



개발일지

day1.

참고 : <https://collabio-kriziu.herokuapp.com/>

링크를 참조해서 대략적인 설계 후 시작 화면까지는 완료.

클라이언트와 서버 둘 다 배웠음에도, 합치기가 결코 쉽지 않다.

뭐부터 해야할지 잘 모르겠고 프로젝트를 시작할 때 마다 배웠던 개념으로는 전혀 충분하지않은걸 알게된다.

프론트 먼저 만들고 서버를 만들어봐야겠다.

R.T.D

Share Your Drawing!

그림판에서 사용할 이름을 설정하세요.

닉네임 설정 *

그림판 id를 입력하세요.

그림판 id설정 *

해당 id를 가진 새로운 방을 만들거나 참가하세요.

방 참가하기

or

방 생성하기

day2.

나도 모르게 흐릿하게 그냥 지나쳤던 form태그를 이용한 요청 방식과, axios를 이용한 요청 방식을 구별하는 것을 알게되었다.

참고 : <https://www.infllearn.com/questions/58743/axios-로-post전송을-구현하는-거랑-form-method로-post-전송하는-거랑-차이가-있나요>

프로그램을 시작하는데 있어서 가장 중요한 부분인데(방 생성이나, 참가냐에 따라 서버에 들어오는 요청과 db에서 읽어오는 자료가 다르기 때문)

크게 놓치고 있었다.

(Start.jsx)

참가하기와 , 생성하기 버튼을 나눠서 요청 url을 다르게했다.

```

...

const joinRoomHandler = (e) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append("drawid", drawid);
  formData.append("nickname", nickname);
  axios({
    method: "post",
    url: "http://localhost:8080/join",
    data: formData,
  })
  .then((result) => {
    console.log("요청성공");
    console.log(result);
  })
  .catch((error) => {
    console.log("요청실패");
    console.log(error);
  });
};

const createRoomHandler = (e) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append("drawid", drawid);
  formData.append("nickname", nickname);
  axios({
    method: "post",
    url: "http://localhost:8080/create",
    data: formData,
  })
  .then((result) => {
    console.log("요청성공");
    console.log(result);
  })
  .catch((error) => {
    console.log("요청실패");
    console.log(error);
  });
};

return (
  <main>
    ...
    <FormText>해당 id를 가진 새로운 방을 만들거나 참가하세요.</FormText>
    <button onClick={joinRoomHandler}>방 참가하기</button>
    or
    <button onClick={createRoomHandler}>방 생성하기</button>
  </div>
</form>
</div>
</main>
);
};

export default Start;

```

처음에는

```

<form action="/join"
method="POST">

```

이렇게 해버려서 요청이 다른데 어떻게 나누지?라고 생각했었다.(useState로 나눌뻔했다;;)

다음에는 필요할 때 아니면 axios사용할 것!

(server.js)

서버에서도 기능 설계에 따라 요청을 나누어 받았다.
아직 join으로 들어왔을 때 입력한 방id가 없을 때 어떤 응답을 줘야하는지 몰라서 비워 놓았다.

```

const express = require("express");
const app = express();

app.use(express.json());
var cors = require("cors");
app.use(cors());

let db; // db 저장할 변수하나 필요
MongoClient.connect(
  "mongodb+srv://admin:mm1166911@mucham.f69xhd2.mongodb.net/?retryWrites=true&w=majority",
  (error, client) => {
    if (error) return console.log(error); // 에러있으면 출력
  }
);

