[2022-12-08] 2반 실습 파일

문제 풀이

1. 사진 갤러리

사전 지식

```
// 힌트

styled.div`
    background-color: #fff4e6;
    display: grid; // 자식들을 grid == 격자로 만들겠다 컬럼 쫙 로우 쫙 설정 가능
    grid-template-columns: 1fr 1fr 1fr; // 1행크기 2행크기 …; // 크기 명시
    // == 이 친구와 동일 repeat(3, 1fr); <- *성능차이 수업시간에 질문주셨는데 찾아보니 내부에서 루프가 돌
더군요!! 일반 리스트 선언하는 것과 루프로 만드는 건 성능 차이가 있겠죵?!
    grid-template-rows: 1fr 1fr 1fr;
    grid-auto-rows: 0; // 행의 개수가 명시한 것 보다 많을 때 명시하지 않은 그 외의 행 크기
    overflow: hidden;
`;
```

▼ 문제 정답

```
import React, { useState } from 'react';
import styled from 'styled-components';

export default function Gallery() {
  const [rowCount, setRowCount] = useState(1);
  const [columnCount, setColumnCount] = useState(1);

  const handleColumnInput = event => {
    setColumnCount(event.target.value);
  };
  const handleRowInput = event => {
    setRowCount(event.target.value);
  };

  return (
    <div>
      <div id="header">
```

```
        <h1 style={{ marginTop: '10px', textAlign: 'center' }}>사진 갤러리</h1>
        <div style={{ textAlign: 'center', margin: '30px 0' }}>
          <Label htmlFor="rowCount">row 개수 설정</Label>
          <Input
            type="number"
            value={rowCount}
            onChange={handleRowInput}
            id="rowCount"
          />
          <Label
            style={{
              marginLeft: '50px',
            }}
            htmlFor="columnCount"
          >
            column 개수 설정
          </Label>
          <Input
            type="number"
            value={columnCount}
            onChange={handleColumnInput}
            id="columnCount"
          />
        </div>
      </div>
      <ImageContainer
        className="image-container"
        rowCount={rowCount} // rowCount수에 따른 grid 열/행을 보여줄 것이므로
        columnCount={columnCount}
      >
        <Image className="image" src="img/1.jpg" alt="gallery image" />
        <Image className="image" src="img/2.jpg" alt="gallery image" />
        <Image className="image" src="img/3.jpg" alt="gallery image" />
        <Image className="image" src="img/4.jpg" alt="gallery image" />
        <Image className="image" src="img/5.jpg" alt="gallery image" />
        <Image className="image" src="img/6.jpg" alt="gallery image" />
        <Image className="image" src="img/7.jpg" alt="gallery image" />
        <Image className="image" src="img/8.jpg" alt="gallery image" />
        <Image className="image" src="img/9.jpg" alt="gallery image" />
        <Image className="image" src="img/10.jpg" alt="gallery image" />
        <Image className="image" src="img/11.jpg" alt="gallery image" />
        <Image className="image" src="img/12.jpg" alt="gallery image" />
        <Image className="image" src="img/13.jpg" alt="gallery image" />
        <Image className="image" src="img/14.jpg" alt="gallery image" />
      </ImageContainer>
    </div>
  );
}

const Image = styled.img`
  width: 100%;
  height: 200px; // 이미 높이 설정
  border: 2px solid #ffa94d;
  box-sizing: border-box;
```

```
`;



const ImageContainer = styled.div`
    background-color: #fff4e6;
    display: grid;
    grid-template-columns: repeat(${props => props.columnCount}, 1fr);
    grid-template-rows: repeat(${props => props.rowCount}, 1fr);
    grid-auto-rows: 0; // 이게 마지막 원소에는 적용이 안되어 overflow로 가려줬어요!!
    grid-auto-columns: 0;
    overflow-y: hidden;
`;

const Label = styled.label`
    background-color: #fff4e6;
    border: 2px solid #f76707;
    font-size: 20px;
    folat: left;
    margin-right: 10px;
`;

// className같은 attrs 도 가능
const Input = styled('input').attrs({
  type: 'number',
})`
    width: 50px;
    font-size: 20px;
`;
```

