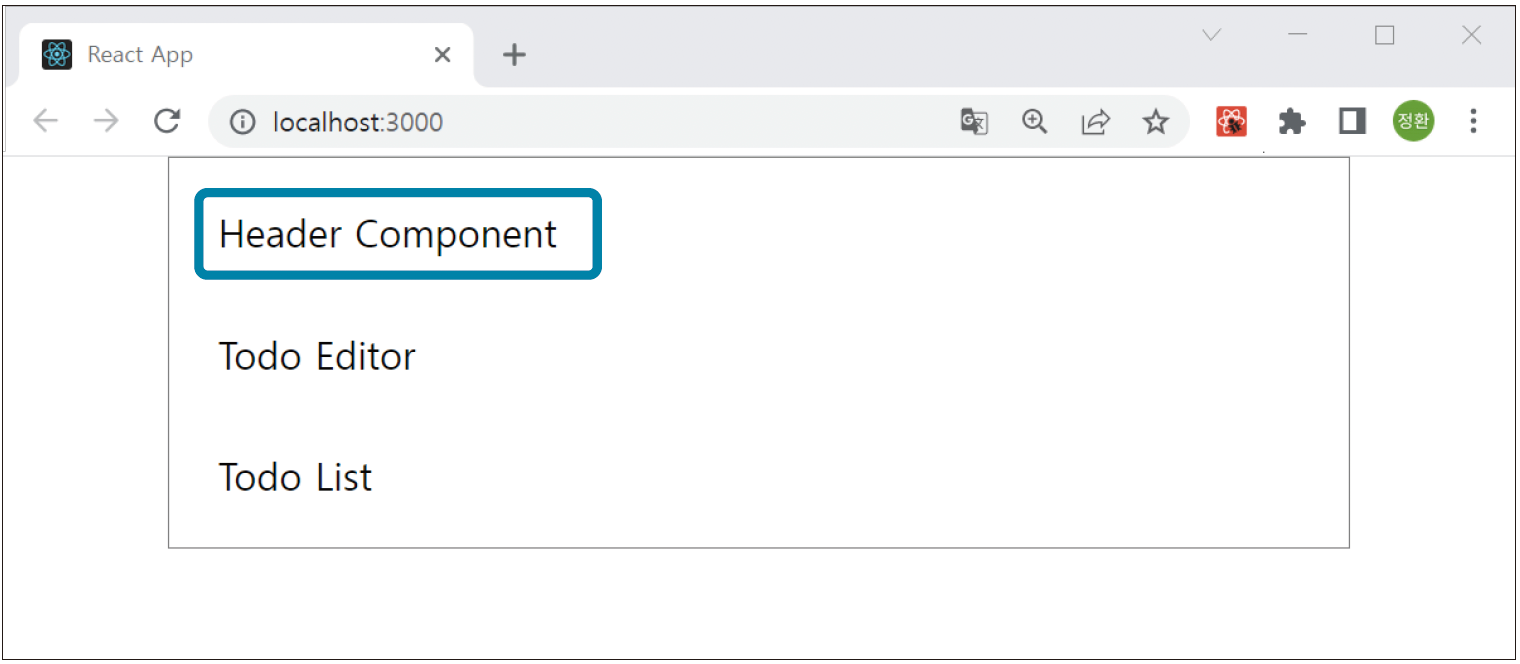
</div>

);

}

export default App;

App 컴포넌트의 return 문 안에 임시로 Header 역할을 수행했던 요소를 제거하고 실제 Header 컴포넌트를 작성했습니다. 저장하고 렌더링 결과를 확인합니다.

Header 컴포넌트 배치

다음으로 Header 컴포넌트가 오늘의 날짜를 렌더링 하도록 다음과 같이 수정합니다.

**Copy**

const Header = () => {

return (

<div className="Header">

<h3>오늘은 📅</h3> ①

<h1>{new Date().toDateString()}</h1> ②

</div>

);

};

export default Header;

**①** 문자열 ‘오늘은’ 옆에 있는 달력 모양의 이모지는 윈도우 이모티콘으로 <window>+<.>(마침표) 키를 누르면 나옵니다. 여기서 적절한 이모지를 골라 사용하면 됩니다. 이모지는 문자열로 취급하므로 일반 문자열을 입력해도 상관없습니다.

**②** 현재의 날짜와 시간을 저장하는 Date 객체를 만들고, toDateString 메서드를 이용해 날짜를 문자열로 표시합니다.

✋🏼

macOS 사용자라면 Ctrl + Command + Spacebar 단축키를 눌러 이모티콘를 입력할 수 있습니다.

계속해서 component 폴더에 Header 컴포넌트를 스타일링하기 위한 Header.css를 생성하고 다음과 같이 작성합니다.

**Copy**

.Header h1 { ①

margin-bottom: 0px;

color: #1f93ff;

}

**①** <h1> 태그 요소의 여백을 0으로 하고 글꼴 색을 지정합니다.

작성한 스타일 규칙을 컴포넌트에 적용하려면 Header.css를 Header.js에서 불러와야 합니다.

