React

목차

- **React** 란?
- 웹사이트에 **React** 추가
- **create-react-app**
- 컴포넌트 만들기**(JSX)**
- **React** 라이프 사이클
- 이벤트 핸들링
- 데이터 다루기
- **Ajax** 호출
- **React**로 사고하기

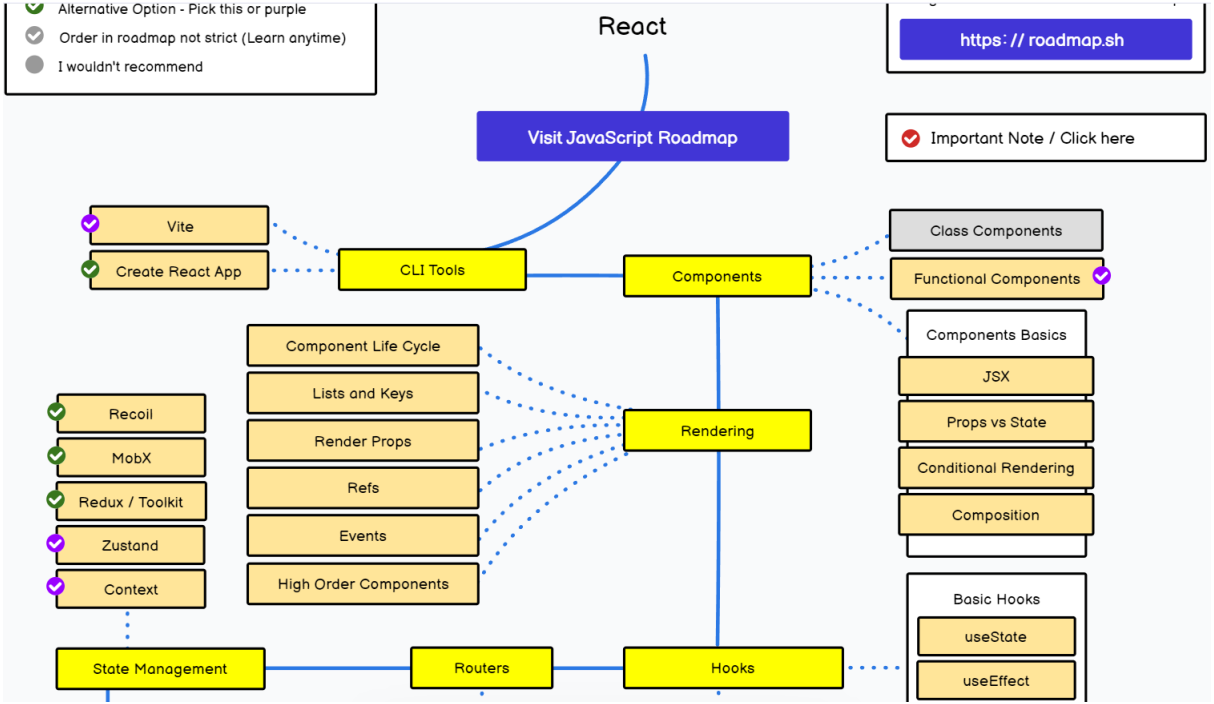
리액트는 사용자 정의 태그를 만드는 기술이다

React 란

필요기술

- **JavaScript**
- **Node.js - JavaScript** 런타임 플랫폼 **https://nodejs.org**
- **VS Code** - 개발 도구 **https://code.visualstudio.com**
- **Git Bash** - 버전 관리와 리눅스 명령 **https://git-scm.com**

React 로드맵



참고사이트 : https://github.com/adam-golab/react-developer-roadmap/blob/master/README-KO.md
https://roadmap.sh/

React 개요

- 사용자 인터페이스를 만들기 위한 **JavaScript** 라이브러리
  - 서버 사이드 렌더링과 클라이언트 사이드 렌더링 모두 지원하여 적용의 범위가 매우 넓음
- **2013년 페이스북(메타)이 오픈 소스로 공개**
- **Declarative(선언형) Programming**
  - JSX 문법을 통해 마치 HTML 처럼 직관적으로 개발이 가능
  - 렌더링 작업의 대부분을 추상화하여 개발자가 UI 디자인에 집중할 수 있게 함.
- **Component-Based**
  - 사용자 인터페이스의 일부를 나타내는 논리적인 코드 조각인 컴포넌트들을 사용하여 전체 UI를 구성
  - React는 하나의 버튼, 인터페이스의 일부분, 혹은 애플리케이션의 사용자 인터페이스 전체를 다룰 수 있음

