[2022-12-06] 2반 실습 파일

학습 목표

- React 앱을 Server Side로 구현하는 방법을 배우고, 제작한 웹 서비스가 검색 엔진에 노출되는 방법을 학습합니다.
- 백엔드와 프론트엔드를 연동하는 것을 익힙니다.
- 2차 프로젝트를 대비하기 위한 10-12주차 과정 내용을 복습하는 실습을 진행합니다.

SSR

이건 이전 수업에서 CSR, SSR 장단점을 비교할 때, CSR 단점이 있었습니다.

바로 client-side-rendering은 SEO에 적합한 방법은 아니라는 것인데요,

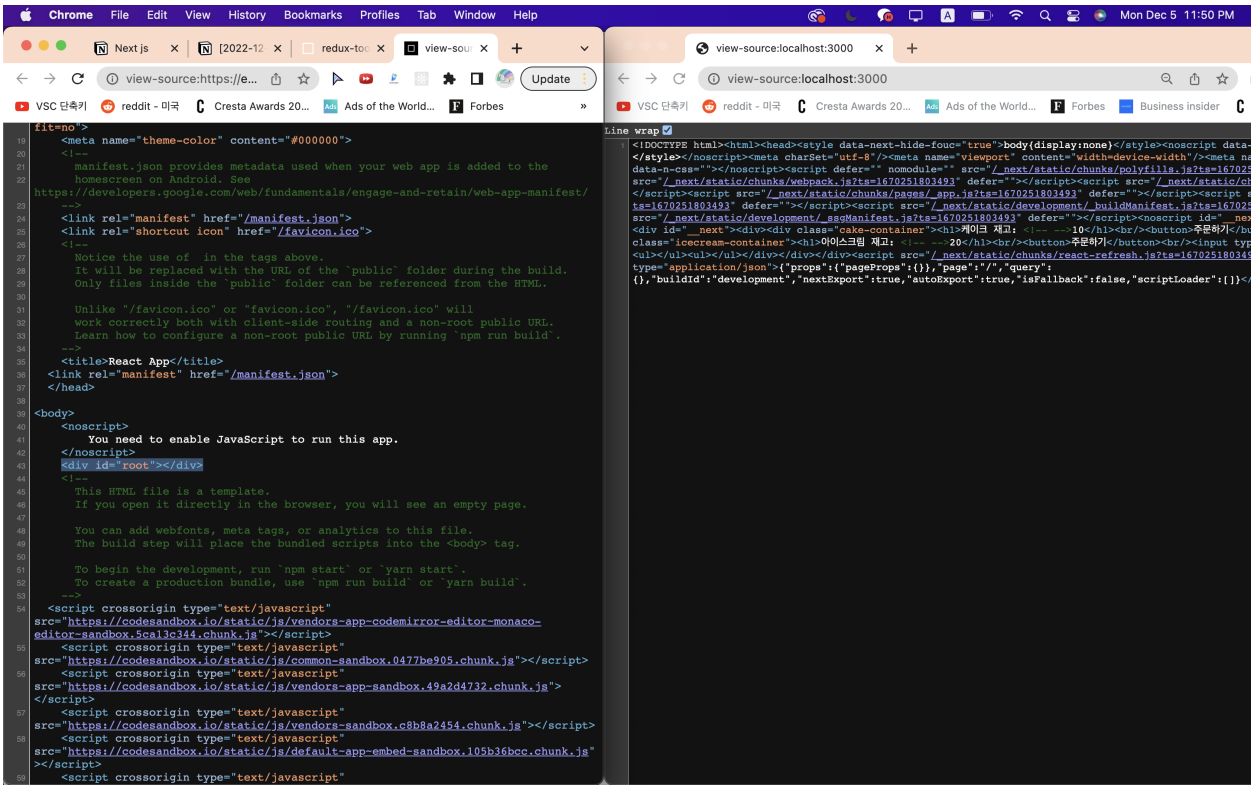
리액트 server-side-rendering을 가능하게 하도록 사용할 수 있는 툴이 있다고 아주 살-짝 언급했었는데, 바로 nextJS를 활용해서 구현할 수 있답니다. (제대로 배우려면 상당히 많은 시간이 걸리기 때문에 아주 간단하게 구현하는 모습만 보여드리겠습니다!)