**src/component/Header.js**

**Copy**

import "./Header.css"; ①

const Header = () => {

return (

<div className="Header">

<h3>오늘은 📅</h3>

<h1>{new Date().toDateString()}</h1>

</div>

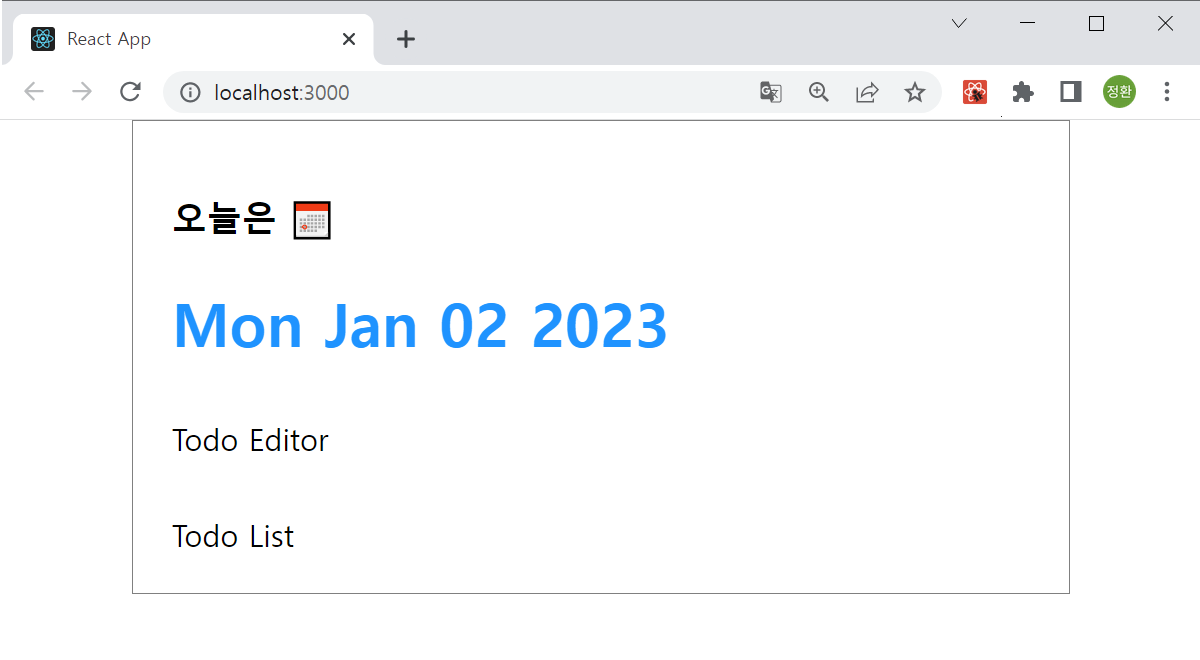
);

};

export default Header;

**①** Header.css 파일을 불러와 작성한 스타일 규칙을 적용합니다.

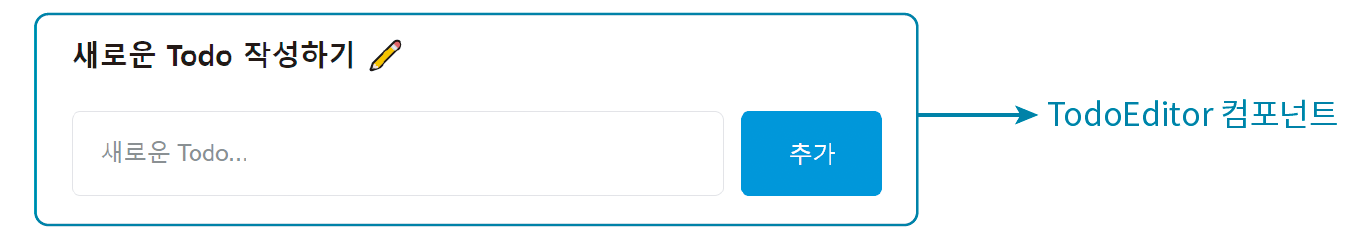
저장한 다음, 스타일을 잘 변경 했는지 렌더링 결과를 확인합니다.

Header 컴포넌트의 스타일 적용

페이지에 렌더링한 날짜는 여러분이 실습하고 있는 당일 날짜가 표시됩니다. 이 책의 결과와는 다르다는 점에 주의합니다.

**TodoEditor 컴포넌트 만들기**

계속해서 할 일 아이템을 생성하는 TodoEditor 컴포넌트를 만듭니다.

TodoEditor 컴포넌트의 모습

component 폴더에 컴포넌트와 스타일을 정의할 TodoEditor.js와 TodoEditor.css 를 각각 생성합니다.

그리고 TodoEditor 컴포넌트에 다음과 같이 작성합니다.

**src/component/TodoEditor.js**

**Copy**

import "./TodoEditor.css";

const TodoEditor = () => {

return <div className="TodoEditor">TodoEditor Component</div>;

};

export default TodoEditor;

다음에는 App에 TodoEditor 컴포넌트를 자식으로 배치합니다.

**src/App.js**

**Copy**

import "./App.css";

import Header from "./component/Header";

import TodoEditor from "./component/TodoEditor";

function App() {

return (

<div className="App">

<Header />

<TodoEditor />

<div>Todo List</div>

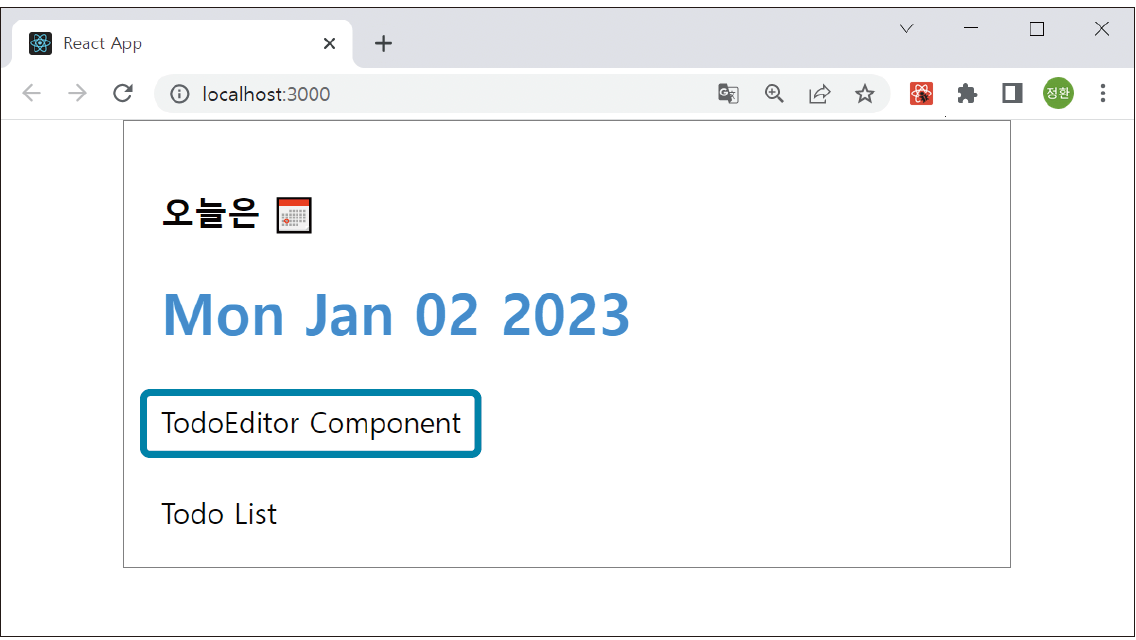
</div>

);

}

export default App;

저장한 다음, TodoEditor 컴포넌트를 페이지에 잘 렌더링하는지 확인합니다.