```

```

db = client.db("drawapp"); // drawapp db로 연결

app.post("/create", (req, res) => {
  // /create로 접속시
  console.log(req.body); // 받은 form 데이터
  // 제출된 id를 가진 방이 존재하는지 확인
  db.collection("drawroom").findOne(
    { roomid: req.body.drawid },
    (err, result) => {
      if (result) {
        res.send(result); // 해당 아이디로 생성된 방이 있다면 프론트에서 아이디 다시 입력
      } else {
        db.collection("drawroom").insertOne(
          // drawroom collection에 insertOne{ 자료 }
          { roomid: req.body.drawid, member: [req.body.nickname] }, // 해당 아이디로 된 방이 없으면 새로만들기
          (error, result) => {
            if (error) return console.log(error);
            console.log("line 31 저장완료", result);
            res.send(result); // 만든방 정보 넘겨주기
          }
        );
      }
    }
  );
});

// .listen 으로 서버를 열 수 있다.
app.listen(8080, () => {
  console.log("listening on 8080");
});
}
);

app.post("/join", (req, res) => {
  // join으로 접속시
  console.log(req.body); // 받은 form 데이터
  db.collection("drawroom").findOne(
    // 입력한 방 id와 같은 이름 찾아서
    { roomid: req.body.drawid },
    (err, result) => {
      if (result) {
        let Participants = [...result.member, req.body.nickname]; // 해당 방의 멤버로 추가해주기
        db.collection("drawroom").insertOne(
          // drawroom collection에 insertOne{ 자료 }
          { roomid: req.body.drawid, member: Participants },
          (error, result) => {
            if (error) return console.log(error);
            console.log("line 31 저장완료", result);
            res.send(result); // 만든방 정보 넘겨주기
          }
        );
      } else {
        // 입력한 방 id가 없다면?
        res.send();
      }
    }
  );
});

app.get("/draw", (req, res) => {
  db.collection("drawroom")
    .find()
    .toArray((err, result) => {
      console.log("line29", result);
      res.send(result); // db에서 받아온 데이터 보내기
    });
});

app.get("/*", function (req, res) {
  res.sendFile(path.join(__dirname, "/react-project/build/index.html"));
});

```

day3.

Express : res.send() res.json() res.end() 비교

<https://velog.io/@yunsungyang-omc/nodejs-Express-res.send-res.json-res.end-비교>

React 와 Express 연동하기

<https://wonyoung2257.tistory.com/5?category=805961>

<https://wonyoung2257.tistory.com/6?category=805961>

formData, formData값 확인

<https://inpa.tistory.com/entry/JS-FormData-정리-fetch-api>

- 방 만들기

받은 데이터를 가진 방으로 이동시켜야함.

Start.jsx에서 데이터가 정상적으로 받아와진다면 데이터를 dispatch해서 store.js에 저장, state 변경

App.js에서 그 state를 받아서 조건문으로 Start.jsx or Draw.jsx 렌더링



받아온 정보가 없다면

중복되는 경우(create), 입력한 id가 존재하지 않는 경우(join) ⇒ Start.jsx 다시 렌더링, 아이디 다시 입력

받아온 정보가 있다면

Draw.jsx를 렌더링 할 때는 받아온 정보를 가지고 렌더링 해야함(방마다 정보가 다르기 때문)

!! 시작 화면에서 닉네임과, id를 설정하지 않고 그냥 들어가는 것을 방지 하기 위해 validation을 추가했다.

(Start.jsx)

```
let [enteredDrawid, setEnteredDrawid] = useState("");
let [enteredNickname, setEnteredNickname] = useState("");
const [formIsValid, setFormIsValid] = useState(false);

useEffect(() => {
  const identifier = setTimeout(() => {
    // console.log("checking validity");
    setFormIsValid(
      enteredDrawid.trim().length > 3 && enteredNickname.trim().length > 0
    );
  }, 300);

  return () => {
    // console.log("cleanUp");
    clearTimeout(identifier);
  };
}, [enteredDrawid, enteredNickname]);

const nicknameHandler = (e) => {
  setEnteredNickname(e.target.value);
};
const drawidHandler = (e) => {
  setEnteredDrawid(e.target.value);
};

...

<form onSubmit={handleSubmit}>
  <div className={classes.split}>
    <FormText>그림판에서 사용할 이름을 설정하세요.</FormText>
    <TextField
      type="text"
    />
  </div>
</form>
```

```

        variant="outlined"
        label="닉네임 설정(한 글자 이상)"
        name="nickname"
        onChange={nicknameHandler}
      />
    </div>
    <div className={classes.split}>
      <FormText>그림판 ID를 입력하세요.</FormText>
      <TextField
        type="text"
        variant="outlined"
        label="그림판 ID(네 글자 이상)"
        name="drawid"
        onChange={drawidHandler}
      />
    </div>
    <div className="form3">
      <FormText>해당 id를 가진 새로운 방을 만들거나 참가하세요.</FormText>
      <button
        type="submit"
        onClick={joinRoomHandler}
        disabled={!formIsValid}
      >
        방 참가하기
      </button>
      or
      <button
        type="submit"
        onClick={createRoomHandler}
        disabled={!formIsValid}
      >
        방 생성하기
      </button>
    </div>
  </form>