---

2. 항공권 예매하기

코드가 길어서 아래 두개 컴포넌트로 나눠서 진행하자는 생각이 들었어요! (물론 코딩할 당시에는 요론 생각을 못하는게 당연합니다)

(1) 항공권 검색하기 컴포넌트

- 도시 데이터로 출발지 도착지 선택할 수 있도록 select, option을 만들자 (반복되니까 함수로 만듭시다!)

```
// (힌트) 오브젝트 iterate 하는 방법

const human = {
    "firstname": "강호",
    "lastname": "송"
}
const newHuman = Object.keys(human).map(k=>(k+"메롱"))
console.log(newHuman)
```

- 선택한 출발지(초기값 인천), 도착지, 날짜를 저장하는 state값을 만들자

api로 데이터 받아오자

- api로 티켓 데이터 받아오자 - 출발지, 도착지 필터링 필요합니다. 검색 결과에서 보여주려면 date, time 데이터를 년 월 일 시 분 구조로 만들어주는 과정도 필요해요. 하지만 나중에 sorting 할 때 원 데이터가 필요할 것이므로 새로운 포매팅된 날짜 시간 데이터가 담긴 새로운 formedData 키를 만들어줘야겠다고 생각했습니다.

이걸 깜빡하고 언급을 못했는데 그러면 fetch해서 받아온 데이터 useMemo로 저장해줄 수 있는 것 아니냐?! 이런 질문이 있을 것 같아서 제가 생각했을 때는 저희가 고르는 와중에 (항공권 생각했을 때) 다른 사람이 티켓을 사면서 구매 가능한 티켓이 변동될 수 있겠죠?! 그런 경우에는 검색을 해줄 때마다 업데이트된 데이터를 가져오기 우한 fetch를 진행해야해서 useMemo (특정 함수에 대한 결과값을 저장하는 용도)가 필요하지 않고 검색 할 때마다 fetch가 실행되어야할 것 같아요!

하지만 그게 아닌 고정된 데이터라면 값 자체를 메모이징 해줄 수 도 있겠죠??!

(2) 필터 및 정렬 및 결과를 보여주는 컴포넌트

항공사필터, 시간정렬, 가격정렬, 각각 선택된 값을 상태값으로 사용하자! 는 생각.

시간, 가격 정렬은 각각 3개의 옵션을 가질 수 있겠다!는 생각.

힌트:

```
// 숫자 정렬
// sort()는 텍스트 정렬 숫자 정렬 방법은 따로 있음

const tickets = [1000,15000,8000,4000]
```

```
const sortedTickets = tickets.sort((a, b) => a - b); // 오름차순

console.log(sortedTickets)


// 날짜 정렬
const tickets = ["2022-09-08T17:10:07.994Z", "2022-12-15T03:39:20.561Z", "2022-09-26T19:4
6:25.028Z"]

const sortedTickets =  tickets.sort((a, b) => {
        if (a > b) return 1;
        else if (b > a) return -1;
        else return 0;
})

console.log(sortedTickets)
```