TodoEditor 컴포넌트 배치

렌더링 오류가 발생한다면 파일명을 제대로 작성했는지, 스타일 파일을 바른 경로로 불러오는지 확인하길 바랍니다.

다음으로 TodoEditor의 UI를 만듭니다.

**src/component/TodoEditor.js**

**Copy**

import "./TodoEditor.css";

const TodoEditor = () => {

return (

<div className="TodoEditor">

<h4>새로운 Todo 작성하기 ✏️ </h4>

<div className="editor\_wrapper">

<input placeholder="새로운 Todo..." />

<button>추가</button>

</div>

</div>

);

};

export default TodoEditor;

TodoEditor 컴포넌트의 UI를 만듭니다. 컴포넌트는 요소의 제목, 할 일 아이템을 생성하는 입력 폼, 클릭하면 실제 할 일 아이템을 생성하는 버튼으로 구성되어 있습니다.

다음에는 TodoEditor 컴포넌트를 스타일링하기 위한 스타일 규칙을 작성합니다

**src/component/TodoEditor.css**

**Copy**

.TodoEditor .editor\_wrapper {

width: 100%;

display: flex;

gap: 10px;

}

.TodoEditor input {

flex: 1;

box-sizing: border-box;

border: 1px solid rgb(220, 220, 220);

border-radius: 5px;

padding: 15px;

}

.TodoEditor input:focus {

outline: none;

border: 1px solid #1f93ff;

}

.TodoEditor button {

cursor: pointer;

width: 80px;

border: none;

background-color: #1f93ff;

color: white;

border-radius: 5px;

}

**①** 입력 폼과 버튼을 감싸는 요소에 스타일을 설정합니다. 너비 100%, display는 flex 속성을 적용합니다. 자식 요소의 간격은 10px입니다.

**②** 입력 폼에 스타일을 설정합니다. flex를 1로 설정하면 해당 요소의 너비가 브라우저의 크기에 따라 유연하게 늘어나고 줄어듭니다.

**③** 입력 폼을 클릭 했을 때의 스타일을 설정합니다. outline 속성을 none으로 설정하면 입력 폼을 클릭했을때 두꺼운 경계선이 생기지 않습니다. 경계선의 색상은 파란색을 적용합니다.

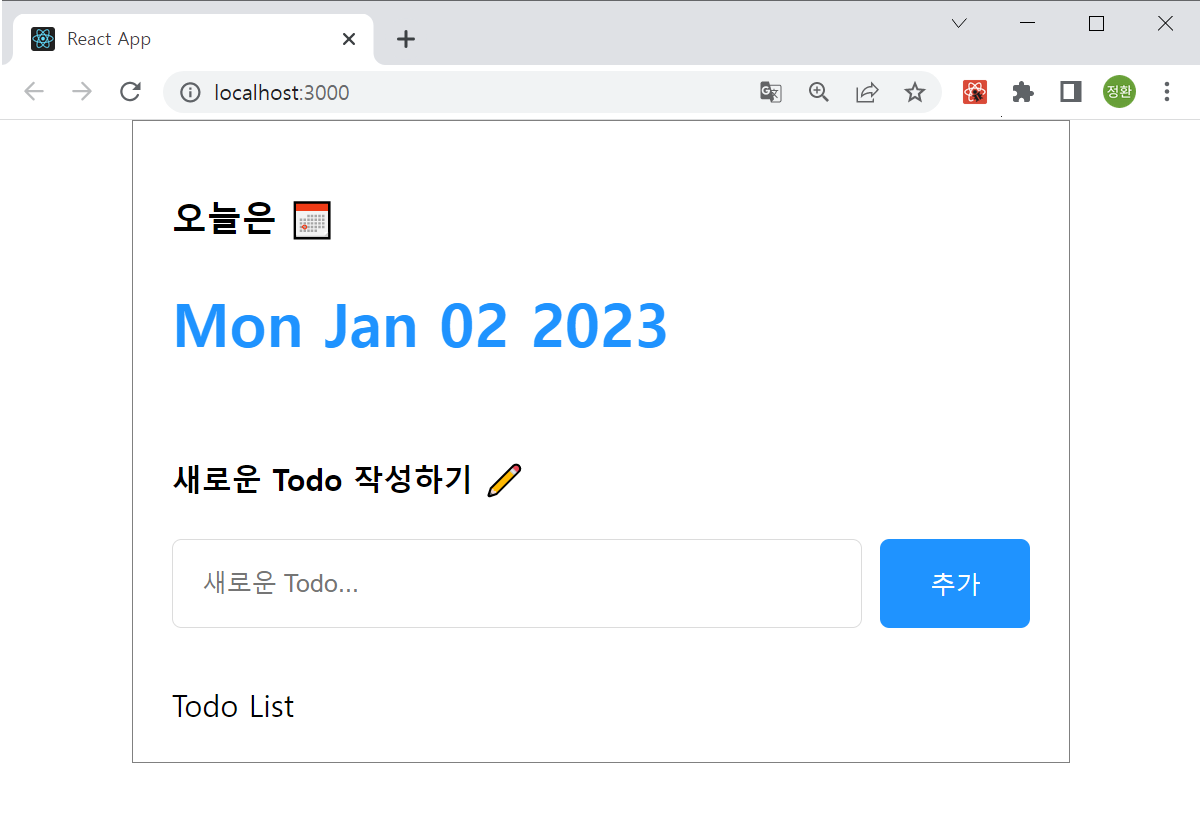
**④** 입력 폼 오른쪽에 위치할 버튼 스타일을 설정합니다. cursor 속성을 pointer로 설정하면 버튼에 마우스 포인터를 올릴 때 모양이 손 모양으로 바뀝니다.