이전에 리액트로 만들었던 빵집!을 next.js로 똑-같이 만들어봤습니다!

> npx create-next-app@latest

리액트빵집바로보기

넥스트빵집바로보기



view-source를 통해 웹사이트 소스보기 기능을 활용하면 CRA으로 만들어진 어플리케이션은 어떠한 컴포넌트가 들어가있지 않은 상태, nextjs로 만든 프로젝트는 정적인 코드들이 소스코드에 반영된 걸 확인할 수 있어요. 자세히 보시면 설정한 초기값이 바인딩된 상태를 관찰할 수 있어요.

#이걸 저희는 SSR과 별개로 SSG (static-site generation)이라고 부르는데 빌드타임에 미리 html을 생성해줍니다→ 그래서 SEO에 유리하답니다.

지금 이미 페이지 로딩 시 정적인 html부분은 로딩이 된 상태이고, 여기서 더하여 로딩 중인 상태를 보여주고 싶지 않고 모든 데이터가 로딩이 된 후에…! 즉 백엔드에서 페이지를 모두 구성한 후에 렌더링하고 싶어하는 경우, getServerSideProps()라는 함수를 활용해서 next.js를 사용하면 선택적으로 ssr을 할 수 있습니다.

이 함수를 생성해주지 않으면 그대로 client-side-rendering이 됩니다.

SEO

우선 우리 웹사이트를 검색 가능하게 하려면 먼저 검색엔진에 우리 사이트를 등록해야해요. (네이버 - 웹마스터도구, 구글 - search console) 그러면서 소스코드에 같이 포함시킬 파일이 2개 더 있습니다.

robots.txt이라는 파일을 작성해서 어떤 검색엔진 크롤러가 크롤링 할 수 있게 허락 / 크롤링을 가능하게 할 루트와 못하게 할 루트 명시 (어드민페이지)..

sitemap.xml 웹사이트 목차 작성… (웹사이트 입력시 자동으로 만들어주는 서비스도 있어요) 크롤러가 크롤링 과정에서 쉽게 발견되지 않는 웹페이지도 색인 시킴)

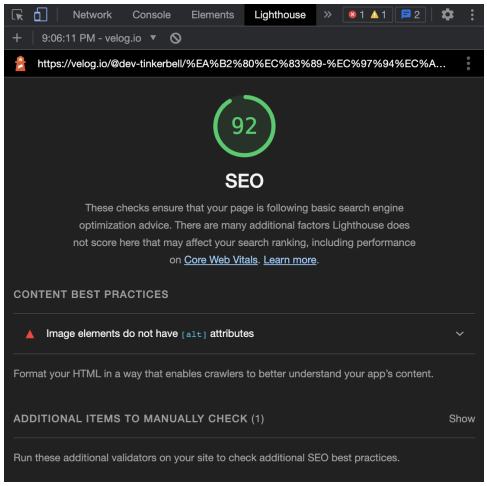
그냥 이런게 있구나 그리고 이런 역할을 하구나만 알아 두시고 필요할 때 보시면 됩니다!

우선 이렇게 웹사이트 검색을 가능하게 하고, 이젠 검색 엔진 최적화를 시켜서 검색 결과의 상위 노출이 되어야하는데요, 구글 검색 엔진 최적화를 위해 엄청 잘 정리해둔 영상이 있어서 영어로 되어있고 잘 번역해준 곳이 없어서 슬쩍 정리해봤어요. 제가 직접 경험해본 건 아니지만 영상 제작자의 많은 시도를 해보셨고 + 효과를 본 많은 사람들을 봤을 때 공유해도 좋을 것 같아 살짝 정리해보았습니다.

키워드 잘 뽑아야한다고 합니다. 먼저 3가지 키워드를 뽑아서 (≠3가지 단어. 키워드는 더 긴 단어일 수 있음) 웹 컨텐츠 + 필요한 태그 등을 채워줘요.

