2025

React Term Project

2501110204

도경진

Project git hub 주소(Public): https://github.com/dgjPolyTech/mogrizi_2.0

개요 및 주제 선정 이유

1학기 당시에 만들었던 그림 기능 사이트 "모그리지"를 리액트로 재구성하게 되었습니다.

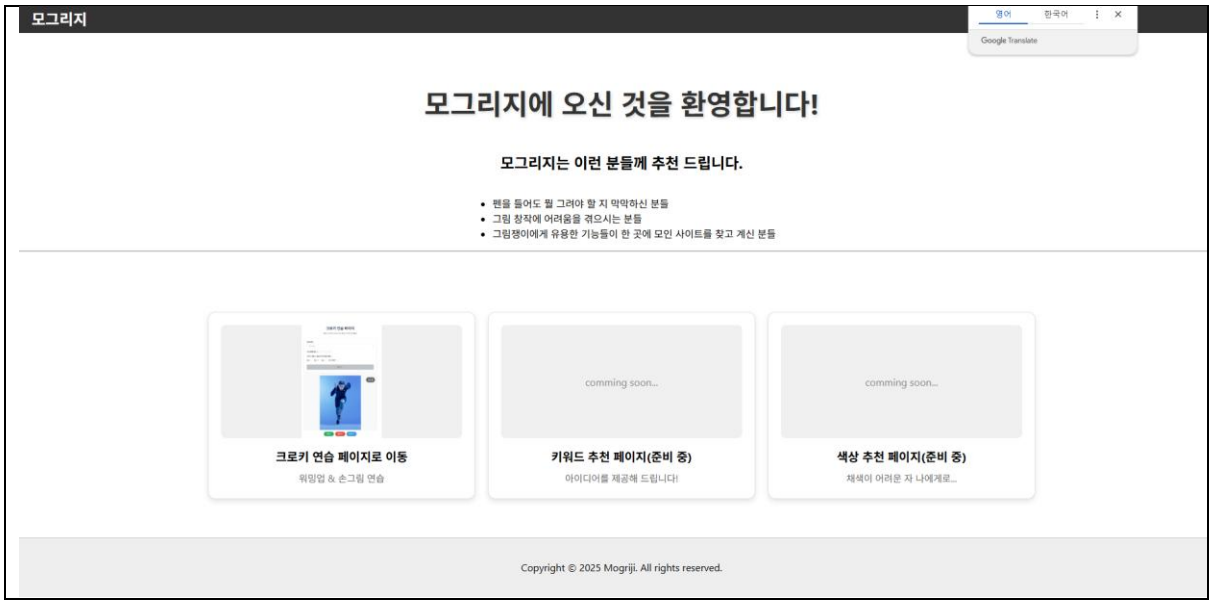
당시 메인 페이지, 크로키 페이지, 키워드 추천 페이지, 색상 추천 페이지를 만들었었는데, 원래는 리액트에서도 똑같이 구현하려 하였으나 당시에 각 페이지를 완성만 하는데 급급하여 추가로 넣을 수 있는 기능도 넣지 못하였고, css도 깔끔하게 넣지 못해서 이번에는 메인 페이지와 추가로 핵심 기능인 크로키 페이지 두 곳을 집중적으로 구현하기로 하였습니다.

페이지별 주요 기술 및 설명

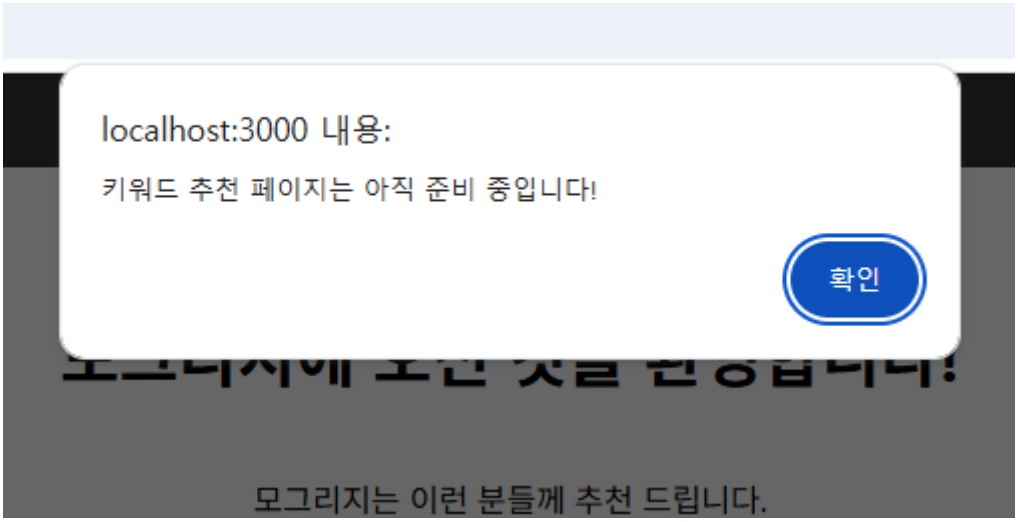
메인 홈(Home.jsx)