```

(begin.module.css)

...

```

button:disabled,
button:focus:disabled,
button:hover:disabled,
button:active:disabled {
  background: #ccc;
  border-color: #ccc;
  color: #666666;
  cursor: not-allowed;
}

```

▼ 코드 짜다가 마주한 Immer 오류

! Error: [Immer] An immer producer returned a new value *and* modified its draft. Either return a new value *or* modify the draft.

검색 결과.

<https://velog.io/@ohy9205/redux-toolkit-Immer-An-immer-producer-returned-a-new-value-and-modified-its-draft.-Either-return-a-new-value-or-modify-the-draft>

redux-toolkit creatSlice로 만든 action을 컴포넌트에서 호출할 때 발생

state 변경 시 새 값을 반환하면서 원본도 수정했기 때문에 발생하는 에러

redux-toolkit의 immer사용 규칙을 보면 원본을 변경하던가 새 값을 리턴하던가 둘 중 하나만 해야함

chatgpt 결과.

오류의 원인은 다음 코드 블록에서 발생할 수 있습니다:

```
dispatch(joinRoom(result));
```

해당 코드는 `dispatch` 함수를 호출하여 `joinRoom` 액션을 디스패치합니다.

그러나 `joinRoom` 액션 생성자에서 값을 수정하고 새로운 값을 반환하지 않은 것으로 보입니다.

Immer는 기본적으로 불변성을 유지하도록 설계되었기 때문에, 값을 수정하는 동시에 새로운 값을 반환하는 것은 규칙을 위반하는 것입니다.

해당 부분을 수정하여 값을 반환하도록 변경할 수 있습니다:

```
dispatch(joinRoom(result.data));
```

위 코드에서 `result.data`를 반환하도록 변경하면 오류가 해결됩니다.

이렇게 하면 `joinRoom` 액션의 페이로드로 `result.data` 값을 전달하게 됩니다.

```
(Start.jsx)

import { Routes, Route, useNavigate } from "react-router-dom";
import { useDispatch } from "react-redux";
import { joinRoomInfo } from "../store/store";
...
// 방 참가하기
const joinRoomHandler = async () => {
  axios
    .post("http://localhost:8080/join", {
      body: {
        drawid: enteredDrawid,
        nickname: enteredNickname,
      },
    })
    .then((result) => {
      console.log("join 요청성공");
      // 여기가 ""이면 입력한 id로 된 방 없음,
      // 있으면 해당 정보 store에 dispatch
      console.log(result.data);
      if (result.data === "") {
        alert("해당 id로 된 방이 없습니다. 방을 새로 생성하세요.");
      } else {
```

데이터가 잘 받아와지는 것 확인함.

서버로 DB에서 받아온 데이터 요청 후

받은 데이터를

useDispatch를 이용해 store에 저장하고,

useNavigate를 이용해 draw로 이동시켰다.