▼ 문제 정답

```
// MakeOptionList.js

// 방법 1. Object.keys(myObject) + map
// 오브젝트의 키를 뽑아 만든 리스트를 순회합니다
function makeOptionList(dict) {
    return Object.keys(dict).map(k => (
      <option value={k} key={k}> // 오브젝트의 key값을 유일하다는 조건이 있어 key prop으로 넣어
줍니다
        {dict[k]}
      </option>
    ));
  }


// 방법2. reduce 활용 (축적값 계산 뿐 아니라 map의 대체 용도로 사용할 수 있습니다)
function makeOptionList(dict) {
    return Object.keys(dict).reduce((acc, curr) => { // key만 뽑기
      acc.push(
        <option value={curr} key={curr}>
          {dict[curr]}
        </option>
      );
      return acc; // 변경값 꼭 리턴 이 값을 참조해서 다음 iteration이 이루어짐
    }, []);
  }

export default makeOptionList;
```

```
// SearchTickets.jsx

import React, { useState } from 'react';
import MakeOptionList from './MakeOptionList';
import { destination, airline } from '../data/data';
import FilterTickets from './FilterTickets';

export default function SearchTickets() {
  // 상태값
  const [departures, setDepartures] = useState('ICN'); // 유저 선택 출발지
  const [arrivals, setArrivals] = useState('ICN'); // 도착지
  const [date, setDate] = useState(''); // 일자
  const [tickets, setTickets] = useState([]); // 검색된 티켓 저장

  const optionList = MakeOptionList(destination);

  function handleDeparturesChange(e) {
    setDepartures(e.target.value);
  }

  function handleArrivalsChange(e) {
    setArrivals(e.target.value);
  }

  function handleDateChange(e) {
    setDate(e.target.value);
  }

  const fetchData = () => { // 얘를 따로 빼서 api > fetch.js 파일에 위치시켜주고 싶다면 한번 해
봐주세용
    let tickets;
    fetch(
      `https://wdsgyqbtnjhqj0lotyaguhwhkoskeyf3.runner-forwarder-a-01.elice.io/ticket
s`
    )
      .then(res => res.json())
      .then(data => {
        tickets = data;

        tickets = tickets.filter(
          elem => elem.departures === destination[departures]
        );

        tickets = tickets.filter(
          elem => elem.arrivals === destination[arrivals]
        );

        for (let i = 0; i < tickets.length; i++) {
          let [dateString, timeString] = tickets[i].date
            .replace('Z', '')
```

```
            .split('T');
          const dateFormArr = dateString.split('-');
          const timeFormArr = timeString.split(':');
          tickets[
            i
          ].formedDate = `${dateFormArr[0]}년 ${dateFormArr[1]}월 ${dateFormArr[2]}일
 ${timeFormArr[0]}시 ${timeFormArr[1]}분`;
        } // 새로운 formDate키 하나 만들어주기
        setTickets(tickets);
      });
  };

  return (
    <div>
      <h1>항공권 예매하기 (2022년 4분기)</h1>
      <select data-testid="departures" onChange={handleDeparturesChange}>
        {optionList}
      </select>
      <select data-testid="arrivals" onChange={handleArrivalsChange}>
        {optionList}
      </select>
      <input data-testid="date" type="date" onChange={handleDateChange} />
      <button data-testid="search" onClick={fetchData}>
        검색
      </button>
      <FilterTickets tickets={tickets} />
    </div>
  );
}
```

```
// FilterTickets.jsx

import React, { useState } from 'react';
import MakeOptionList from './MakeOptionList';
import { airline } from '../data/data';

export default function FilterTickets({ tickets }) {
  const [airlineFilter, setAirlineFilter] = useState('no-airline');
  const [dateSort, setDateSort] = useState('date-no');
  const [priceSort, setPriceSort] = useState('price-no');

  function handleFilterAirlineChange(e) {
    setAirlineFilter(e.target.value);
  }
  function handleSortDateChange(e) {
    setDateSort(e.target.value);
  }
  function handleSortPriceChange(e) {
    setPriceSort(e.target.value);
  }
```

```
  const airlineList = MakeOptionList(airline);

  // 필더링 된 티켓들을 화면에 렌더링해주는 함수를 만들자!는 목표
  const printTickets = () => {
    // 가격정렬
    if (priceSort === 'price-no') {
      // price-no
    } else if (priceSort === 'price-cheap') {
      // price-cheap
      tickets.sort((a, b) => a.price - b.price); // ascending.
    } else {
      // price-expensive
      tickets.sort((a, b) => b.price - a.price); // descending
    }

    // 날짜 정렬
    if (dateSort === 'date-no') {
      // date-no
    } else if (dateSort === 'date-fast') {
      // date-fast
      tickets.sort((a, b) => {
        if (a.date > b.date) return 1;
        else if (b.date > a.date) return -1;
        else return 0;
      });
    } else {
      // date-sloe
      tickets.sort((a, b) => {
        if (a.date > b.date) return -1;
        else if (b.date > a.date) return 1;
        else return 0;
      });
    }

    // 에어라인 필터 정렬
    if (airlineFilter !== 'no-airline') {
      tickets = tickets.filter(
        ticket => ticket.airline == airline[airlineFilter]
      );
    }

    return tickets.length !== 0
      ? tickets.map(ticket => (
          <li key={ticket.id}>
            {ticket.airline} / {ticket.formedDate} / {ticket.price}원
          </li>
        ))
      : '검색 결과가 존재하지 않습니다.';
  };

  return (
    <div>
      <h2>검색 결과</h2>
      <div>항공사 필터 / 시간순 정렬 / 가격순 정렬</div>
```

```
      <select data-testid="filter-airline" onChange={handleFilterAirlineChange}>
        <option key="no-airline" value="no-airline">
          항공사 필터 없음
        </option>
        {airlineList}
      </select>
      <select data-testid="sort-date" onChange={handleSortDateChange}>
        <option key="date-no" value="date-no">
          시간 정렬 없음
        </option>
        <option key="date-fast" value="date-fast">
          출발 일시 빠른순
        </option>
        <option key="date-slow" value="date-slow">
          출발 일시 늦은순
        </option>
      </select>
      <select data-testid="sort-price" onChange={handleSortPriceChange}>
        <option key="price-no" value="price-no">
          가격 정렬 없음
        </option>
        <option key="price-cheap" value="price-cheap">
          가격 낮은순
        </option>
        <option key="price-expensive" value="price-expensive">
          가격 높은순
        </option>
      </select>
      <br />
      <br />
      {printTickets(tickets)}
    </div>
  );
}
```

```
// data > data.js


// 도시 데이터
export const destination = {
  ICN: '인천',
  CJU: '제주',
  HKG: '홍콩',
  HAN: '하노이',
  KIX: '오사카',
  NRT: '도쿄',
  SHA: '상하이',
  CDG: '파리',
  LHR: '런던',
  SPN: '사이판',
};
```

```
// 항공사 데이터
export const airline = {
  KOREANAIR: '대한항공',
  ASIANA: '아시아나항공',
  TEEWAY: '티웨이항공',
  JINAIR: '진에어',
  AIRSEOUL: '에어서울',
  JEJUAIR: '제주항공',
  EASTAR: '이스타항공',
};
```