전

후



- 사이트의 중심인 메인 페이지입니다. 구조는 이전과 거의 동일하게 하되, 조금 더 심플하고 차분한 UI로 구성하였습니다.



- 키워드 추천 및 색상 추천 페이지는 이번 과제에서는 구현하지 않아서, 누르면 이동하는 대신 alert 알림 창이 나오게 하였습니다. 두 페이지는 과제 제출 후 추후 방학 등 자투리 시간을 활용해서 만들어볼 생각입니다.

사용된 주요 기술

```
unction Home() {
    // useNavigate: 특정 상황에서 url을 변경하여 페이지 이동 시켜준다.
    const navigate : NavigateFunction = useNavigate();

    // 매개변수로 받은 path url로 페이지를 옮겨준다.
    const move = (path) : void => {
        navigate(path);
    };

    // 매개변수로 받은 페이지명을 토대로 "~는 아직 준비 중입니다."라는 문구 출력.
    const stop = (value) : void => {
        alert(value+"는 아직 준비 중입니다!");
    }
```

- 페이지 이동에는 **useNavigate()** state를 사용하였습니다.
- 해당 state는 특정 상황에서 현재 페이지의 url을 바꾸는 기능을 하는데, onClick 이벤트 리스너로 move 함수가 구동되면서 각 카드마다(현재는 크로키 카드만) 해당하는 페이지로 이동하게끔 구성하였습니다.
- 상술한 키워드 추천 및 색상 추천 페이지는 이번엔 구현하지 않았으므로 navigate 대신 alert 알림창을 띄웁니다.

주요 css 스타일

```
html, body {
    min-height: 100vh; /* 보여지는 화면 기준 최소 높이를 100%로 주기 */
    margin: 0;
    padding: 0;

    display: flex; /* 자식 요소를 원하는 방향으로 정렬할 수 있게 해주는 핵심 속성 */
    flex-direction: column; /* column(열)은 flex로 정렬되는 방향을 세로 기준으로 함.(열처럼) */
    /*
        a
        b
        c
    */
}
```

- 메인 페이지 자체를 포함해 대부분의 요소의 css를 **flex** 속성으로 주어, 영역 마다 원하는 방향대로 flex-direction을 구성해 요소를 배치할 수 있게끔(메인 소개글은 세로 기준인 column, 카드 배치는 가로 기준인 row 식) 구성하였습니다.

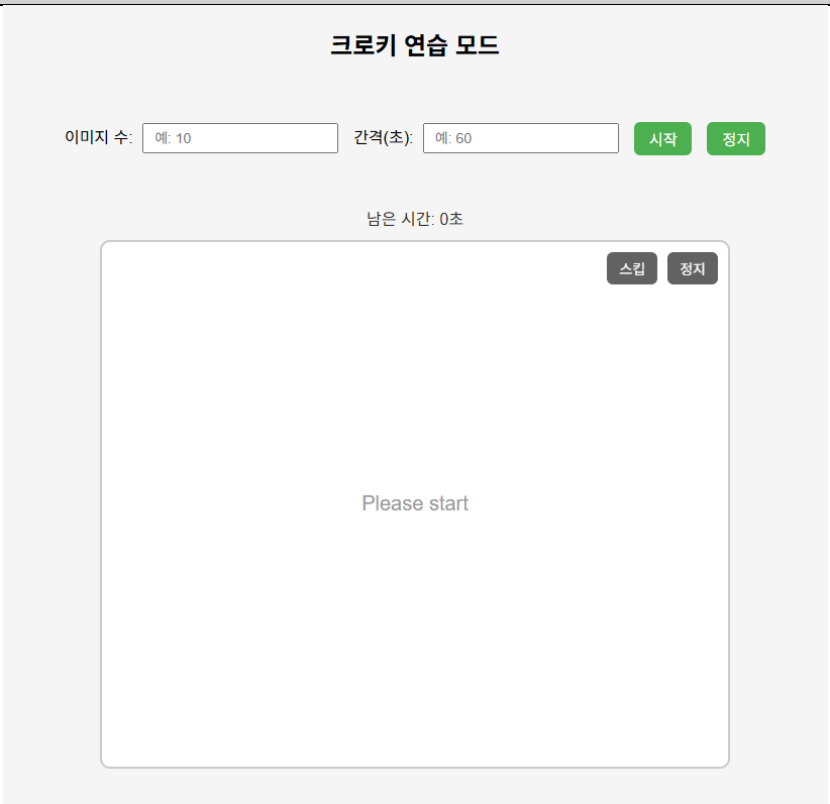
- 이전에 html로 진행한 것과 마찬가지로, 카드 위에 마우스를 올렸을 때(**:hover**) **transform: translateY** 효과와 **box-shadow** 효과를 적용해 사용자가 시각적으로 선택한 카드를 확실히 알아볼 수 있게 하였습니다.

```css
.home-intro-title {
    font-size: 2.8rem;
    font-weight: 700;
    text-align: center;
    color: #333; /* 기본 어두운 회색 */
    margin-bottom: 25px;

    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.15); /* 말 그대로 텍스트 뒤에 그림자 효과. */
}

/* 위의 text-shadow 색상을 조절할 수 있음. */
.home-intro-title .highlight {
    color: #FF6B6B; /* 예쁜 코랄색 포인트 */
}
```