React 개요

- **SPA**
  - 싱글 페이지 애플리케이션(Single-page application, SPA)은 하나의 HTML 페이지와 애플리케이션 실행에 필요한 JavaScript와 CSS 같은 모든 자산을 로드하는 애플리케이션입니다. 페이지 또는 후속 페이지의 상호작용은 서버로부터 새로운 페이지를 불러오지 않으므로 페이지가 다시 로드되지 않습니다.
- **Learn Once, Write Anywhere : react native, next.js**
  - React는 프레임워크가 아니고, 심지어 웹에만 사용할 수 있는 것도 아닙니다. React는 다른 프레임워크/라이브러리들과 함께 특정한 환경을 렌더링하는 데 사용됩니다.
  - React Native는 모바일 애플리케이션을 만드는 데 사용
  - React 360은 가상 현실 애플리케이션을 만드는 데 사용
  - ReactDOM은 웹을 만들기 위해서 사용
  - 다른 프로젝트와 혼용해서 사용할 수 있기 때문에 개발이 완료된 서비스에도 적응이 가능함

프로그래밍 패러다임

- **Declarative(선언형) 프로그래밍**
  - WHAT : 어떻게 해야하는지 보다 무엇과 같은지 설명하는 프로그램
  - Logic Programing, Functional Programing

```
function addOne (arr) {
    return arr.map((i) => i+1);
}
```

```
// 배열의 각 요소에 1을 더하기
```

- **imperative(명령형)프로그래밍**
  - HOW : 컴퓨터가 수행하는 절차를 일일이 코드로 작성
  - Structured Programing, Procedure Programing

```
function addOne (arr) {
    let results = [];
    for(let i=0; i<arr.length; i+=1){
        results.push(arr[i]+1);
    }
    return results;
}
```

참고사이트 : https://mangkyu.tistory.com/111

프로그래밍 패러다임

DECLARATIVE(선언형) 프로그래밍

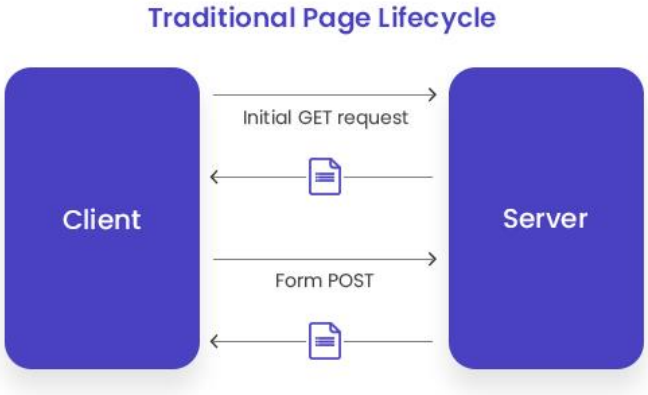
```
const header = <h1>Hello, World!</h2>; // jsx
ReactDOM.render(header,document.getElementById('root'));
```

IMPERATIVE(명령형)프로그래밍

```
const root = document.getElementById('root');
const header = document.createElement('h1');
const headerContent = document.createTextNode('Hello, World!');

header.appendChild(headerContent);
root.appendChild(header);
```

SPA

- Single Page Application으로 필요한 데이터만 비동기로 받아와 동적으로 현재화면에 다시 렌더링
- 다수의 페이지를 표시하는데 있어서 과거의 전통적인 방식으로 페이지 전환을 수행하지 않고 마치 하나의 페이지인 것처럼 처리하는 기술
- 페이지 전환을 할 때도 깜빡임 없이 부드럽게 넘어가 사용자의 몰입도를 높임.



개발환경 구축

- **node.js 설치**
- **yarn 설치** 패키지 관리 도구
  - 설치가이드 : https://yarnpkg.com/getting-started/install
  - npm을 이용한 설치
  - corepack을 이용한 설치
- **크롬브라우저에 React DevTools 설치**
  - components 탭이 추가됨
  - props, hooks 확인
  - console 창에서 this.$r 전역객체로도 확인가능함
- **vscode 확장팩 설치**
  - 확장팩 설치 -> 파일 생성 -> rafce 엔터 (자동완성)

```
> npm install -global yarn
> npm i -g yarn

> node -v
v18.16.1
> yarn -v
1.22.19
```

```
> corepack enable   //관리자 권한으로 실행
> yarn -v
1.22.19
```