- <title>, , <h1>, <h2><h3>, <meta>(description, author,keywords…), <header> 등 그런데 외울 필요는 없어요 왜냐면

개발자 도구 > Lighthouse > 에서 SEO 점수 확인이 가능하거든요. 절대적인 기준은 될 순 없지만 키워드가 들어가야하는 태그에 넣지 않았으니 넣으라고 알람을 준답니다.



자 이제 다시 SEO 방법론으로 들어가서 키워드를 뽑는 기준이

- 너무 경쟁적이지 않은?! 수요가 어마어마하지 않은 키워드? (== 틈새시장을 노려라)

키워드를 뽑을 때 다음과 같이 참고할 수 있는 웹사이트가 있습니다.

answerthepublic: 블로그, sns 등 사용자들이 자주하는 질문 모음집

explodingtopics: 토픽 트렌드 단어

문제 풀이

1. 음악장르 맞추기 게임

계획 세우기 먼저! 3개의 라우트가 필요하겠다는 생각이 들었습니다.

(1) 메인홈페이지, (2) 제목을 클릭했을 때 라우트, (3) 그 페이지에서 장르 클릭했을 때 선택결과를 보여주는 라우트

(1) "/": Home 컴포넌트 렌더링

(2) "music/:<<노래이름/name>>": 원데이터셋을 보니 제목이 쫙~ 장르별로도 쫙~ 각 이름들을 구분지을 수 있는 유니크한 값?이 보이지 않아서 노래 이름을 url 내 param으로 구분해주자라는 생각. (유저가 음악 이름을 클릭했을 때 이곳으로 라우팅되도록 하자). Home.jsx로 가 이름 클릭하는 곳 링크 걸어주기 고고

이렇게 파라미터를 추가해서 데이터를 보내주는 방법이 있을 수 있고~ post요청으로 서버에 데이터를 보낼 수 있지만~ 로그인 처럼 credential 민감한 데이터는 당연히 url param으로 보낸다면 문제가 있겠죠?

(3) 세번째 페이지에서도 현재 음악 이름 + 선택한 장르가 필요해 -> 둘 다 파라미터로 받아올 수 있겠다는 생각과 함께!
// "music/:<<노래이름/name>>:<<선택한장르/genre>>"로 받아오면 어떨까?

