1. 서버사이드 렌더링(ServerSide Rendering)이란 무엇인가?

서버사이드 렌더링이란 검색엔진을 통해서나 주소창을 통해서 들어왔을 경우 혹은 새로고침 시 즉, 첫 방문시에는 브라우저가 프론트 서버에 요청을 하면 바로 응답하지 않고, 프론트 서버는 백엔드 서버로부터 데이터를 받아온 다음에 프론트 서버에서 데이터 + HTML 파일을 합쳐서 화면을 그려준다. 이후 이것을 브라우저에게 전달해 화면에 그려내므로 로딩창 없이 완전한 화면을 볼 수 있다.

브라우저 => 프론트서버 => 백엔드서버 => 프론트서버 => 브라우저

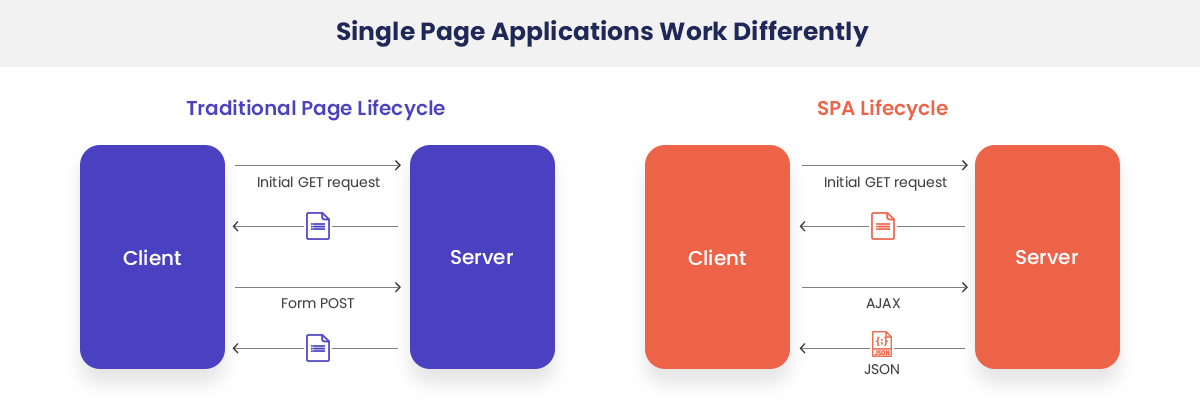
1. SPA란 무엇이고 어떤 장점이 있는가?

SPA는 Single Page Application의 준말이며 단일 페이지로 구성된 웹 어플리케이션을 말한다. SPA는 화면이동 시에 필요한 데이터를 서버사이드에서 HTML으로 전달받지 않고(서버사이드 렌더링 X), 필요한 데이터만 서버로부터 JSON으로 전달받아 동적으로 렌더링한다.

**기존 어플리케이션과 SPA의 차이**

기존 어플리케이션은 화면이동 시에 화면 이동에 필요한 HTML을 서버사이드에서 받아서 처음부터 다시 로딩하기 때문에 시간이 걸린다.

반면, SPA에서는 화면 구성에 필요한 모든 HTML을 클라이언트가 갖고 있고 서버사이드에는 필요한 데이터를 요청하고 JSON으로 받기 때문에 기존의 어플리케이션에 비해 화면을 구성하는 속도가 빠르다.



출처: <https://www.excellentwebworld.com/what-is-a-single-page-application/>

**장점**

* 하나하나 화면 전체를 렌더링할 필요가 없기 때문에 화면이동이 빠르다.
* 화면에 필요한 부분의 데이터만 받아서 렌더링하기 때문에 처리과정이 효율적이다.
* 유저의 입장에서 사용하기 편리하다.

**단점**

* 처음 화면을 로딩할 때, 모든 화면이 미리 준비되어 있어야 하기 때문에 로딩에 시간이 걸린다.
* 어플리케이션을 구현하는데 보다 시간이 걸리며 복잡하다.

출처 : <https://velog.io/@josworks27/SPA-%EA%B0%9C%EB%85%90>

1. 리액트에 사용되는 디자인패턴은 어떤 것들이 있는가. 각각 설명하시오.

실습 예시를 통해 리액트에 사용되는 디자인패턴에 대해 이해해보자.