웹사이트에 React 추가

전체 소스

```
<script src="https://unpkg.com/react/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

<div id="root"></div>

<script type="text/babel">
    const element = <h1>Hello, world!</h1>
    const domContainer = document.querySelector("#root");

    // ReactDOM.render(element, domContainer);       ← React version 17
    const root = ReactDOM.createRoot(domContainer);  ← React version 18
    root.render(element);
</script>
```

참고사이트 https://ko.legacy.reactjs.org/docs/add-react-to-a-website.html

1단계: HTML 파일에 DOM 컨테이너 설치

- ** 태그를 추가.**
  - 이 태그가 바로 React를 통해 원하는 내용을 표시할 수 있는 위치가 됨.

```
<!-- ... existing HTML ... -->

<div id="root"></div>

<!-- ... existing HTML ... -->
```

2단계: 스크립트 태그 추가하기

- **react, react-dom**
  - 개발모드 → React 개발자 도구, devtools 프로파일러, 핫 모듈 교체, 진단 등 다양한 기능을 활용 디버깅 가능

```
<script src="https://unpkg.com/react/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
```

  - 프로덕션 모드 → Javascript 및 기타 리소스의 압축하여 코드 크기를 줄이고 성능이 훨씬 빠름

```
<script src="https://unpkg.com/react@18/umd/react.production.min.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@18/umd/react-dom.production.min.js" crossorigin></script>
```

- **babel**
  - → JSX 사용 가능함

```
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
<script type="text/babel" src="Hello.js"></script>
```

3단계: React 컴포넌트 만들기

- **Hello.js**

```
function Hello() {
  return <h1>Hello, world!</h1>;
}

const domContainer = document.querySelector("#root");
const root = ReactDOM.createRoot(domContainer);        // React 앱을 생성
root.render(<Hello />);                                // 컴포넌트 추가
```

html과 React 비교

한번 더 설명 듣고싶음

HTML

```
<button onclick ="setIsClciked (true)">
   Click here!
</button>

<script>
 let isClicked = false ;

 function setIsClciked (state ) {
  isClicked = state ;
  event.target.innerText =
      isClicked ? "Clicked" : "Click here!"
 }
</script>
```

React

```
<div id="root"></div>

<script type="text/babel">
  function MyButton(props) {
    const [isClicked, setIsClicked] = React.useState(false);

    return (
      <button onClick={() => setIsClicked(true)}>
        {isClicked ? "Clicked" : "Click here!"}
      </button>
    );
  }

  const domContainer = document.querySelector("#root");
  const root = ReactDOM.createRoot(domContainer);
  root.render(<MyButton />);
</script>
```

ES 모듈 방식

```
<div id="root"></div>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<script async src="https://ga.jspm.io/npm:es-module-shims@1.7.0/dist/es-module-shims.js"></script>
<script type="importmap">
{  "imports": {  "react": "https://esm.sh/react?dev",
                 "react-dom/client": https://esm.sh/react-dom/client?dev   }
}
</script>
<script type="text/babel" data-type="module">
  import React, { useState, useEffect } from 'react';
  import { createRoot } from 'react-dom/client';

  function App() {
    return (       <h1>Hello, world!</h1>    );
  }
  const domContainer = document.querySelector("#root");
  const root = createRoot(domContainer );
  root.render(<App />);
</script>
```

Class vs function

- **function 방식**

```
function MyButtonFunction(props) {
  const [isClicked, setIsClicked] =
                          React.useState(false);

  return React.createElement(
    "button",
    { onClick: () => setIsClicked(true) },
    isClicked ? "Clicked" : "Click here!"
  );
}
```

- **class 방식**

```
class MyButtonClass extends React.Component {
  constructor(props) {
   super(props);
  this.state = { isClicked: false };
 }

 render() {
  return React.createElement(
   "button",
   { onClick: () => this.setState({isClicked:true}),
   this.state.isClicked ? "Clicked" : "Click here!"
  );
 }
}
```

```
const domContainer = document.querySelector("#root");
const root = ReactDOM.createRoot(domContainer);
root.render(React.createElement(MyButtonClass));
```