되는지 안해봄.

```

        dispatch(joinRoomInfo(result.data));
        navigate("/draw", { replace: true });
    }
})
.catch((error) => {
    console.log("join 요청실패");
    console.log(error);
});
});

// 방 생성하기
const createRoomHandler = async () => {
    axios
        .post("http://localhost:8080/create", {
            body: {
                drawid: enteredDrawid,
                nickname: enteredNickname,
            },
        })
        .then((result) => {
            console.log("create 요청성공");
            // 여기서 이미 해당 id로 존재하는지 구별 해야함.
            // 없는 방 만들때 == result.data.insertedId
            // 이미 존재하는 방일때 == result.data.roomid
            console.log("create", result.data);
            if (result.data === "") {
                alert("해당 ID로 된 방이 이미 존재합니다. 다른 ID로 만들어 주세요.");
            } else {
                console.log(result.data);
                dispatch(joinRoomInfo(result.data));
                navigate("/draw", { replace: true });
            }
        })
        .catch((error) => {
            console.log("create 요청실패");
            console.log(error);
        });
};

...
return (
    <main>
    ...
        <Routes>
        { /* <Route path="/" element={<Start /> } /> */ }
        <Route path="/draw" element={<Draw /> } />
        </Routes>

    </main>
)

```

day4.

Palette

<https://casesandberg.github.io/react-color/#api>

<https://velog.io/@hami/React-Color>

stroke

<https://clownhacker.tistory.com/115>

기능 개발은 레이아웃과 전체적인 모양을 대충 잡고,

그림 그리는 방법과 그걸 socket.io로 공유하는 방법을 좀 더 공부하고 나서 해야겠다.

• 그림도구 컴포넌트

- 펜, 지우개, 펜 두께, 색 고르기

아이콘 모양부터 가져와서 누를것 만들기.

아이콘 마우스 hover시 tooltip 띄우기.

- 사진 업로드(사진 위에 그림 가능)
- 그림 저장, 방 나가기, **초대하기**

아이콘들 들어가 있는 toolbox만들기.

화면 한쪽에 toolbox, chatbox 위치 ⇒ absolute 아님.

방 접속중 유저는 chatbox 상단에 표시.

생각해보니 각각의 기능이 모두 컴포넌트이자 함수여서 다 다른 파일로 분할했다.

페이지 라우팅 개념이 아직 헷갈린다.

첫 페이지에서 방 참가, 생성 후 해당 방으로 이동해야하는데, 넘어가질 않음.

내일 다시 해봐야지

day5.

라우팅 : <https://velog.io/@velopert/react-router-v6-tutorial#34-route-컴포넌트로-특정-경로에-원하는-컴포넌트-보여주기>

```
(App.jsx)

import React from "react";
import { Routes, Route } from "react-router-dom";

import Start from "../component/Start";
import Invited from "../component/Invited";
import Draw from "../component/Draw/Draw";

import "../App.css";

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Start />} />
        <Route path="/invited" element={<Invited />} />
        <Route path="/draw" element={<Draw />} />
      </Routes>
    </div>
  );
}

export default App;
```

App.jsx에서 라우팅을 해주고,

Start에서 navigate를 이용해 링크를 클릭하지 않고, 페이지를 이동시키는 코드를 작성했다.

```
import { useNavigate } from "react-router-dom";

const navigate = useNavigate();

navigate("/draw");
```


! button type="submit" 문제 ⇒ 2시간 반 삽질

axios로 데이터를 주고받는데는 전혀 문제가 없었고,

페이지 라우팅도 문제가 없었지만

이상하게 방 참가하기와 방 생성하기 버튼을 눌렀을 때 동작해야하는 이벤트 핸들러가 전혀 먹히지 않았다.

chatgpt에 코드를 모두 보여줘도 이상한 말만 늘어났다.

문제는 방 참가하기와 방 생성하기 버튼에 있었다.

type이 submit으로 되어있어서 그랬다.

type="submit" 으로 설정된 버튼은 클릭하면 기본 동작인 폼 제출(submit)을 수행해서

페이지를 다시 로드하거나 새로고침하는 동작을 수행하는데,

이 때 이벤트 핸들러가 실행되지 않을 수 있다는 사실을 모르고 있었다.

