React Admin 기능 분석

React Admin 기능 및 특징

어드민을 구현하기 위한 프론트엔드 프레임 워크

- 거의 모든 요소가 **커스터 마이즈** 가능 (API fetch, Authentication, Layout, dashBoard, Route, Redux)

- 데이터 프로바이더라는 어댑터로 유연하게 Api와 연동이 가능
  따라서 서로 다른 스펙을 사용하는 여러개의 백엔드 서버와 연동이 가능 (Rest, GraphQL, Web Socket)

- 관리할 리소스에 대한 페이지를 신속하게 부츠트래핑이 가능 (유용한 기본 컴포넌트를 다수 제공)
  리소스간의 연결이 자연스럽게 사용가능함

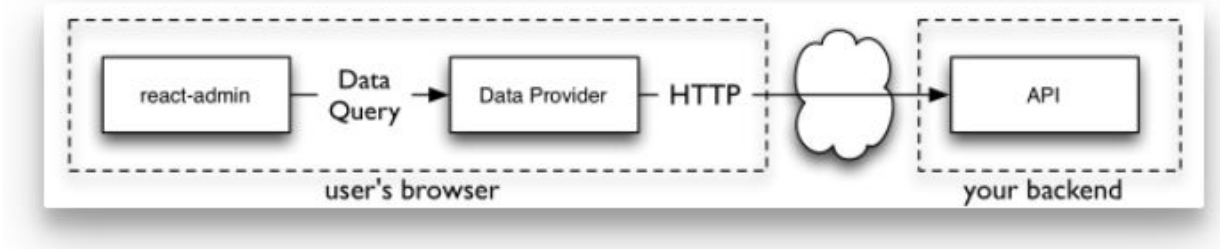
기본 개념

- 리소스: crud가 가능한 하나의 api 엔드포인트
- 뷰 : 각 자원을 관리(crud)하기 위해 구현해야 할 페이지 컴포넌트 (list, show, create, edit)
- 액션 : create, update, delete, refresh 와 같이 버튼으로서 구현될 기능
- 필드 : List, Show등의 뷰에서 레코드의 프로퍼티를 read-only로 표현하기 위한 컴포넌트군
- 인풋 : create, edit 등의 뷰에서 데이터를 수정하기 위해 제공되는 컴포넌트군

Data Provider를 통한 유연한 Api Fetch

각 자원을 crud할 뷰에서 직접적으로 API를 호출하는 것이
아닌 Data Provider라는 어댑터에 위임해서 유연하게 fetch

Data Provider에 관리 자원의 crud에 대한 구현을 프로미스로 작성 (유연한 구현이 가능!!)



액션 or 라우트 이동 ⇔ DataProvider 호출

- 각 관리 자원에 대한 뷰 이동 or 액션 호출 시
  React-Admin이 DataProvider를 통해 API Fetch
- fetch 결과가 각 뷰 페이지에 전달됨 (create, list, show, edit 페이지)

```
const dataProvider = {
    getList:    (resource, params) => Promise,
    getOne:     (resource, params) => Promise,
    getMany:    (resource, params) => Promise,
    getManyReference: (resource, params) => Promise,
    create:     (resource, params) => Promise,
    update:     (resource, params) => Promise,
    updateMany: (resource, params) => Promise,
    delete:     (resource, params) => Promise,
    deleteMany: (resource, params) => Promise,
}
```

Method name	API call
`getList`	`GET http://my.api.url/posts?sort=['title','ASC']&range=[0, 24]&filter={title:'bar'}`
`getOne`	`GET http://my.api.url/posts/123`
`getMany`	`GET http://my.api.url/posts?filter={id:[123,456,789]}`
`getManyReference`	`GET http://my.api.url/posts?filter={author_id:345}`
`create`	`POST http://my.api.url/posts/123`
`update`	`PUT http://my.api.url/posts/123`
`updateMany`	Multiple calls to `PUT http://my.api.url/posts/123`
`delete`	`DELETE http://my.api.url/posts/123`
`deteleMany`	Multiple calls to `DELETE http://my.api.url/posts/123`

Response Format

Data Providers methods must return a Promise for an object with a `data` property.