- 또한 메인 페이지의 제목 역할인 "**모그리지에 오신 것을 환영합니다!**"라는 문구에는 폰트 크기 및 색상을 조절하고, **text-shadow** 및 **highligt** 속성을 적용하여 일반적인 텍스트화 차별화 되게 꾸몄습니다.

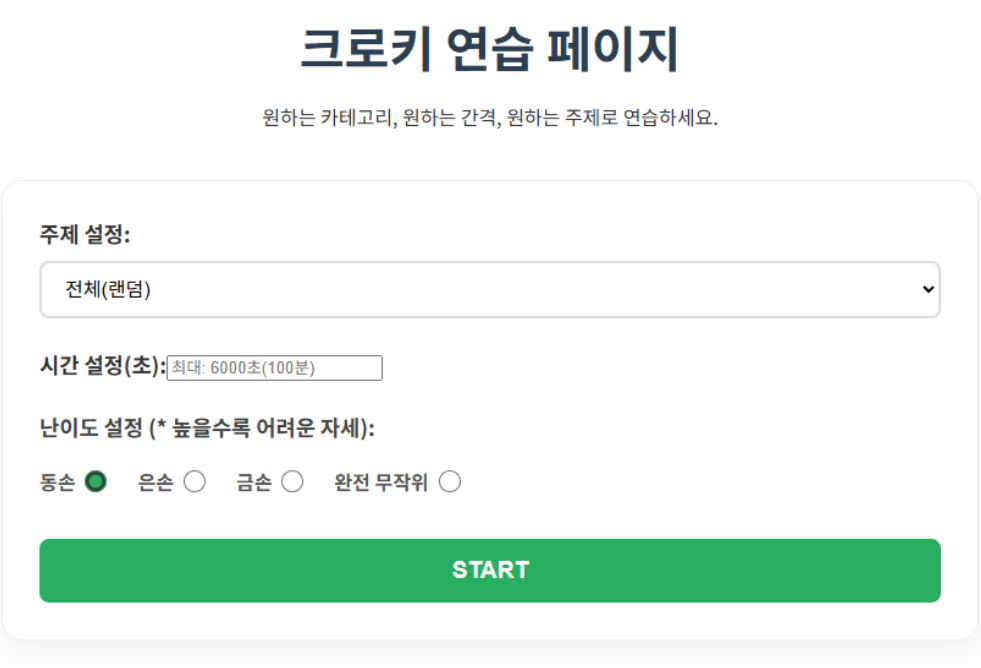
크로키 페이지(Croquis.jsx & Croquis_view.jsx)

전



후

크로키 설정 및 메뉴 화면



실제 크로키 진행 시 화면

- 이번 과제의 핵심인 크로키 연습 페이지 입니다. 설정값을 받아 넘겨주는 Croquis.jsx 페이지와, 그 설정값을 토대로 이미지를 보여주는 Croquis_view.jsx 자식 컴포넌트로 구성되어 있습니다.
- 단순히 이미지 수와 시간(간격)만 배치할 수 있던 이전과 달리, 주제(스포츠, 캐쥬얼, 액션)와 난이도를 설정하여(난이도가 높을 수록 더 어려운 자세가 등장) 연습하기 좋도록 하였습니다.
- Ui 또한 좀 더 아기자기 하게 배치했습니다.