✋🏼

CSS는 공부할 내용이 많습니다. 이 책은 리액트를 다루는 방법을 배우는 게 주목적이기 때문에 CSS 속성들에 대해서는 상세히 다루지 않습니다. 따라서 모든 내용을 다 이해하지 못한다고 해서 낙심하지 않았으면 좋겠습니다. 다만 웹 프론트엔드 개발에서 CSS 스타일링은 기능 구현 못지않게 중요한 소양이므로 꾸준히 공부해 익숙해지는 게 필요합니다. 일단 이 책에서 다루는 CSS 스타일만이라도 검색 등을 이용해 충분히 숙지하기를 바랍니다.

저장한 다음, 페이지에서 렌더링 결과를 확인합니다.

코드에서는 입력 폼을 클릭하면 폼의 경계선을 푸른색으로 표시(input:focus)하도록 구현했습니다. 입력 폼을 클릭해 푸른색 경계선이 나타나는지도 확인합니다.

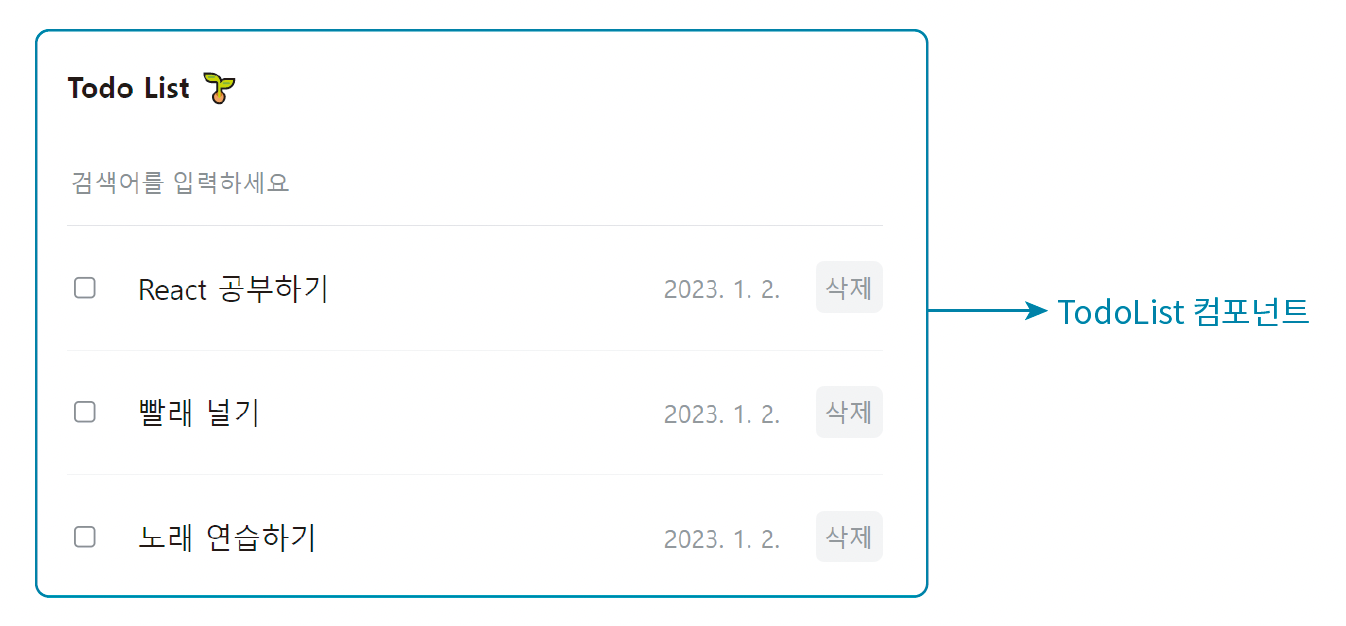
TodoEditor 컴포넌트의 스타일 지정

**TodoList, TodoItem 컴포넌트 만들기**

[할 일 관리] 앱의 `TodoList에는 TodoItem 컴포넌트가 여러개 있습니다. 따라서 이 두 컴포넌트를 함께 만들겠습니다.

**TodoList 컴포넌트 만들기**

먼저 TodoList 컴포넌트부터 만들겠습니다.

TodoList 컴포넌트의 모습

컴포넌트와 스타일을 정의하는 TodoList.js와 TodoList.css를 각각 생성합니다. 다음과 같이 TodoList 컴포넌트를 작성합니다.

**src/componen/TodoList.js**

**Copy**

import "./TodoList.css";

const TodoList = () => {

return <div className="TodoList">TodoList Component</div>;

};

export default TodoList;

TodoList를 App 컴포넌트의 자식으로 배치합니다.

**src/App.js**

**Copy**

import "./App.css";

import Header from "./component/Header";

import TodoEditor from "./component/TodoEditor";

import TodoList from "./component/TodoList";