브라우저 콘솔에 아무런 오류도 뜨지 않고, 서버 콘솔에는 받아온 데이터가 잘 뜨길래 더더욱 문제를 잡을 수 없었다.

방 생성하기 이벤트 핸들러에서

```
const createRoomHandler = async () => {
  axios
    .post("http://localhost:8080/create", {
      body: {
        drawid: enteredDrawid,
        nickname: enteredNickname,
      },
    })
    => 여기까지는 실행이 되고 서버에서 응답을 보내 줬지만, 페이지가 새로고침 되어버려서 응답을 받을 수 없었다.
    .then((response) => {
      console.log("create 요청성공");
      // 여기서 이미 해당 id로 존재하는지 구별 해야함.
      // 없는 방 만들때 == response.data.insertedId
      // 이미 존재하는 방일때 == response.data.roomid
      console.log("create", response.data);
      if (response.data === "") {
        alert("해당 ID로 된 방이 이미 존재합니다. 다른 ID로 생성해 주세요.");
      } else {
        console.log(response.data);
        dispatch(joinRoomInfo(response.data));
        navigate("/draw");
      }
    })
    .catch((error) => {
      console.log("create 요청실패");
      console.log(error);
    });
};
```

앞으로 버튼의 타입을 처음부터 제대로 설정해야겠다.

페이지가 새로고침 되는 것 같다면 form이 제출되는 것은 아닌지 체크할 것.

form 태그에 대해 좀 더 공부해야겠다.

이벤트 핸들러를 실행하려면 type="button" 으로 설정,

폼 제출을 위해 버튼을 사용하려면 type="submit" 으로 설정할 것.

draw 페이지 레이아웃을 먼저 만들고 기능을 하나하나 입혀야겠다.

tools의 각 버튼들을 모두 기능 개발하려면 은근히 시간 좀 걸릴 것 같다.

day6.

begin.module.css 파일에 main태그와 button태그가 전역으로 스타일이 먹혀있어서

draw 레이아웃 만드는데 애를 먹었다.

앞으로는 내가 만드는 컴포넌트는 전역으로 먹이는 일이 없도록 해야겠다.



body태그에

overflow: hidden;

을 먹이면 화면에 스크롤이 생기지 않는다.

▼ **place-content: flex-end;**



4시간을 삽질

place-content: flex-end;

draw 페이지의 레이아웃을 거의 다 완성하고,

채팅 컴포넌트(.bubbles) 안에 채팅이 설정한 height을 넘어가면 스크롤이 생기게 하려고 했지만, 어떤 이유에선지 생기질 않았다.

구글링도 해보고 gpt에도 물어 봤지만 명확한 답이 없었다.

혹시나 body에 넣은 overflow:hidden 속성이 문제 일까봐 썼다 지웠다만 500번은 한 것 같다.

부모 태그들의 높이도 모두 다 조절 해보고, html태그에 overflow:scroll도 넣어보고 다 봤다.

어쩔 수 없이 있는 속성들을 모두 하나씩 검사 하다가 **place-content: flex-end;**를 주석처리하자 **scroll**이 되기 시작했다.

(index.css)

```
body {
  margin: 0;
  overflow: hidden;
}
```

(Chat.module.css)