▼ 문제 정답

```
// App.jsx

import './app.css';
import Home from './Component/Home';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import Music from './Component/Music';
import Genre from './Component/Genre';
```

```
export default function App() {
  // 지시사항을 수행할 수 있도록 수정해보세요.
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
// 그런데 exact path="/"를 쓰지 않으면 '/music/:name'에 종속되어 컴포넌트가 여러개 렌더링되어야하는
거 아닌가요?
// Routes로 감싸주면 이미 exact 속성을 사용해주지 않아도 기본적으로 정확히 일치하는 라우트만 렌더링해줍니
다.
// (Routes vs Switch: https://blog.woolta.com/categories/1/posts/211)
        <Route path="/music/:name" element={<Music />} />
        <Route path="/music/:name/:genre" element={<Genre />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}

function NotFound() {
  return (
    <div>
      <h1>페이지를 찾을 수 없습니다.</h1>
      <Link to="/">Home</Link>
    </div>
  );
}
```

```
// Home.jsx

import * as MUSIC from '../data/music';
import { Link } from 'react-router-dom';

export default function Home() {
  // 지시사항을 수행할 수 있도록 수정해보세요.
  // music/:name 형식으로 링크 걸어주기
  return (
    <div>
      <h1>Home</h1>
      <h1>음악을 골라주세요.</h1>
      <div className="music-list">
        <ul>
          {MUSIC.dataSet1.map(
            // 실행 후... 어랏? 링크가 안걸려있네?! 라우터 만들어야겠다! -> App.jsx
            (
              item
            // index를 Key prop으로 전달해주지 않은 이유?
            // 어떤 배열에 중간에 어떤 요소가 삽입될때 그 중간 이후에 위치한 요소들은 전부 index가 변
경
            // React는 key가 동일 할 경우, 동일한 DOM Element를 보여주지만 달라지는 경우 리액트가
 헷갈려서 잘못된 순서로 element를 렌더링할 수 있다고 합니다
```

```
  // 그래서!! 배열 요소가 수정이 일어나지 않을 경우 + 정렬, 필터 요소로 순서가 바뀌지 않는 경우 + 쓸만한 적정한 key가 없는 경우 사용할 수 있어용
            ) => (
              <li key={Math.random().toString()}>
                <Link to={`/music/${item}`}>
                  <div>{item}</div>
                </Link>
              </li>
            )
          )}
        </ul>
        <ul>
          {MUSIC.dataSet2.map(item => (
            <li key={Math.random().toString()}>
              <Link to={`/music/${item}`}>
                <div>{item}</div>
              </Link>
            </li>
          ))}
        </ul>
        <ul>
          {MUSIC.dataSet3.map(item => (
            <li key={Math.random().toString()}>
              <Link to={`/music/${item}`}>
                <div>{item}</div>
              </Link>
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
}
```

```
// Music.jsx

import { Link, useParams } from 'react-router-dom';

export default function News() {
  // 지시사항을 수행할 수 있도록 수정해보세요.
  const { name } = useParams(); // param을 받아오자
  //   console.log('name', name); // 오브젝트 형태로 들어가는구나~
  return (
    <div>
      <h1>선택한 음악에 장르를 골라보세요.</h1>
      <h3>{name ? '선택한 음악: ' + name : ''}</h3>
      <ul>
        <li>
          <Link to={`/music/${name}/classic`}>Classic</Link>
        </li>
        <li>
```

```
          <Link to={`/music/${name}/jazz`}>Jazz</Link>
        </li>
        <li>
          <Link to={`/music/${name}/rock`}>Rock</Link>
        </li>
        <li>
          <Link to="/">Back</Link>
        </li>
      </ul>
    </div>
  );
}
```

```
// Genre.jsx

import * as MUSIC from '../data/music';
import { useParams, Link, useNavigate } from 'react-router-dom';

export default function Genre() {
  // 지시사항을 수행할 수 있도록 수정해보세요.
  const { genre, name } = useParams();

  const navigate = useNavigate(); // 이전 페이지로 돌아가기. Link는 target url을 명시해주는데
 useNavigate은 지금 당!장! 어디로 가라!

  const genreSwitch = genre => {
    if (genre === 'classic') {
      // classic이면 MUSIC데이터셋에서 classic안에 있는지 점검하는 구나
      return check(MUSIC.classic);
    } else if (genre === 'rock') {
      return check(MUSIC.rock);
    } else {
      return check(MUSIC.jazz);
    }
  };

  // 배열.indexOf(값)해서 값이 없으면 -1 있으면 그 값의 인덱스 리턴
  const check = data =>
    data.indexOf(name) === -1 ? '틀렸습니다.' : '정답입니다.';

  // <Link to="/">Back</Link>
  return (
    <div>
      <h1>{genreSwitch(genre)}</h1>
      <div>
        <Link to="/" onClick={() => navigate(-1)}>
          Back
        </Link>
      </div>
    </div>
```

```
  );
}
```

---

2. 장바구니 만들기

또 다른 리둑스 문제 등장! 두둥!

gif를 보고 딱 드는 생각… 유저가 상품을 선택하면 선택한 상품이 장바구니에 담기겠고, 그 장바구니 내 있는 상품들이 상태값으로써 리둑스로 관리가 되겠구나 생각이 들었습니다.

아오…여기선 redux-toolkit이 아니라 redux를 그대로 사용했지만 저는 redux-toolkit > redux 우선순위를 이렇게 생각합니다. 그래서 redux 코드 이해할 정도 공부하되, redux-toolkit을 사용해도 되지 않을까?! 조심스레 생각해봅니다

그런데 엄청 어렵지는 않아요. useSelector(저장된 store에 저장된 상태값을 불러와줌), useDispatch(그 상태값을 업데이트 하기 위한 대행자를 불러와줌) 훅이 없을 때는 더 다음과 같이 리덕스만들 사용하기 위해서 3가지 단계를 거쳤어야 했거든요 (하지만 다루지 않을 겁니다)

1. mapStateToProps, mapDispatchToProps를 정의한다.
2. 정의한 상태값과 액션들을 해당 컴포넌트에 props로 내려준다.
3. mapStateToProps, mapDispatchToProps를 컴포넌트와 connect로 연결한다.