function App() {

return (

<div className="App">

<Header />

<TodoEditor />

<TodoList />

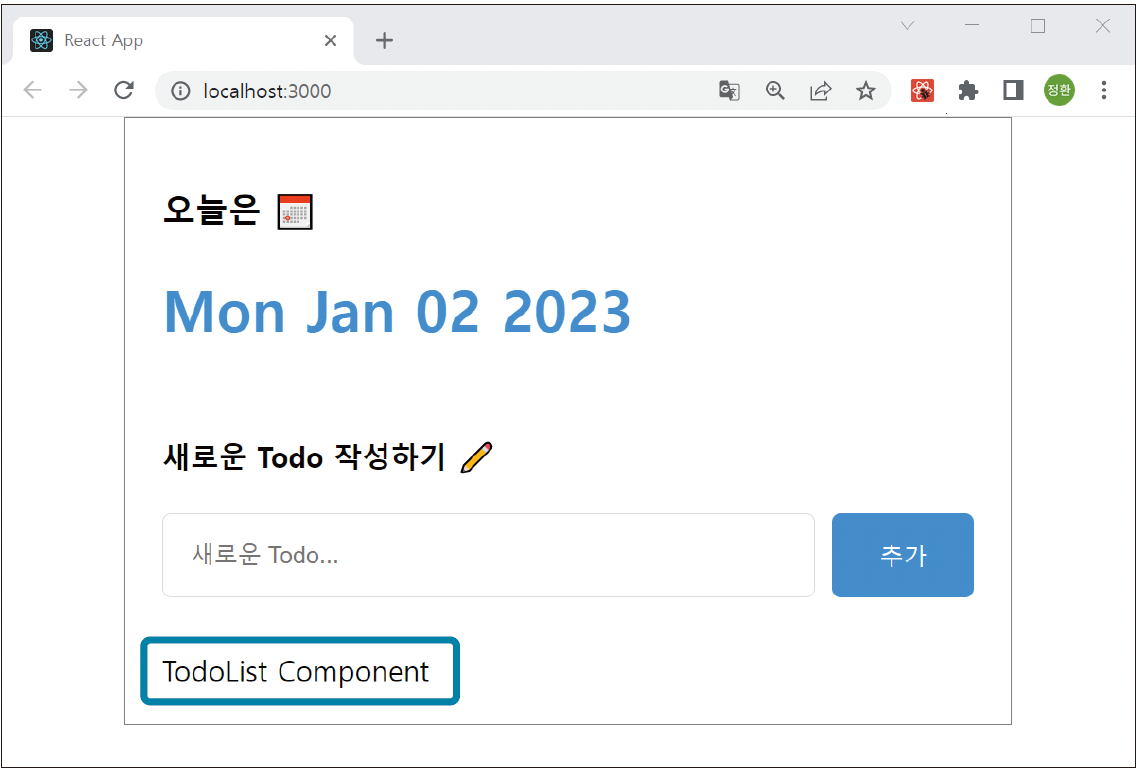
</div>

);

}

export default App;

저장한 다음, TodoList 컴포넌트가 페이지에 나타나는지 렌더링 결과를 확인합니다.

TodoList 컴포넌트 배치

TodoList 컴포넌트는 크게 할 일 아이템을 조회하는 검색 폼과 조회한 할일 아이템을 목록 형태로 보여주는 리스트 두 부분으로 구성되어 있습니다. 검색 결과에 따라 TodoItem 컴포넌트를 리스트로 페이지에 렌더링해야 합니다.

먼저 TodoList 컴포넌트 상단에 위치할 검색 폼부터 만들겠습니다.

**src/component/TodoList.js**

**Copy**

import "./TodoList.css";

const TodoList = () => {

return (

<div className="TodoList">

<h4>Todo List 🌱</h4>

<input className="searchbar" placeholder="검색어를 입력하세요" />

</div>

);

};

export default TodoList;

TodoList 검색 폼의 스타일링을 위해 TodoList.css를 다음과 같이 작성합니다.

**src/component/TodoList.css**

**Copy**

*/\* 검색 폼에 스타일 적용 \*/*

.TodoList .searchbar {

margin-bottom: 20px;

width: 100%;

border: none;

border-bottom: 1px solid rgb(220, 220, 220);

box-sizing: border-box;

padding-top: 15px;

padding-bottom: 15px;

}

*/\* 검색 폼을 클릭했을 때의 스타일 적용 \*/*

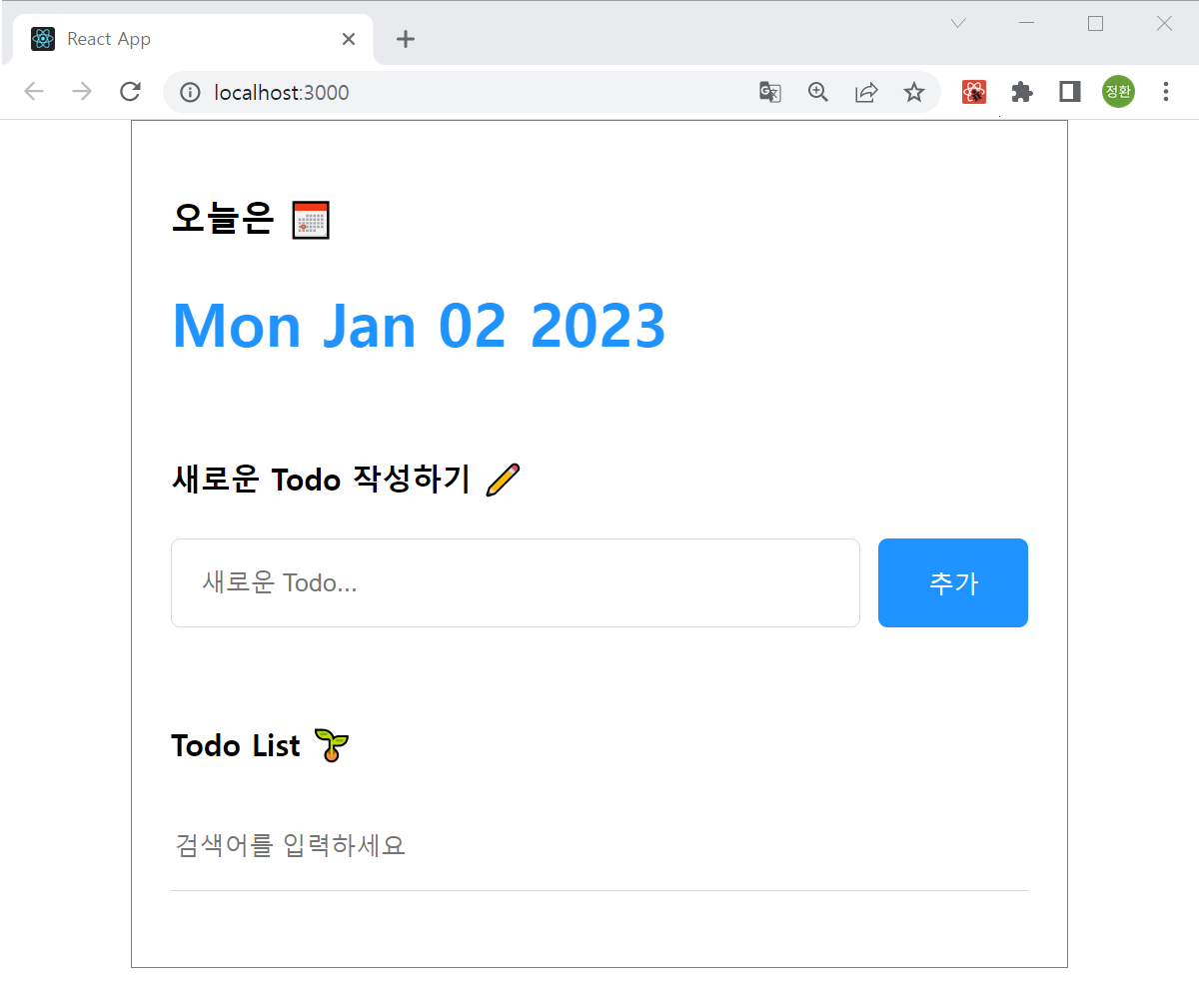
.TodoList .searchbar:focus {

outline: none;

border-bottom: 1px solid #1f93ff;