Method	Response format
`getList`	`{ data: {Record[]}, total: {int} }`
`getOne`	`{ data: {Record} }`
`getMany`	`{ data: {Record[]} }`
`getManyReference`	`{ data: {Record[]}, total: {int} }`
`create`	`{ data: {Record} }`
`update`	`{ data: {Record} }`
`updateMany`	`{ data: {mixed[]} }` The ids which have been updated
`delete`	`{ data: {Record|null} }` The record that has been deleted (optional)
`deleteMany`	`{ data: {mixed[]} }` The ids of the deleted records (optional)

<Admin />과 <Resource />

```
const App = () => (
    <Admin dataProvider={jsonServerProvider('https://jsonplaceholder.typicode.com')}>
        <Resource name="posts" list={PostList} create={PostCreate} edit={PostEdit} show
        <Resource name="users" list={UserList} />
        <Resource name="comments" list={CommentList} create={CommentCreate} edit={Comme
        <Resource name="tags" />
    </Admin>
);
```

<Admin />

- 개별적인 상태, 라우팅, 컨트롤러 로직을 가지는 어드민을 생성, 설정하는 컴포넌트
- 하나이상의 리소스를 가질것으로 기대된다.
- data provider, auth provider, theme, layout등 설정이 주로 이뤄진다.

<Resource />

- 각 api 엔드 포인트를 다루기 위한 컴포넌트
- name prop을 통해 엔드 포인트의 이름을 정의

/name/	list
/name/create	create
/name/:id	edit
/name/:id/show	show

Material ui를 기반으로 구현된 다양한 컴포넌트 제공

- 각 자원의 property만을 source에 넘겨 직관적으로 api response를 표기가 가능해
  빠르게 부츠트랩핑이 가능 (Input, Field)
- 직관적인 구조로 Field와 Input을 쉽게 커스터마이즈 가능
- 기본적으로 반응형 레이아웃을 제공

```
const PublisherEdit: FC = props => {
  return (
    <Edit {...props}>
      <SimpleForm>
        <TextInput disabled source="id" />
        <TextInput source="name" />
      </SimpleForm>
    </Edit>
  )
}
```

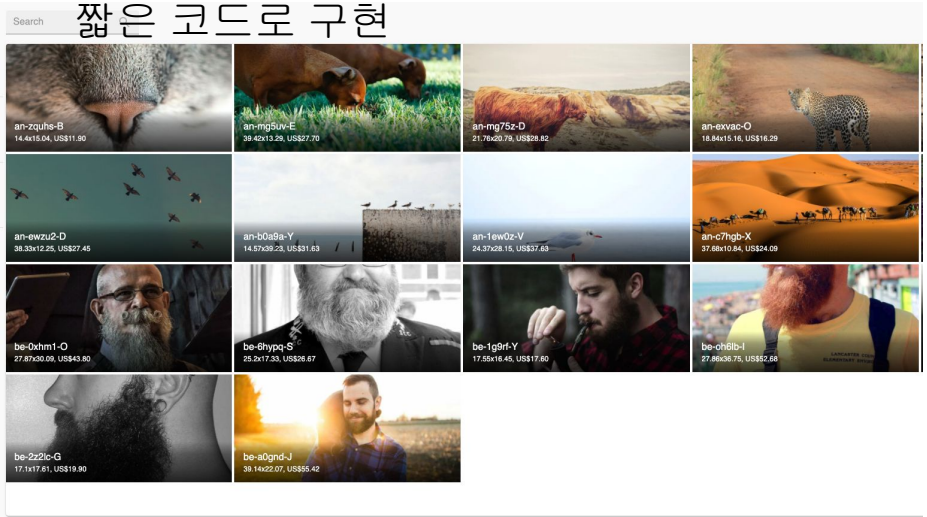
<List />를 커스터 마이즈

- 기본적으로 <List />는 <Datagrid />를 자식으로 받아 테이블을 그려서 데이터 리스트를 렌더링 한다.
- <DataGrid />는 <List />가 fetch된 데이터를 iterate하는것에 불과하므로 커스터 마이즈 가능

```
const BuyerList: FC = props => {
  return (
    <List {...props} sort={{ field: 'createdAt', order: 'DESC' }}>
      <Datagrid>
        <TextField source="id" sortable={false} />
        <TextField source="name" sortable={false} />
        <TextField source="endpoint" sortable={false} />
        <DateField source="createdAt" />
        <ShowButton />
        <EditButton />
      </Datagrid>
    </List>
  )
}
```

```jsx
<div className={classes.root}>
    <MuiGridList
        cellHeight={180}
        cols={getColsForWidth(width)}
        className={classes.gridList}
    >
        {ids.map(id => (
            <GridListTile
                component={Link}
                key={id}
                to={linkToRecord(basePath, data[id].id)}
            >
                <img src={data[id].thumbnail} alt="" />
                <GridListTileBar
                    className={classes.tileBar}
                    title={data[id].reference}
                    subtitle={
                        <span>
                            {data[id].width}x{data[id].height},{' '}
                            <NumberField
                                className={classes.price}
                                source="price"
                                record={data[id]}
                                color="inherit"
                                options={{
                                    style: 'currency',
                                    currency: 'USD',
                                }}
                            />
                        </span>
                    }
                />
            </GridListTile>
        ))}
    </MuiGridList>
```

fetch된 데이터를 iterate하는
커스텀 컴포넌트를 작성해
이미지 갤러리 형식의
리스팅이
짧은 코드로 구현



optimistic rendering

- 데이터 프로바이더에서 api response를 redux-store에 항상 저장한다.
- 따라서 캐싱에 활용가능 !!
- undo와 관련있다.
- 리스트 페이지 이동시에 유저 경험을 좋게한다.