JavaScript에서 HTML을 직관적으로 다룰 수 있는 표현식

JSX

JSX

- **JavaScript eXtension**
- **JavaScript를 확장한 문법으로 문자열과 HTML의 결합**
- **JavaScript + XML(HTML)**
- **JavaScript에서 HTML을 직관적으로 다룰 수 있는 표현식**
- 동적으로 **HTML** 요소를 표현하기 때문에 변수를 이용한 연산과, 함수 호출 결과 등으로 상황에 맞게 유연한 렌더링을 할 수 있음
- **React 의 Element(컴포넌트 렌더링 요소)**를 생성
- 컴포넌트는 **JSX** 문법을 통해 **Element**의 집합을 반환

JSX

- **JSX에서 camelCase 사용, class는 className, tabindex는 tabIndex**
- **주입 공격을 방지**
- **브라우저는 JSX를 기본적으로 지원하지 않으므로 번들러 및 기타 도구 필요함**
  - Webpack, Parcel, Snowpack
  - Babel

```
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
<script type="text/babel" src="Hello.js"></script>
```

JSX

- **JSX는 후에 JavaScript 코드로 컴파일됨**
  - JSX를 React.createElement() 호출로 컴파일

```
const element = <h1>Hello, World! <h1>;
```

create-react-app

툴체인

- **React를 배우고 있거나** 아니면 **새로운** 싱글 페이지 **앱**을 만들고 싶다면 Create React App.
- **서버 렌더링 Node.js 웹사이트를 만들고 있다면** Next.js을 시도해보세요..
- **고정적인 콘텐츠 지향적 웹사이트를 만들고 있다면** Gatsby를 시도해보세요..
- **컴포넌트 라이브러리** 혹은 **이미 있는 코드 베이스에 통합을 한다**면 더 유연한 툴체인.
  - Neutrino는 webpack의 장점과 React의 단순함과 미리 설정된 앱과 컴포넌트를 합친 것입니다.
  - Nx는 풀스택 모노레포 개발을 위한 도구이며, React, Next.js, Express 등을 기본적으로 지원합니다.
  - Parcel은 React와 함께 사용할 수 있고 빠르고 설정이 필요 없는 웹 애플리케이션 bundler입니다.
  - Razzle은 서버 렌더링 프레임워크며 설정이 필요 없지만, Next.js보다 다루기 쉽습니다.
- **커맨드-라인 인터페이스(command-line interface; CLI) 툴**

React 프로젝트 만들기

- **create-react-app** 툴체인

```
> npx create-react-app my-app
> cd my-app
> npm start
```

```
http://localhost:3000/
```

'npx'는 npm 5.2+ 버전의 패키지 실행 도구입니다.

- **vite**

```
> mkdir ~/workspace-react
> cd ~/workspace-react
> npm create vite@latest my-app
   -- --template react
> cd my-app
> npm install
> npm run dev
```

1. my-app 폴더 생성
2. 앱의 기능에 필수적인 npm 패키지들을 설치.
3. 애플리케이션을 시작하고 서비스하기 위한 스크립트를 작성.
4. 기본적인 앱 아키텍처를 정의하는 파일과 디렉토리의 구조를 만듬.
5. 컴퓨터에 깃이 설치되어있다면, 디렉토리를 깃 레포지토리로 초기화.

create-react-app

D:\react>**npx create-react-app moz-todo-react**

Need to install the following packages:
  create-react-app@5.0.1
Ok to proceed? (y)
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.

Creating a new React app in D:\react\moz-todo-react.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

added 1491 packages in 2m

258 packages are looking for funding
  run `npm fund` for details

Initialized a git repository.

Installing template dependencies using npm...

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

Created git commit.
Success! Created moz-todo-react at D:\react\moz-todo-react
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

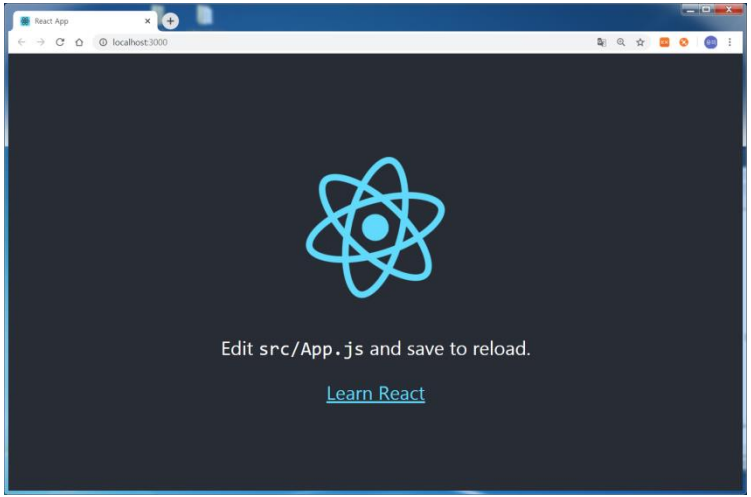
  cd moz-todo-react
  npm start

Happy hacking!

create-react-app

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.
Starting the development server...