사용된 주요 기술

데이터 전송

```
function Croquis(){
    // croquis_view에게 보내야 할 사용자가 선택한 주제, 입력한 시간, 시작 여부, 난이도 값을 useState로 정의.
    const [category : string , setCategory] = useState( initialState: "all");
    const [time : string , setTime] = useState( initialState: "");
    const [isStart : boolean , setIsStart] = useState( initialState: false);
    const [difficult : number , setDifficult] = useState( initialState: 1);
```

- 먼저 croquis 페이지에서는 메뉴 탭에서 받아들이는 사용자 입력값을 각각 category(주제), time(이미지별 시간(간격)), isStart(시작 여부), difficult(난이도)라는 이름의 **useState** state로 만듭니다.

각 state 별 관련 함수

```
// 시간 변경 시마다, time state 값을 변경함.
const handleTimeChange = (event) : void  => {
    let value = event.target.value;

    if (Number(value) > 6000) {
        alert("최대 6000초(100분)까지 가능합니다.");
        return;
    }
    setTime(value);
};

// selectBox에서 옵션 변경 시마다 category 변경
const handleOptionChange = (event) : void  => {
    setCategory(event.target.value);
};

// 라디오 버튼으로 다른 거 선택할 때마다 difficult 변경
const handleDifficultChange = (event) : void  => {
    setDifficult(Number(event.target.value));
};
```

- handle~ 로 이름붙여진 함수들을 메뉴의 각 입력 요소에 심어두어서, 시간 값이 새로 입력되거나 난이도 설정의 radio 버튼이 변경되었다거나, 주제 선택 select box를 새로 선택했다던가 하는 상황에서 설정값을 다시 설정하게끔 합니다.

form 기반 Props 전달 구조

```
<form onSubmit={handleStart}>
    <div className="croquis-menus">
        <ul>
            <li>
```

```
{/* croquis_view 영역은 isStart가 true일 때만(=크로키 진행 중일 때만) 보이게 설정함. */}
{isStart &&
    <div className="croquis-view-container">
        <Croquis_view
            category={category}
            time={Number(time) * 1000} // 밀리초 변환
            difficult={difficult}
            onStop={handleStop}
        />
    </div>
}
</div>
```

- 이러한 state들을 form으로 묶어, submit 실행 시 자식 컴포넌트로 데이터를 보내는 식으로 구동합니다. (**Props 전달 구조**)

크로키 진행 시 메뉴 항목 비활성화

```
// start 버튼 눌렀을 시 발동.
const handleStart = (event) : void  => {
    event.preventDefault(); // 새로고침 방지.

    // 시간 설정은 사용자가 선택하는게 아니라 입력하는 값이기에 유효성 검사를 진행함.
    if(time === "" || Number(time) <= 0){
        alert("시간을 1 이상 입력해주십시오.");
        return;
    }

    if (!/^\d*$/.test(time)) {
        alert("시간을 올바르게 입력해주십시오.");
        return;
    }

    // isStart true로 만듦과 동시에, 요소별 disabled 값에 영향을 줘서 크로키 진행 중에는 입력 추가로 못하게 방지함.
    setIsStart( value: true);
};

const handleStop = () : void  => {
    setIsStart( value: false);
};
```

```
{/* 시작 버튼 (시작 중엔 회색으로 변함) */}
<button type="submit" disabled={isStart}>
    {isStart ? "진행 중..." : "START"}
</button>
```

```
{/* 게임 화면 (isStart가 true일 때만 보임) */}
{isStart &&
    <div className="croquis-view-container">
        <Croquis_view
            category={category}
            time={Number(time) * 1000} // 밀리초 변환
            difficult={difficult}
            onStop={handleStop}
        />
    </div>
}
/div>
```