Auth Provider를 통한 인증 커스터 마이즈

src > JS authProvider.js > login > username

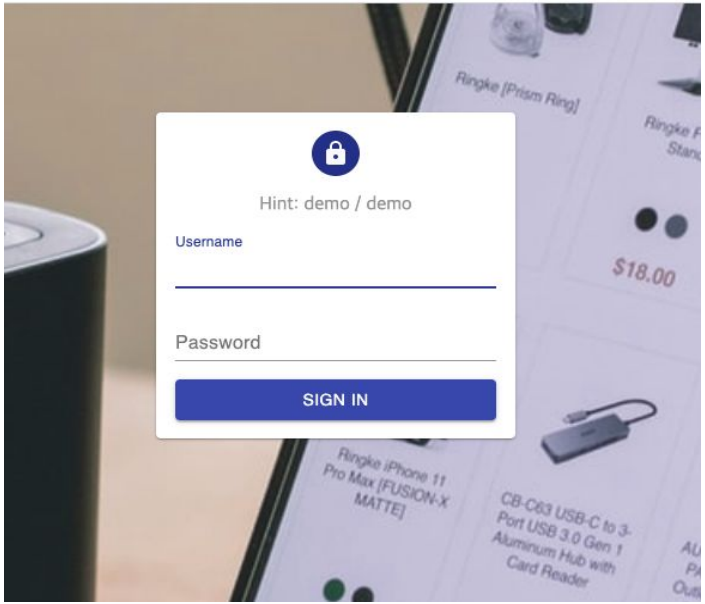
```
export default {
  login: ({ username }) => {
    // 로그인 시도시 호출
    localStorage.setItem("username", username);

    console.log("로그인을 시도 ");

    return Promise.resolve();
  },
  logout: () => {
    // 유저가 로그아웃 시도시 호출
    localStorage.removeItem("username");

    return Promise.resolve();
  },
  checkError: ({ status }) => {
    // api가 에러를 리턴시 호출
    if (status === 401 || status === 403) {
      localStorage.removeItem("username");
      return Promise.reject();
    }

    return Promise.resolve();
  },
  checkAuth: () => {
    // 내비게이션으로 이동시 호출  =>  인증을 체크

    return localStorage.getItem("username")
      ? Promise.resolve()
      : Promise.reject();
  },
  // 내비게이션 이동시 호출  =>   퍼미션/룰을 체크
  getPermissions: () => Promise.resolve()
};

```

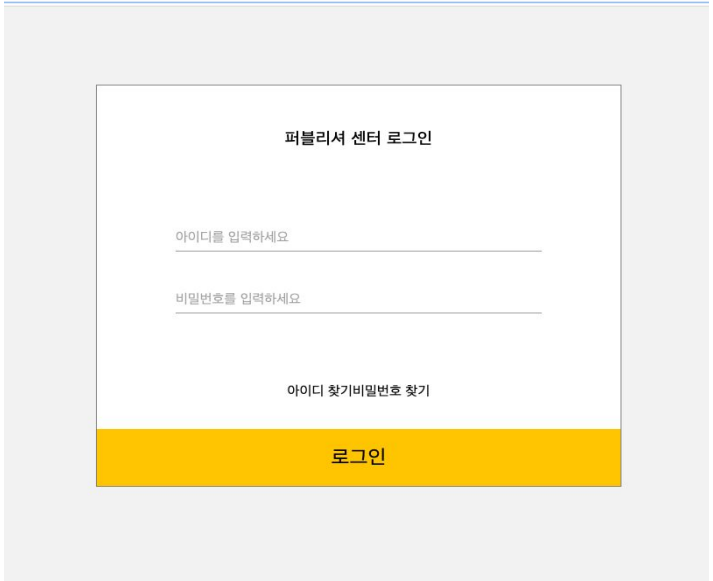
<Admin />에 authProvider를 왼쪽과 같이

넘김으로써 authentication을 각 프로덕트에 맞게 커스터마이즈

로그인 페이지 및 레이아웃 커스터마이즈

- <Admin />의 props로 기본 로그인 페이지와 레이아웃을 커스터마이즈 가능하다.