One of your dependencies, babel-preset-react-app, is importing the
Compiled successfully!

You can now view moz-todo-react in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



애플리케이션 구조

```
moz-todo-react
├── README.md
├── node_modules
├── package.json
├── package-lock.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

컴포넌트 만들기

- 스크립트는 중괄호 안에 작성
- key로 사용할 속성이 없으면 {index}
- 주석 **{/* 여기는 주석 */}**

```
const Comp01 = () => {
   const style = {
      color:'green', fontSize:'50px',
      backgroundColor:'yellow'
   }
   return (
      <div>
         {/* 여기는 주석 */}
         <h1 style={style}>첫번째 컴포넌트</h1>
         { products.map(p => <Product
                              key={p.name}  prd={p}/>)     }

      </div>
   )
}
export default Comp01;
```

컴포넌트 만들기

- **function component**
- **export component**

```
function HelloComponent() {
  return (
    <div>
      <h1>Hello Component</h1>
    </div>
  )
}

export default HelloComponent
```

https://ko.react.dev/learn/your-first-component

컴포넌트 사용하기

- **import component**
- **tag**
- **props로 데이터 전달**

```
import HelloComponent from './HelloComponent'

<HelloComponent></HelloComponent>

<HelloComponent />
```

데이터 전달

- 부모가 자식 컴포넌트에게 **props**로 전달
- 자식은 **props**로 받음

```
<HelloComponent value="hello"></HelloComponent>

function HelloComponent(props) {
  return (
    <div>
      <h1>Hello Component : {props.value}</h1>
    </div>
  )
}
```

데이터 흐름

- **리액트 개발 방식 = 컴포넌트 단위 개발**
  - **페이지를 만들기 이전에 컴포넌트를 먼저 만들고 조립함**
- **상향식(bottom-up) 앱 개발 ➡ 테스트가 쉽고 확장성이 좋음**
- **컴포넌트는 컴포넌트 바깥에서 props를 이용해 데이터를 속성처럼 전달받을 수 있다.**
  - 즉, 데이터를 전달하는 주체는 부모 컴포넌트
  - 컴포넌트는 props를 통해 전달받은 데이터가 어디에서 왔는지 알지 못한다.
- **데이터 흐름은 하향식 (top-down)**
- **React는 단방향 데이터 흐름(One-way data flow)을 따른다.**

데이터 흐름

- **상태 위치 정하기**
  - 두 개의 서로 다른 컴포넌트가 특정 상태에 영향을 받는 경우?
    ➔ 공통 소유 컴포넌트(공통의 부모)를 찾아 그 곳에 상태를 위치시켜야 한다.
- **역방향 데이터 흐름 추가**
  - 부모 컴포넌트의 상태가 하위 컴포넌트에 의해 변하는 경우?
    ➔ "State 끌어올리기" 로 해결
    상태를 변경시키는 함수(handler)를 하위 컴포넌트에 props로 전달
    https://ko.reactjs.org/docs/lifting-state-up.html

React로 사고하기

React로 사고하기

- 1단계 : UI를 컴포넌트 계층 구조로 나누기
- 2단계: React로 정적인 버전 만들기
- 3단계: UI state에 대한 최소한의 표현 찾아내기
- 4단계: State가 어디에 있어야 할 지 찾기
- 5단계: 역방향 데이터 흐름 추가하기

https://ko.react.dev/learn/thinking-in-react
https://ko.reactjs.org/docs/thinking-in-react.html

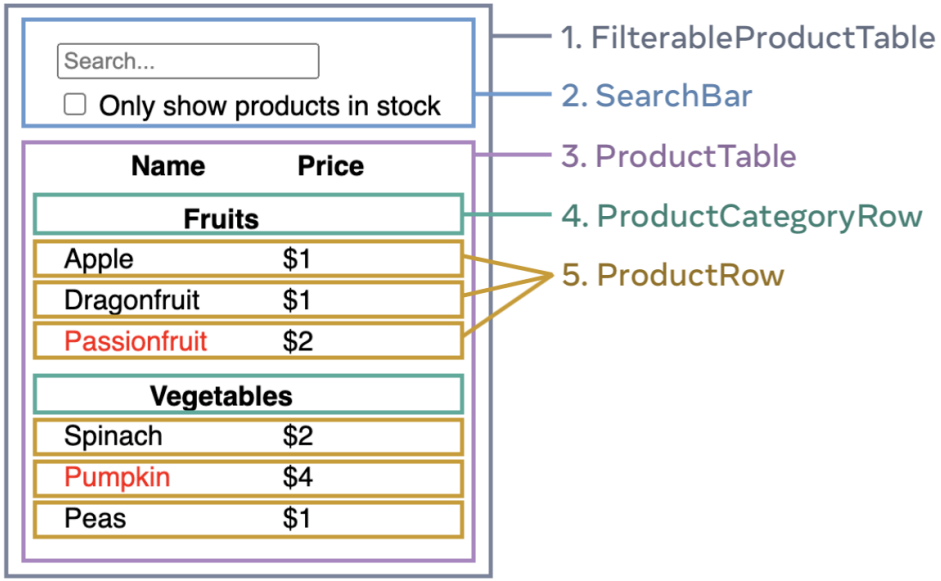
React로 사고하기

- **Step 1: UI를 컴포넌트 계층으로 쪼개기**

Search...

Only show products in stock

Name	Price
Fruits	
Apple	$1
Dragonfruit	$1
Passionfruit	$2
Vegetables	
Spinach	$2
Pumpkin	$4
Peas	$1



React로 사고하기

- **Step 2: React로 정적인 버전 구현하기**
  - 데이터 모델을 렌더링하는 앱의 정적인 버전을 만들기 위해 다른 컴포넌트를 재사용하고 props를 이용하여 데이터를 넘겨주는 컴포넌트를 구현할 수 있습니다. props는 부모가 자식에게 데이터를 넘겨줄 때 사용할 수 있는 방법입니다.
  - 앱을 만들 때 계층 구조에 따라 상층부에 있는 컴포넌트 (즉 FilterableProductTable부터 시작하는 것)부터 하향식(top-down)으로 만들거나 혹은 하층부에 있는 컴포넌트 (ProductRow)부터 상향식(bottom-up)으로 만들 수 있습니다.
  - 간단한 예시에서는 보통 하향식으로 만드는 게 쉽지만,
  - 프로젝트가 커지면 상향식으로 만들고 테스트를 작성하면서 개발하기가 더 쉽습니다.

React로 사고하기