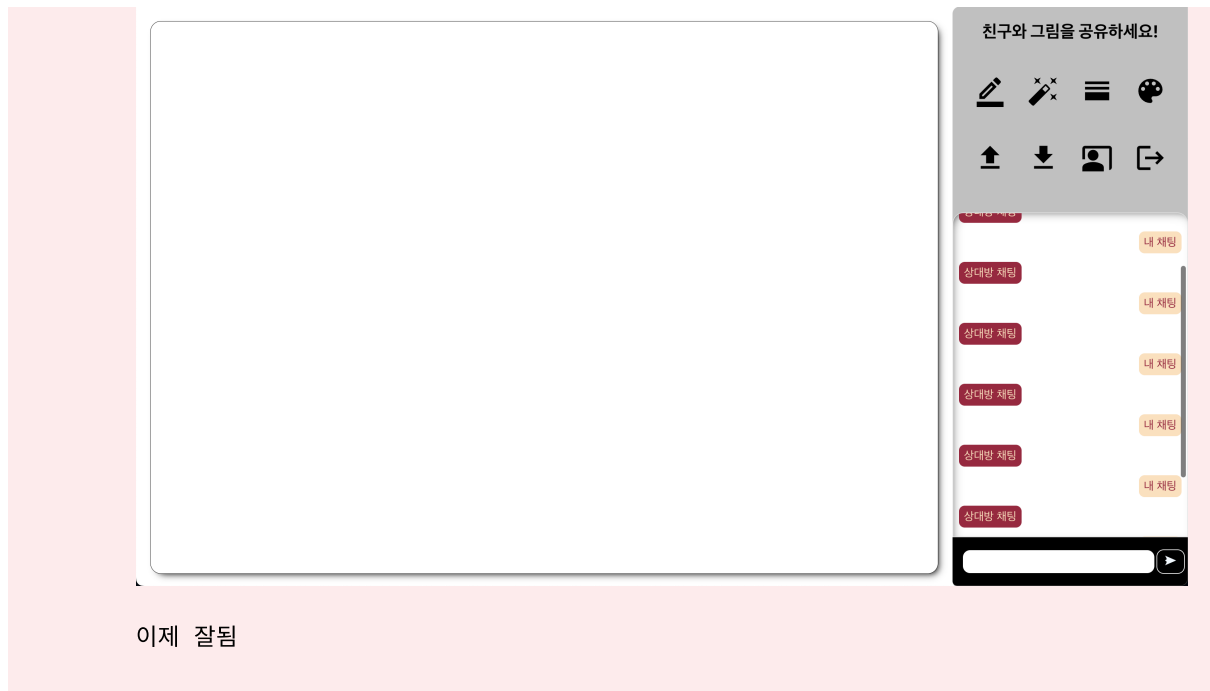
```
.bubbles {
  height: 451px;
  overflow: scroll;
  display: flex;
  flex-direction: column;
  border: 1px solid rgba(166, 166, 166, 0.345);
  border-radius: 15px 15px 0px 0px;
  /* place-content: flex-end; */
  background-color: white;
  box-shadow: 5px 5px 15px -10px inset;
}
```

```
import React from "react";

import classes from "./Chat.module.css";

const Chat = () => {
  return (
    <div className={classes.chatbox}>
      <div className={classes.chatting}>
        <div className={classes.bubbles}>
          <div className={classes.bubble}>상대방 채팅</div>
          <div className={` ${classes.bubble} ${classes.bubbleMe}`}>내 채팅</div>
          <div className={classes.bubble}>상대방 채팅</div>
        </div>
        <div className={classes.send}>
          <input type="text" />
          <button>></button>
        </div>
      </div>
    </div>
  );
};

export default Chat;
```



day7.

레이아웃 완성 후



/draw페이지 만들기

1. 그림판 그리기 기능 ⇒ 그리는 기능 먼저, 그 다음 socket으로 실시간 공유.
2. tools 각 기능 개발(그림판과 연계됨)
3. chatting 기능 ⇒ socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.

참고 :

<https://velog.io/@mokyoungg/React-React에서-Canvas-사용하기마우스-그리기>

<https://lts0606.tistory.com/599>



const canvasRef = useRef();

1. `useRef()`를 사용하여 `canvasRef`라는 Ref 객체를 만들었다.
2. 해당 객체를 DOM으로 조작하고 싶은 `canvas` 태그의 Ref 값으로 설정하였다.
3. `useEffect`를 통해, 페이지가 렌더가 될때 `canvas`에 `canvasRef`라는 ref 객체의 `.current`라는 값 할당하였다.

이제 `canvas`는 조작하려는 `canvas` 태그를 가리킨다.