급하게 최적화 복습!!!

1. useSelector로 값 꺼내쓸 때 최적화

독립 선언!! 두둥

이전에 다뤘던 예제

```
// cakeSlice.js

const initialState = {
  numOfCakes: 10, // 초기 남아있는 재고
  price: 10000
};

reprice: (state, action) => {
      state.price += 1000;
    }

export {reprice}
```

```
// Cake2. js

import { reprice } from "./cakeSlice";
import { useSelector, useDispatch } from "react-redux";

const Cake2 = () => {
  const dispatch = useDispatch();
  const { price } = useSelector((state) => state.cakeReducer);
  // const price = useSelector((state) => state.cakeReducer.price);

// 기본적으로 스토어에 변형이 생기면 값들이 리렌더링되는데,
// 그런데 이렇게 불러오면 어떤 상태값에 변화가 생겼는지 몰라서 모든 값을 새로 할당합니다
```

```
// 그래서 이렇게 독립선언을 해줘야합니다!

  return (
    <>
      <br></br>
      <h1>케이크 가격: {price}</h1>
      <button
        onClick={() => {
          dispatch(reprice());
        }}
      >
        1000원인상!
      </button>
    </>
  );
};

export default Cake2;
```

```
// Cake.js

import Cake2 from "./Cake2";

const Cake = () =>{
  // const numOfCakes = useSelector((state) => state.cakeReducer.numOfCakes);
  // const price = useSelector((state) => state.cakeReducer.price);
  return ...
  <Cake2 />
}
```