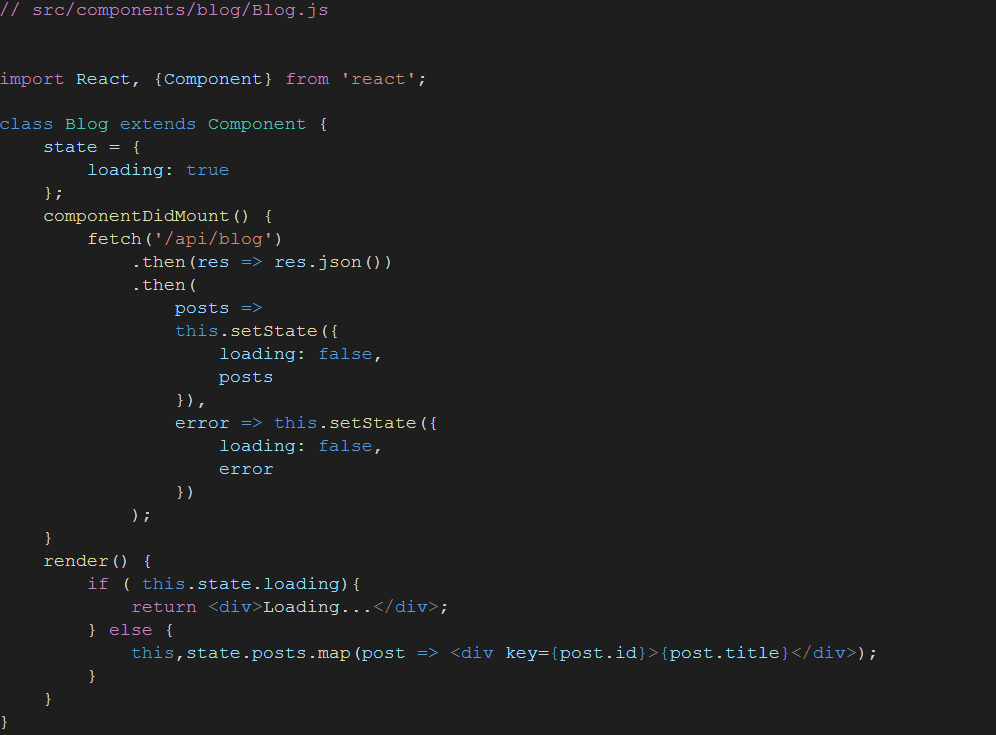
React 디자인 패턴의 종류는 각양 각색이지만 우선 많은 곳에서 쓰여지고 있는

**Container Presenter Pattern**에 대하여 알아보겠다.

실습 목표

Call an API to get the latest blog posts. Show a ‘loading’ status in the meantime. When the call to the API is resolved, render the blog posts.

우선 패턴 사용 없이 진행해 보자.



state에서 loading의 초기 상태는 true

Component가 마운트되면 api를 부르고, 완료되면 포스팅을 불러오고, 로딩 : false하고, 포스팅을 올린다. 에러가 있으면 loading: false하고 에러를 보여준다.

render function에서는 loading state가 true이면, Loading 문구를 띄우고 loading state가 false이면, 포스팅 제목들을 화면에 렌더링한다.

결국에는 데이터를 불러오고, 블로그 포스팅을 보여주는 작업을 하는 거다. 그런데 크게 보았을 때 이는 좋지 않다. 같은 장소에서 데이터를 불러오고 보여주면 안된다. 왜냐하면 presentation과 logic을 분리해야 되기 때문이다.

여기까지가 보통 리액트를 짜는 방법이다.

이를 위해 등장한 것이 디자인 패턴이며, 그 중에서도 container + presenter 방법에 대해 소개해보려 한다.