어김없이 등장하는 뼈대 두둥

이전에 redux-toolkit을 사용할때는 createSlice를 통해 action creator를 한번에 만들고, 여러 reducer를 한번에 combine할 수 있었죠?! 그냥 redux 라이브러리 쓸 때는 조금 달라요.

```
// ===== reducer1.js

function reducer1(state, action){
  switch (action.type) {
    case <<액션1>>: {
      return ... // non-mutate
    }
```

```
  }
}

export default reducer1

// ===== actions.js

const <<action>> = (payload) =>
  return {
        type: "",
        payload,
  }


{//===== reducer.js (reducer가 여러개인 경우)

import {combineReducers} from "redux";


const rootReducer = combineReducers({
  reducer1,
  reducer2
  ...
})

export default rootReducer;


// ===== index.js

const store = createStore(rootReducer)

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>
)

// ===== 사용하기

import {useSelector, useDispatch} from "react-redux"
import <<Action>> // 정의한 action import

const App = () => {
  const state = useSelector(state=>state)
  const dispatch = useDispatch()
  return (
    <>
    <h1>{state}</h1>
    <button onClick={()=>dispatch(<<Action>>}></button>
    </>
  )
}
```

접근 방법~

코드 먼저 쭉 확인하고… main.jsx 음~ 스마트폰 혹은 노트북 보러가기 버튼을 다이내믹하게 보여주는 코드구나~

notebook, smartphone.jsx 내 클릭된 아이템들을 상태값을 담는 그 스토어에 기록해서 shopping.jsx에서 보여주면 되겠군~ check했을 때 uncheck 했을 때 우선 두가지 액션이 있을 수 있겠다~

checkIn할 때는 기존 상태값에 새로운 값을 만들어 추가해주자 (어떤 구조가 될진 모르겠지만). store에 저장된 값을 화면상 보여주기 위한 용도이니까, 상품명 + 가격 + 평균배송일 이런걸 payload로 받아와 오브젝트 구조로 만들어야겠다, checkOut 할 때는 기존 상태값에서 선택된 값을 제거해주자. 데이터셋을 보니 id가 유니크한 값이네. payload로 id만 받아와도 되겠다.

이 두 생각을 합쳐서 받아온 id, name, price, delivery값들을 리스트 형태로 저장하자는 계획을 세웠습니다.