`useRef` 혹은 변수를 생성하고 해당 변수를 참조하는 리액트 컴포넌트를 만들 때 사용됩니다.

초기값으로 `null`을 주는 이유는 `useRef` 혹은 생성된 변수를 참조하기 위한 용도로 사용되기 때문입니다.

초기값으로 `null`을 주면 이후에 해당 변수에 원하는 값을 할당할 수 있습니다.

`useRef` 혹은 리액트 컴포넌트의 렌더링과 관련된 데이터를 유지하고 업데이트하는 데 사용됩니다.

`useRef`를 사용하면 컴포넌트의 렌더링과 관련되지 않는 데이터를 저장하고 사용할 수 있으며, 컴포넌트의 재렌더링이 발생해도 해당 변수의 값은 변하지 않습니다.

따라서, `useRef(null)`은 초기값으로 `null`을 주어 리액트 컴포넌트 내에서 참조할 변수를 생성하기 위한 것입니다.

이후에 해당 변수에 원하는 값을 할당하거나 참조하여 사용할 수 있습니다.

`useRef()`를 호출하고 초기값을 주지 않고 사용하는 것도 가능합니다. `useRef()`를 호출하면 초기값으로 `undefined`가 할당됩니다.

```
const ref = useRef();
```

이렇게 초기값을 주지 않고 `useRef()`를 호출하면 `ref` 변수에는 초기값으로 `undefined`가 할당됩니다.

이후에 해당 변수에 원하는 값을 할당하거나 참조하여 사용할 수 있습니다.

다만, 초기값을 주지 않고 `useRef()`를 사용하는 경우에는 주의해야 할 점이 있습니다. `ref.current`에 접근할 때 `undefined`일 수 있으므로 적절한 체크가 필요합니다.

예를 들어, 다른 코드에서 `ref.current`를 사용하기 전에 유효성 검사를 수행하여 `undefined`인지 확인해야 합니다.

```
if (ref.current) {
  // ref.current를 사용하는 코드
}
```

따라서, 초기값을 주지 않고 `useRef()`를 사용할 때는 주의하여 사용해야 하며,

변수의 유효성을 검사하는 코드를 추가해야 합니다. 초기값을 명시적으로 지정하는 것이 가독성과 안전성 측면에서 더 좋은 방법입니다.

초기값으로 `null`을 넣어주는 이유는, 명시적으로 빈 레퍼런스를 생성했음을 알려주기 위함이다.



그림 그리는 것까지 성공

```

(Draw.jsx)
import React, { useRef, useState, useEffect } from "react";

import Tools from "../Tools/Tools";
import Chat from "../Chat/Chat";

import classes from "../Draw.module.css";

// import { useSelector } from "react-redux";

const Draw = () => {
  const canvasRef = useRef(); // 1. useRef()를 사용하여 canvasRef라는 Ref 객체를 만들었다.
  const [ctx, setContext] = useState();
  const [isDrawing, setIsDrawing] = useState(false);

  useEffect(() => {
    // 3. useEffect를 통해, 페이지가 렌더가 될때 canvas에 canvasRef라는 ref 객체의 .current라는 값 할당하였다.
    // 이제 canvas는 조작하려는 canvas 태그를 가리킨다.
    const canvas = canvasRef.current;
    canvas.width = 1100;
    canvas.height = 770;
    const context = canvas.getContext("2d");
    // console.log(context);
    context.lineWidth = 1;
    context.strokeStyle = "black";
    // console.log(contextRef.current);
    setContext(context);
  }, []);
  // console.log("ctx :", ctx);

  const drawing = ({ nativeEvent }) => {
    const { offsetX, offsetY } = nativeEvent;

    // console.log(offsetX, offsetY, isDrawing);
    if (isDrawing) {
      // 여기에 beginPath가 있으면 안됨,
      // isDrawing이 false일 때 moveTo로 시작점을 따라 가다가,
      // isDrawing이 true로 바뀌는 순간 beginPath로 새로운 시작점이 있어야하는데 lineTo밖에 없기 때문
      ctx.lineTo(offsetX, offsetY);
      ctx.stroke();
    } else {
      ctx.beginPath();
      ctx.moveTo(offsetX, offsetY);
    }
  };

  const startDrawing = () => {
    setIsDrawing(true);
  };
  const stopDrawing = () => {
    setIsDrawing(false);
  };

  return (
    <main className={classes.draw}>
      { /* 2. 해당 객체를 DOM으로 조작하고 싶은 canvas 태그의 Ref 값으로 설정하였다. */ }
      <canvas
        ref={canvasRef}
        className={classes.canvas}
        onMouseMove={drawing}
        onMouseDown={startDrawing}
        onMouseUp={stopDrawing}
        onMouseLeave={stopDrawing}
      ></canvas>
      <aside className={classes.side}>
        <Tools />
        <Chat />
      </aside>
    </main>
  );
};

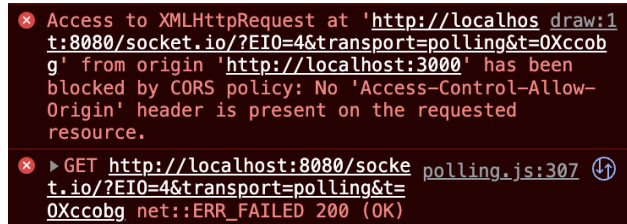
export default Draw;