```
function ProductCategoryRow({ category }) {
  return (
    <tr>
      <th colSpan="2">{category}</th>
    </tr>
  );
}

function ProductRow({ product }) {
  const name = product.stocked ? (
    product.name
  ) : (
    <span style={{ color: "red" }}>{product.name}</span>
  );

  return (
    <tr>
      <td>{name}</td>
      <td>{product.price}</td>
    </tr>
  );
}
```

```
function ProductTable({ products }) {
  const rows = [];
  let lastCategory = null;

  products.forEach((product) => {
    if (product.category !== lastCategory) {
      rows.push(
        <ProductCategoryRow
          category={product.category}
          key={product.category}
        />
      );
    }
    rows.push(<ProductRow product={product} key={product.name} />);
    lastCategory = product.category;
  });
  return (
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>{rows}</tbody>
    </table>
  );
}
```

React로 사고하기

```
function SearchBar() {
  return (
    <form>
      <input type="text" placeholder="Search..." />
      <label>
        <input type="checkbox" /> Only show products
in stock
      </label>
    </form>
  );
}
```

```
function FilterableProductTable({ products }) {
  return (
    <div>
      <SearchBar />
      <ProductTable products={products} />
    </div>
  );
}

const PRODUCTS = [
  { category: "Fruits", price: "$1", stocked: true, name: "Apple" },
  { category: "Fruits", price: "$1", stocked: true, name: "Dragonfruit" },
  { category: "Fruits", price: "$2", stocked: false, name: "Passionfruit" },
  { category: "Vegetables", price: "$2", stocked: true, name: "Spinach" },
  { category: "Vegetables", price: "$4", stocked: false, name: "Pumpkin" },
  { category: "Vegetables", price: "$1", stocked: true, name: "Peas" },
];

const domContainer = document.querySelector("#root");
const root = ReactDOM.createRoot(domContainer);
root.render(<FilterableProductTable products={PRODUCTS} />);
```