}

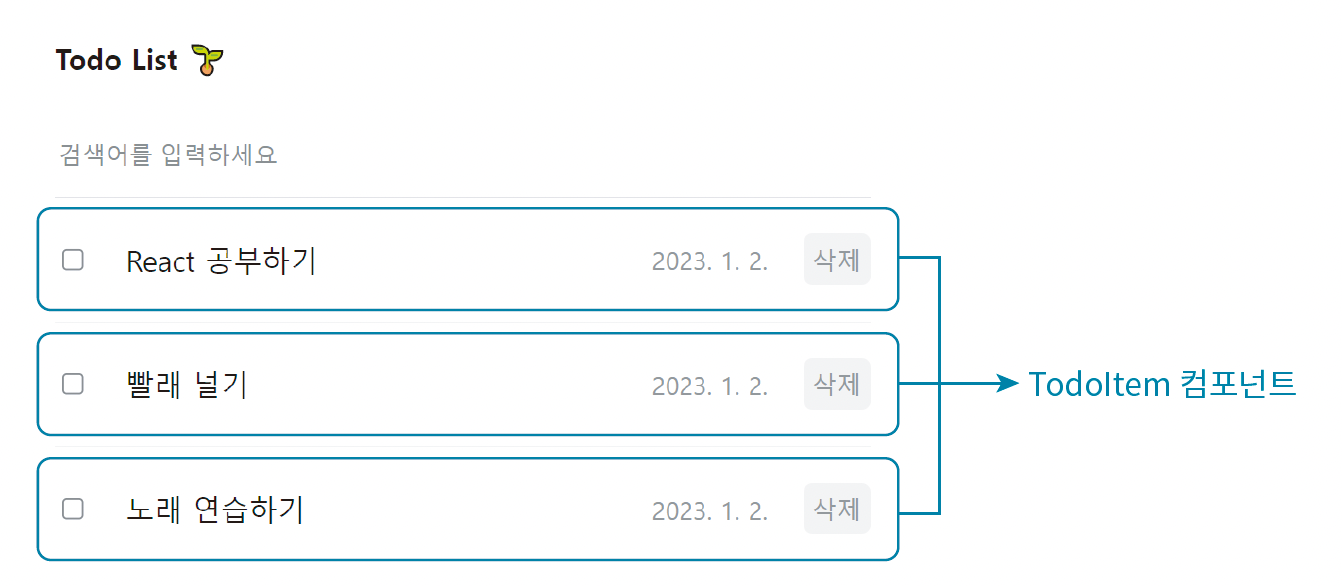
저장하고 페이지에서 렌더링 결과를 확인합니다.

TodoList 컴포넌트의 검색 폼 스타일 지정

검색 폼에서 검색어를 입력하면 조건에 일치하는 할 일 아이템이 하단에 리스트로 출력됩니다. 아직 TodoItem 컴 포넌트를 만들지 않았으므로 할 일 아이템을 출력하지는 못합니다.

**TodoItem 컴포넌트 만들기**

다음 그림과 같이 TodoList에서 낱낱의 할 일 아이템을 표현하는 TodoItem 컴포넌트를 만들겠습니다.

TodoItem 컴포넌트의 모습

component 폴더에서 컴포넌트와 스타일을 정의하는 TodoItem.js와 TodoItem.css를 각각 생성합니다.

TodoItem.js에서 다음과 같이 작성합니다.

**src/component/TodoItem.js**

**Copy**

import "./TodoItem.css";

const TodoItem = () => {

return (

<div className="TodoItem">

<div className="checkbox\_col"> ①

<input type="checkbox" />

</div>

<div className="title\_col">할 일</div> ②

<div className="date\_col">{new Date().toLocaleDateString()}</div> ③

<div className="btn\_col"> ④

<button>삭제</button>

</div>

</div>

);

};

export default TodoItem;

**①** 할 일 아이템 가장 왼쪽에는 할 일 완료 여부를 표시하는 체크박스를 배치합니다.

**②** 사용자가 작성한 할 일을 렌더링할 요소를 배치합니다. 지금은 임시로 ‘할 일’이라는 문자열을 렌더링합니다.

**③** 할 일 아이템이 작성된 시간을 렌더링할 요소를 배치합니다. 지금은 임시로 현재 시각을 렌더링합니다.

**④** 할 일을 삭제하는 버튼을 배치합니다.

**TodoList에 TodoItem 컴포넌트 배치하기**

TodoItem에 스타일을 적용하기 전에 TodoList에 이 컴포넌트를 배치해야 합니다. 다음과 같이 TodoList.js를 수정합니다.

**src/component/TodoList.js**

**Copy**

import TodoItem from "./TodoItem";

import "./TodoList.css";

const TodoList = () => {

return (

<div className="TodoList">

<h4>Todo List 🌱</h4>

<input className="searchbar" placeholder="검색어를 입력하세요" />

<div className="list\_wrapper"> ①

<TodoItem />

<TodoItem />

<TodoItem />

</div>

</div>

);