▼ 문제 정답

```
// reducer.js

// 지시사항을 수행할 수 있는 reducer를 만드세요.
function reducer(state, action) {
  switch (action.type) {
    case 'CheckIn': {
      return [
        ...state,
        {
          id: action.payload.id,
          name: action.payload.name,
          price: action.payload.price,
          delivery: action.payload.delivery,
        },
      ];
    }
    case 'CheckOut': {
      return state.filter(item => item.id !== action.payload);
    }
    default:
      return state;
  }
```

```
}

export default reducer;
```

```
// actions.js

// 지시사항을 수행할 수 있는 action을 만드세요.
export const checkIn = (payload) => {
    return {
        type: "CheckIn",
        payload, // 여기 id, name, price, delivery
    }
}

export const checkOut = (payload) => {
    return {
        type: "CheckOut",
        payload // 여기 id
    }
}
```

```
// Shopping.jsx

import { useSelector } from 'react-redux';

export default function Shopping() {
  // 지시사항에 맞춰 적절히 store를 다시 사용하세요.
  const store = useSelector(state => state); // reducer가 하나니까~!

  // table html https://developer.mozilla.org/en-US/docs/Web/HTML/Element/td
  // tr: table row, th: table header, td: table data
  return (
    <div className="shopping">
      <h2>장바구니</h2>
      <table>
        ...
      </table>
      <h3>
        총 가격{' '}
        {store
          .reduce(
            (sum, val) => (sum += Number(val.price.replace(/[^0-9]/g, ''))),
            0
          )
          .toLocaleString('ko-KR') + '원'}
      </h3>
    </div>
  );
}
```

```
// 배열.reduce((누적값, 현잿값, 인덱스, 요소) => { return 결과 }, 초깃값);
// let temp = [1,2,3,4]
// console.log(temp.reduce((acc,curr,idx)=>acc+curr))

// "123,000원".replace(/[^0-9]/g, '') <- /g: global. 매칭되는 모든 substring에 대해
// 0-9가 아닌것 "^"

// toLocaleString 돈 천단위마다 콤마
// ko-KR은 왜냐하면 베트남 돈 같은 경우는 ,안쓰고 .을 사용함
```

```
// Smartphone.jsx

import { smartphoneData } from '../data/smartphone';
import { useSelector, useDispatch } from 'react-redux';
import { checkIn, checkOut } from '../redux/actions';

export default function Smartphone() {
  //지시사항을 수행하세요.

  // 유저가 check 했을 때 useDispatch 함수가 액션을 보내주도록~
  // 그리고 그 check 여부가 화면에 나타나도록~
  const store = useSelector(state => state);
  const dispatch = useDispatch();

  // 유저가 체크/언첵을 했을 때 실행시킬 함수를 만들자

  const onChange = e => {
    if (e.target.checked) {
      //   console.log(e.target.value); // dispatch로 보내줄 값을 받아와야해! 어떻게? input
 value에 넣어 가져오자
     dispatch(checkIn(JSON.parse(e.target.value))); // 직렬화된 값을 다시 자바스크립트 내장
 오브젝트로 읽을 수 있도록 parsing 해주기
    } else {
      dispatch(checkOut(JSON.parse(e.target.value).id));
    }
  };

  return (
    <div className="table--container">
      <table>
        <thead>
          <tr>
            <th></th>
            <th>상품명</th>
            <th>가격</th>
            <th>평균배송일</th>
          </tr>
        </thead>
        <tbody>
          {smartphoneData.map(item => {
```

```
            return (
              <tr key={item.id}>
                <td>
                  <input
                    data-testid="input"
                    type="checkbox"
                    onChange={onChange}
                    value={JSON.stringify(item, null, 2)} // 아이템 오브젝트를 value에 담아서 전달하겠다. 세번째 파라미터로 들여쓰기 줄 수 있어
                    // 이렇게 웹으로 데이터 전달해주기 위해서는
                    // 직렬화(특정 언어의 내장 타입의 데이터를 외부에 전송하기 용이하도록 문자열로 변환하는 과정)가 필요해 stringify: JS value -> JSON 문자열로 변경
                    checked={
                      store.filter(stItem => stItem.id === item.id).length === 0
                        ? false
                        : true
                    }
                  />
                </td>
                <td>{item.name}</td>
                <td>{item.price}</td>
                <td>{item.delivery}</td>
              </tr>
            );
          })}
        </tbody>
      </table>
    </div>
  );
}
```

```
// Notebook.jsx

import { smartphoneData } from '../data/smartphone';
import { useSelector, useDispatch } from 'react-redux';
import { checkIn, checkOut } from '../redux/actions';

export default function Smartphone() {
  //지시사항을 수행하세요.

  // 유저가 check 했을 때 useDispatch 함수가 액션을 보내주도록~
  // 그리고 그 check 여부가 화면에 나타나도록~
  const store = useSelector(state => state);
  const dispatch = useDispatch();

  const onChange = e => {
    if (e.target.checked) {
      //   console.log(e.target.value);
      dispatch(checkIn(JSON.parse(e.target.value)));
    } else {
      dispatch(checkOut(JSON.parse(e.target.value).id));
```

```
    }
  };

  return (
    <div className="table--container">
      <table>
        <thead>
          <tr>
            <th></th>
            <th>상품명</th>
            <th>가격</th>
            <th>평균배송일</th>
          </tr>
        </thead>
        <tbody>
          {smartphoneData.map(item => {
            return (
              <tr key={item.id}>
                <td>
                  <input
                    data-testid="input"
                    type="checkbox"
                    onChange={onChange}
                    value={JSON.stringify(item, null, 2)}
                    checked={
                      store.filter(stItem => stItem.id === item.id).length === 0
                        ? false
                        : true
                    }
                  />
                </td>
                <td>{item.name}</td>
                <td>{item.price}</td>
                <td>{item.delivery}</td>
              </tr>
            );
          })}
        </tbody>
      </table>
    </div>
  );
}
```