Container 컴포넌트 + Presenter 컴포넌트

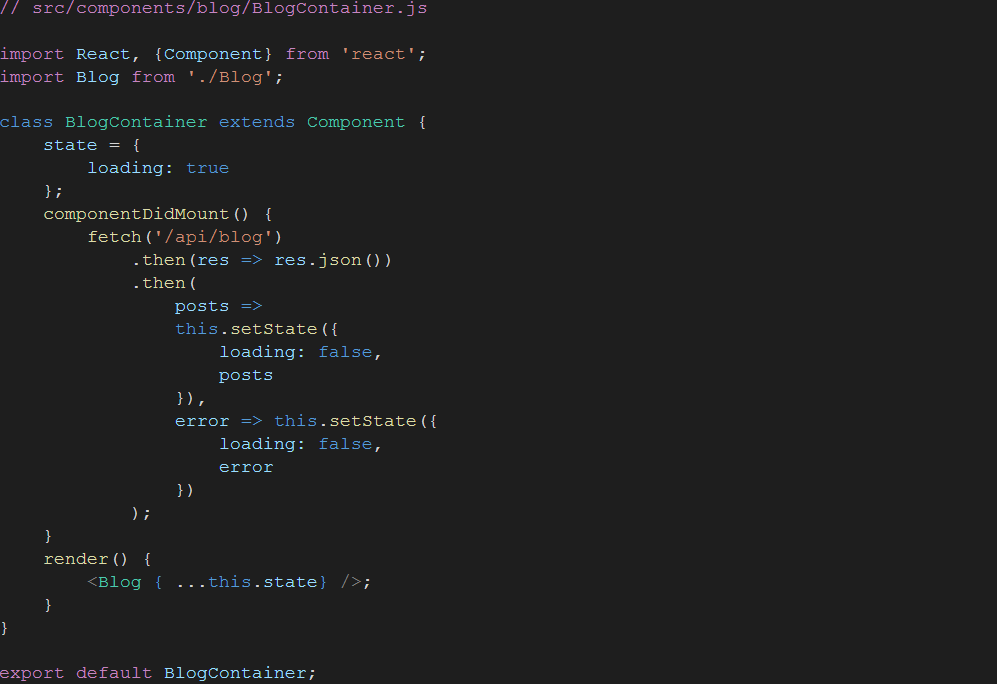
Container

Logic : API Requests, errors etc...

Presenter

Data comes from props. No logic. Only UI

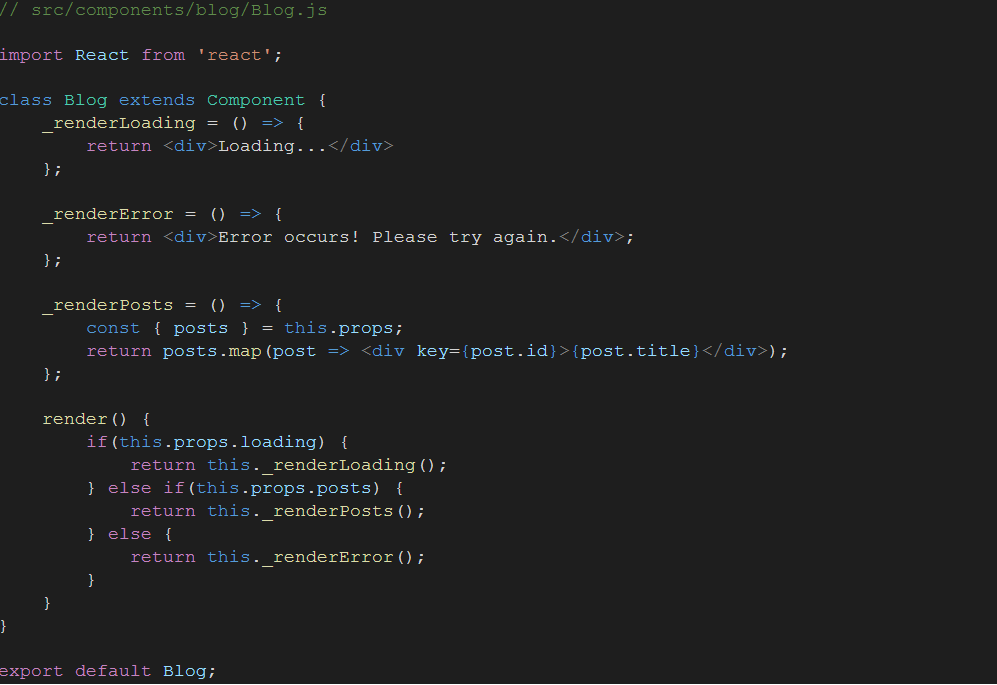
container 컴포넌트부터 살펴보자.



기존의 Blog.js와 거의 다 똑같은데 차이점은 이 컴포넌트는 ‘블로그 컨테이너’인 것이다. 블로그 컨테이너는 블로그 컴포넌트를 render, return 한다. (state를 유지하면서...) 로딩, 포스트, 에러를 렌더할 것이다.

자, 그러면 presenter에서는 props에서 데이터가 나오고, 로직이 없고, 오직 UI만 이야기한다. state에 대한 이야기가 없다. 블로그 컨테이너가 state변화를 이야기한다. presenter는 logic, 논리 이야기는 없다. 오직 유저 인터페이스만 이야기할 뿐이다.

presenter 컴포넌트는 이렇게 생겼다.



props에서 데이터를 갖고오고, state 변경 없고, 로딩중이면, 렌더 로딩 함수를 불러와서 로딩창을 화면에 렌더링하고 로딩 중이 아니면 props에 post가 있으면 포스팅을 렌더링하고 로딩도 아니고, 포스트도 아니면 에러창을 화면에 렌더링한다.

간단히 정리하면 다음과 같다.

1. Container는 state도 이해하고, api, 리덕스를 부를 수도 있다. state를 변경. 리덕스 액션 불러오고, 액션 디스패치하고,, 등등
2. Presenter에서는 props에 따라 데이터가 어떻게 보이는지 알려준다.

이것이 Container Presenter Pattern이다.

핵심 : 데이터 그리고 데이터 출력 방법을 분리하는 것!

출처 : <https://dev-yong.tistory.com/12>