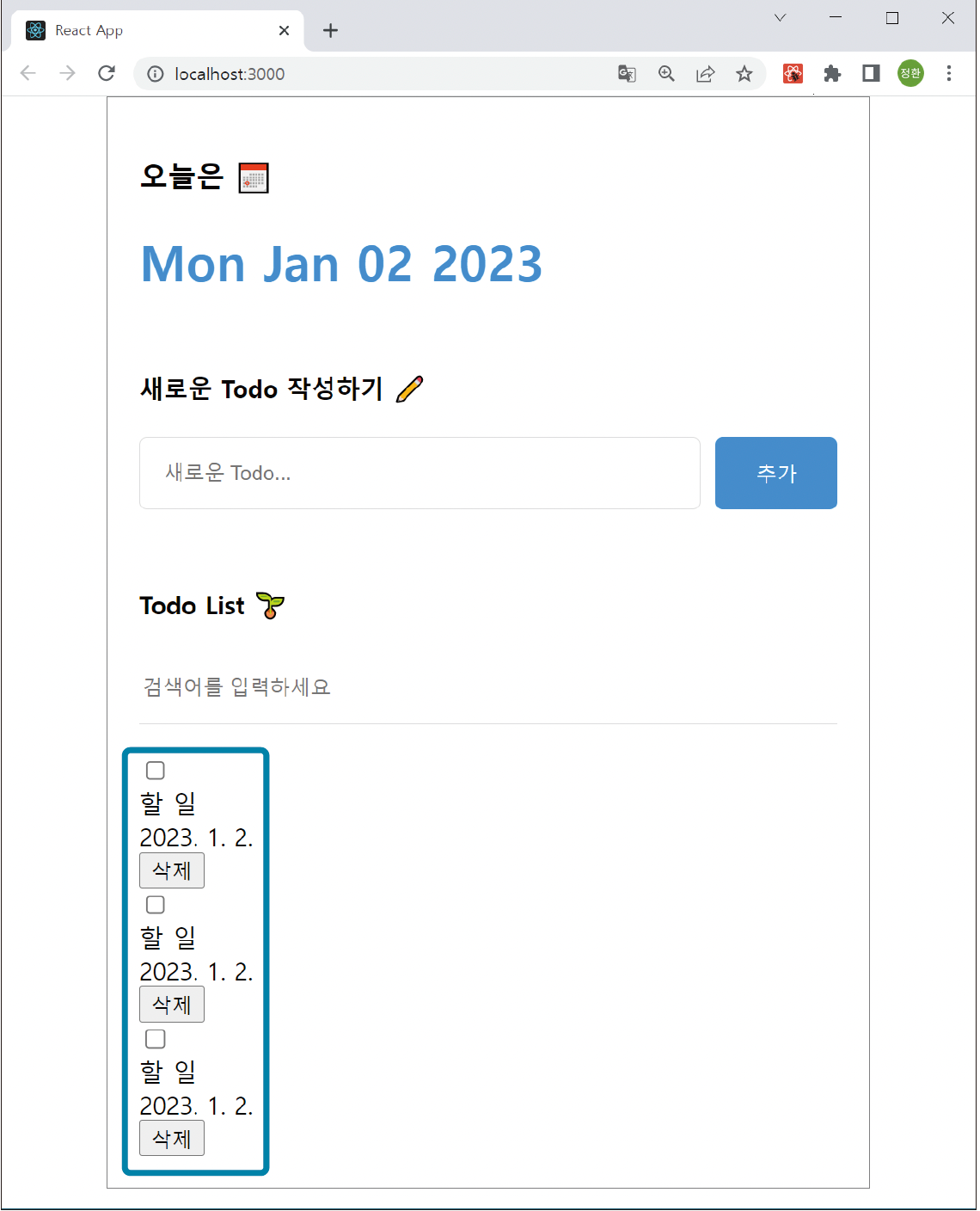
};

export default TodoList;

**①** 여러 개의 할 일 아이템을 리스트로 보여줄 <div> 태그 요소를 배치합니다. 지금은 리스트로 보여 줄 할 일 아이템이 없습니다. 따라서 임시로 그 역할을 수행할 컴포넌트 TodoItem을 3개 배치합니다.

3개의 TodoItem을 TodoList 컴포넌트의 자식으로 배치했습니다.

저장한 다음, 페이지에서 렌더링 결과를 확인합니다.

TodoItem 컴포넌트 3개 배치

결과를 보니 리스트로 배치한 3개의 아이템은 간격이 없어 답답 해보입니다.

아이템 사이에 적절한 간격을 주려면 부모 컴포넌트인 TodoList에서 여러 개의 TodoItem을 감싸고 있는 ‘list\_wrapper’ 요소에 스타일링을 적용해야합니다.

TodoList.css 파일에 다음 내용을 추가합니다.

**src/component/TodoList.css**

**Copy**

(...)

TodoList .list\_wrapper {

display: flex;

flex-direction: column;

gap: 20px;

}

저장한 다음, 개발자 도구에서 스타일링이 바르게 적용되었는지 렌더링 결과를 확인합니다.

[Element] 탭에서 class="list\_wrapper"를 찾아 클릭하면 할 일 리스트가 모두 하이라이트되는 것을 확인할 수 있습니다.

TodoItem 컴포넌트에 flex 속성 적용

이제 TodoItem 개별 요소의 스타일링을 위해 TodoItem.css에서 다음과 같은 스타일 규칙을 작성합니다.

**src/component/TodoItem.css**

**Copy**

*/\* 할 일 아이템 박스 스타일 적용 \*/*

.TodoItem {

display: flex;

align-items: center;

gap: 20px;

padding-bottom: 20px;

border-bottom: 1px solid rgb(240, 240, 240);

}

*/\* 체크박스를 감싼 박스에 스타일 적용 \*/*

.TodoItem .checkbox\_col {

width: 20px;

}

*/\* 할 일 텍스트를 감싼 박스에 스타일 적용 \*/*

.TodoItem .title\_col {

flex: 1;

}

*/\* 할 일 아이템 등록 시간을 감싼 박스에 스타일 적용 \*/*

.TodoItem .date\_col {

font-size: 14px;

color: gray;

}

*/\* 삭제 버튼에 스타일 적용 \*/*

.TodoItem .btn\_col button {

cursor: pointer;

color: gray;

font-size: 14px;

border: none;