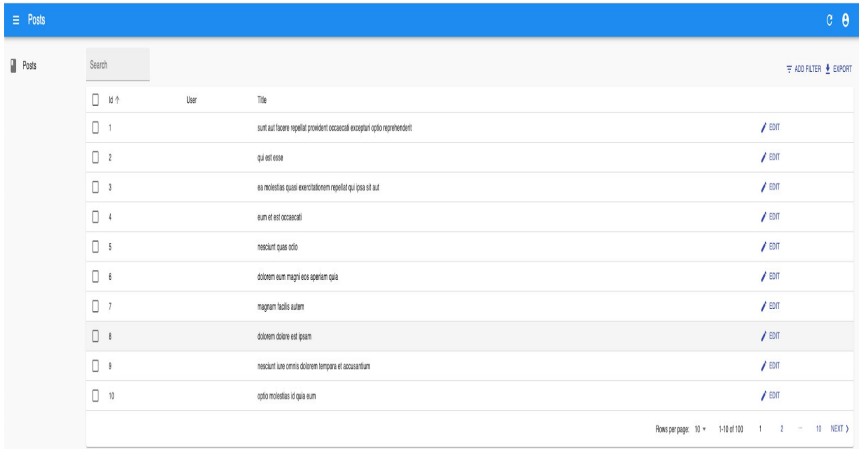


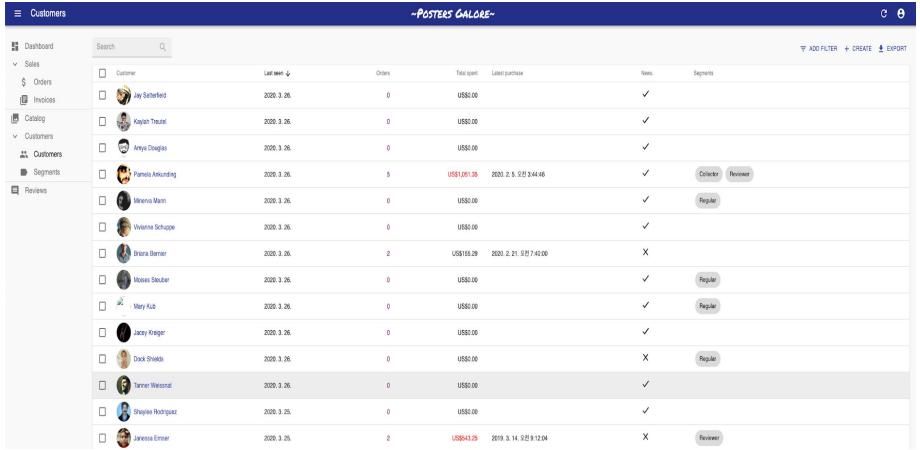


- 레이아웃은 기본적으로 상단 앱바, 사이드바, 메뉴, 메인으로 구성되어 있음
- 상단 앱바, 사이드바, 메뉴를 커스터마이즈 가능

```
export default (props: any) => {
    const theme = useSelector((state: AppState) =>
        state.theme === 'dark' ? darkTheme : lightTheme
    );
    return (
        <Layout
            {...props}
            appBar={AppBar}
            sidebar={CustomSidebar}
            menu={Menu}
            theme={theme}
        />
    );
};
```

기본 레이아웃과 커스텀된 레이아웃





라우트 커스터 마이즈

- 기본적으로 리소스에 종속된 페이지 이동은 DataProvider를 호출
- 페이지가 단일 리소스에 종속되지 않았거나 위의 기본 액션을 회피해야 할경우를 위해
  커스텀 라우트를 배열을 통해 추가할 수 있다

```
import React from 'react';
import { Route } from 'react-router-dom';
import Configuration from './configuration/Configuration';
import Segments from './segments/Segments';

export default [
    <Route exact path="/configuration" render={() => <Configuration />} />,
    <Route exact path="/segments" render={() => <Segments />} />,
];
```

커스텀 Reducer 추가 가능

- 내부 구현이 redux, redux-saga로 구성됨
- 각 프로덕트를 위한 전역 상태 관리가 필요시 reducer를 추가로 등록 가능

```
import { CHANGE_THEME } from './configuration/actions';

export default (previousState = 'light', { type, payload }) => {
    if (type === CHANGE_THEME) {
        return payload;
    }
    return previousState;
};
```

```
<Admin
    title=""
    dataProvider={dataProvider}
    customReducers={{ theme: themeReducer }}
    customRoutes={customRoutes}
>
```