2. useMemo

이전에 공부했던 React.Memo, useCallback 실습 예제에서 useMemo도 집고 넘어가고 싶었는데 시간이 부족했네용 (배달예제)

지난번처럼 함수가 생성될 때는 그 값에 메모리 주소가 담겼던 것과 같은 원리로 오브젝트가 생성되어도 똑같이 주소가 할당되며 다시 오브젝트가 생성될 때마다 그 주소가 바뀝니다!

그래서 동일한 원리로 props으로 해당 오브젝트를 넘겨주는 경우 useMemo로 주소값이 바뀌지 않게 해줄 수 있고, 아니면 함수 결과값이 특정값에 따라 계속 바뀔 수 있는데 계산하는데 시간이 오래 걸리는 경우 useMemo를 사용할 수 있어요.

할까말까했던… framer-motion…

쓸데없이 좀 있어보니는 애니메이션이 아니라 사소하지만 사용자 경험을 풍부하게 하는 애니메이션을 구현할 수 있습니다!!

우리에게 `motion` 컴포넌트를 제공하는데요, props를 넘겨주면서 animation을 만들 수 있습니다.

아래 코드처럼 애니메이션 컴포넌트를 만들고 아래처럼 children props로 넘겨주면 됩니다!! 쉽죠?!

<애니메이션컴>

<애니메이션적용해주고싶은컴포넌트>

</애니메이션컴>

하나씩 돌려보시면 쉽고 풍부하게 적용이 가능하다는 걸 느끼실 거에요~!

```
import { motion } from "framer-motion";


export const LeftSlide = ({ children }) => {
  return (
    <motion.div
      initial={{ opacity: 0, x: 100 }} // 시작 위치. 처음 마운트 될 때 상태,
      animate={{ opacity: 1, x: 0 }} // // 애니메이션이 끝났을 때의 상태. animate 속성에 적용된 값이 initial이랑 다르면 자동으로 애니메이션을 적용해 줍니다.
      transition={{ duration: 1.5 }} // 요소의 변화를 일정기간동안 일어나게
    >
      {children}
    </motion.div>
  );
};

export const UpSlide = ({ children }) => {
  return (
    <motion.div
      initial={{ opacity: 0, y: 100 }}
      animate={{ opacity: 1, y: 0 }}
      transition={{
      // delay: 0.5,
        y: {
          ease: "easeInOut",
          duration: 1.5,
        },
      }}
    >
      {children}
    </motion.div>
```

```
  );
};


export const Rotate360 = ({ children }) => {
  const transitionValues = {
        repeat: Infinity,
        repeatDelay: 1,
        ease: "easeInOut",
        type: "spring", // 용수철처럼 띠용띠용 애니메이션
        stiffness: 100,
        scale: {
          duration: 1
  }
  return (
    <motion.div
      initial={{ scale: 0 }}
      animate={{ rotate: 360, scale: 1 }}
      transition={transitionValues}
    >
      안뇽
    </motion.div>
  );
};


export const CoolView = ({ children }) => {
// 빙글빙글 도는 애니메이션
  return (
    <motion.div
      initial={{ opacity: 0.2 }}
      whileInView={{ // 안보였다가 보일 때
        opacity: 1,
        rotate: [0, 360],
        fontWeight: [100, 600],
        transition: { delay: 0.5 }
      }}
      whileHover={{ // 호버했을 때
        scale: 1.2,
        transition: { type: "spring", stiffness: 400, damping: 10 }
      }}
    >
      Hola
    </motion.div>
  );
};
```