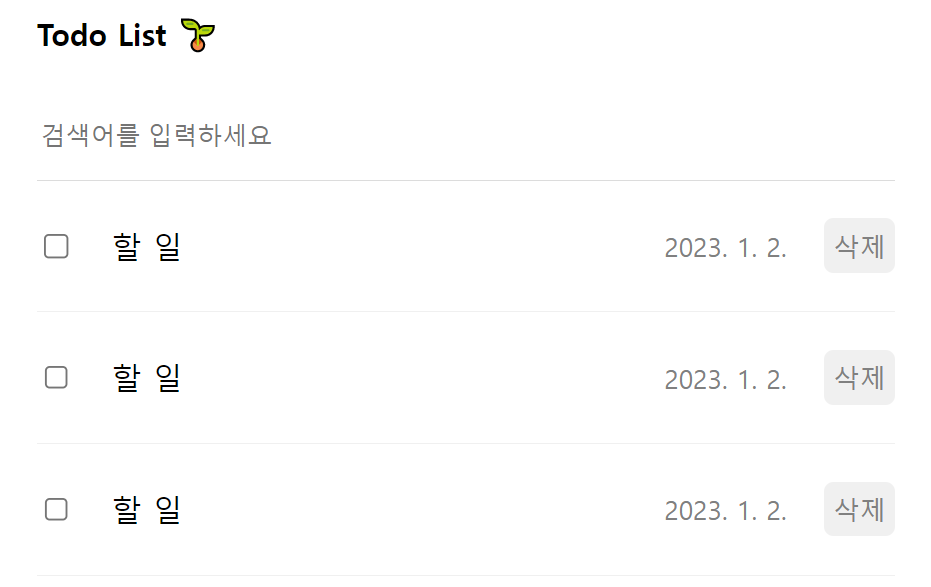
border-radius: 5px;

padding: 5px;

}

마지막으로 렌더링 결과를 확인합니다.

이렇게 [할 일 관리] 앱의 UI 구현을 모두 완료하였습니다. 이제 App 컴포넌트에서 경계를 확인할 필요가 없으니, App.css의 border 속성은 제거 또는 주석 처리합니다.

TodoItem 컴포넌트의 스타일 지정

**src/App.css**

**Copy**

.App {

max-width: 500px;

width: 100%;

margin: 0 auto;

box-sizing: border-box;

padding: 20px;

*/\* border: 1px solid gray; <- 삭제하거나 주석 처리 하세요 \*/*

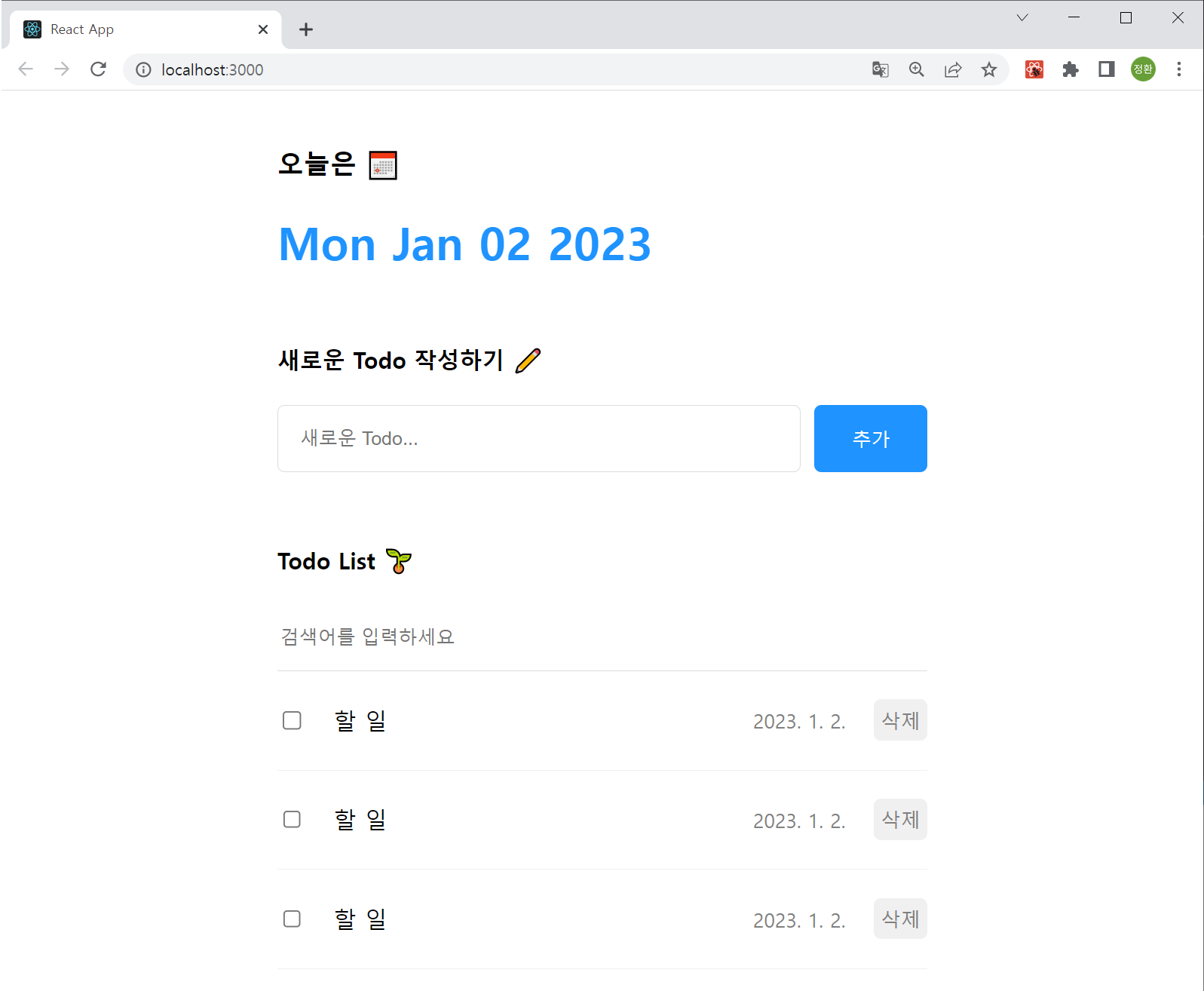
display: flex;

flex-direction: column;

gap: 30px;

}

최종적으로 완성된 UI는 다음과 같습니다.

[할 일 관리] 앱의 최종 UI

UI 구현을 흔히 ‘퍼블리싱’ 또는 ‘UI 개발’이라고 합니다. UI 개발은 데이터를 가공하고 상태를 관리하는 구현 기능과 더불어 프론트엔드 엔지니어의 기본 소양 중 하나입니다. 가볍게 여기지 말고 꾸준히 학습하면서 충분한 경험을 쌓는 게 꼭 필요합니다.