- 그리고, 사용자가 시작 버튼을 누를 시 isStart=True로 바뀌며, 메뉴 창의 요소들의 disabled 여부 역시 true로 바뀌어 결과적으로 크로키 진행 하는 동안 메뉴의 입력값을 실수로 입력해 바뀌는 등의 불상사를 방지합니다.

```
// require.context는 컴포넌트 밖에서 선언
const imgContext = require.context("../assets", true, /\.png$/);
```

```
// 이미지 변경 함수
const change_img = useCallback( callback: () : void => {
    const allKeys = imgContext.keys();
    const validKeys = allKeys.filter((path) => {

        const isCategoryMatch = props.category === 'all' || path.includes(props.category);

        const diffPattern : RegExp = new RegExp( pattern: `[\\/\\\\]${props.difficult}_|^${props.difficult}_`)
        const isDiffMatch : boolean = props.difficult === 0 || diffPattern.test(path);

        return isCategoryMatch && isDiffMatch;
    });

    if (validKeys.length === 0) {
        console.warn( message: "이미지를 찾을 수 없습니다.");
        return;
    }

    const randomIdx : number = Math.floor( x: Math.random() * validKeys.length);
    const selectedKey = validKeys[randomIdx];

    // 이미지 데이터 가져오기 (.default 처리)
    const imageModule = imgContext(selectedKey);
    const imagePath = imageModule.default || imageModule;

    setCurrImg(imagePath);
    setTimeLeft( value: props.time / 1000); // 시간 리셋
}, deps: [props.category, props.difficult, props.time]);
```

- Croquis_view 페이지에서는 받아 온 설정값들을 기반으로 보여질 이미지의 경로를 사용자의 설정값을 기준으로 랜덤하게 설정합니다.
- 정확한 원리는, 먼저 **requireContext** 함수를 통해, assests 폴더 안의 .png로 끝나는 이미지들을 imgContext 변수에 집어넣습니다. 이후 **filter**() 함수를 통해, 사용자가 정의한 조건에 해당하는 이미지(각 이미지들이 모인 폴더명은 카테고리명의 값과 같고, 이미지 별로 앞에 적힌 1_@, 2_@와 같은 숫자는 그 이미지의 난이도를 나타냅니다.)만 모인 리스트를 반환하게 되며, 해당 리스트에 모인 이미지를 Math.random으로 무작위 값을 도출한 **randomIdx**값을 통해 무작위 이미지를 반환하게 되는 것입니다.

주요 css 스타일

```css
.croquis-view-box img {
    max-width: 100%;
    max-height: 100%;
    object-fit: contain;
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
}
```

- 크로키 이미지를 박스 안에 알맞게 배치하는 것에는 **object-fit: contain**이라는 스타일이 적용 되었습니다. 해당 속성은 이미지를 칸 안에 이미지 자체의 비율을 유지하면서 알맞게 들어가는 역할을 합니다.

```css
/* 타이머 디자인 */
.timer-display {
    position: absolute;
    top: 20px;
    right: 20px;
    background-color: rgba(0, 0, 0, 0.6);
    color: white;
    padding: 10px 20px;
    border-radius: 30px;
    font-size: 1.5rem;
    font-weight: bold;
    z-index: 10;
    box-shadow: 0 2px 10px rgba(0,0,0,0.2);
}
```

- 이미지 위 타이머는 **position: absolute** 속성을 주어, 이미지 위에 표시되게끔 하였습니다. 정확히는 타이머의 부모에 해당하는 croquis-view-box 에는 **relative** 속성을 주어 자식인 타이머가 부모인 croquis-view-box 자신을 기준으로 위치하게 하고, 타이머(timer-display)는 **absolute** 속성을 주어 다른 개체보다 무조건 상위에 표시되도록 합니다. absolute 속성은 부모 컴포넌트에 position 관련 속성이

설정되어야 제대로 사용할 수 있다고 합니다.

```css
/* 애니메이션 */
/* keyframe은 쉽게 요약하면 요소별로 쓰일 애니메이션을 직접 커스텀해서 쓰고 싶을 때 사용*/
@keyframes fadeIn {
    /* 애니메이션 효과 처음 시작할 때 */
    from { opacity: 0; transform: translateY(20px); }

    /* 애니메이션 효과 끝날 때 효과*/
    to { opacity: 1; transform: translateY(0); }
}
```

```css
.croquis-view-container {
    margin-top: 30px;
    text-align: center;
    animation: fadeIn 0.5s ease-in-out;
}
```

- 그 외에 버튼을 눌렀을 때 꾹 눌리는 듯한 애니메이션은 **transform** 속성을 사용하였고, 버튼이나 크로키 박스 이미지가 부드럽게 나타나는 등의 연출은 **transition** 속성을 활용하였습니다.
- **keyframe**은 이런 애니메이션 효과 설정값을 커스텀 하여 만들어두고 쓰고 싶을 때 사용합니다. 이번 과제에서는 fadeIn이라는 이름으로 만들어 버튼 및 크로키 이미지 박스(croquis-view-container) 등장 연출에 사용하였습니다.