```

이제 여기서 socket으로 data를 보내면 서버에서 broadcast 해주어야 하는데 cors에러 때문인지

콘솔창에 에러만 뜨고, 실행되지 않는다.

내일 다시 해볼것.



Access to XMLHttpRequest at 'http://localhost:8080/socket.io/?EIO=4&transport=polling&t=0Xccobg' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

▶ GET http://localhost:8080/socket.io/?EIO=4&transport=polling&t=0Xccobg net::ERR_FAILED 200 (OK)

day8.



/draw페이지 만들기

1. 그림판 그림 socket으로 실시간 공유.
2. tools 각 기능 개발(그림판과 연계됨)
3. chatting 기능 ⇒ socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.



cors에러

```
const io = new Server(http, {
  cors: {
    // 허용할 도메인 주소
    origin: "http://localhost:3000",
    // 허용할 HTTP 메서드
    methods: ["GET", "POST"],
  },
});
```

서버에서 CORS를 처리하는 미들웨어인 `cors()` 를 적용했지만, Socket.io를 통해 전달되는 요청에 대해서는 CORS를 처리하지 않고 있기 때문에 에러가 발생했다.

Socket.io는 HTTP를 통해 초기 연결을 설정한 후에는 실시간 양방향 통신을 위해 WebSocket을 사용하게 된다.

일반적인 HTTP 요청과는 다른 프로토콜이므로, CORS 정책에 따라 별도의 처리가 필요하다.

서버에 해당 코드를 추가해서 cors에러를 해결했다.

그림 그리는걸 다 개발하고나서 공유하는 기능을 개발하는게 아니라, 처음부터 같이 개발해야하는 것 같다.

그래서 처음부터 다시 함

참고 :

<https://github.com/socketio/socket.io/blob/master/examples/whiteboard/index.js>

<https://github.com/socketio/socket.io/blob/master/examples/whiteboard/public/main.js>



설명, 코드

기존의 drawing함수와 socket을 섞어야해서 거의 다시 만든거나 다름없다.

일단 useRef();를 이용해서 변수들을 선언해준다.

▼ 이유는



const canvasRef = useRef();

1. useRef()를 사용하여 canvasRef라는 Ref 객체를 만들었다.
2. 해당 객체를 DOM으로 조작하고 싶은 canvas 태그의 Ref 값으로 설정하였다.
3. useEffect를 통해, 페이지가 렌더가 될때 canvas에 canvasRef라는 ref 객체의 .current라는 값 할당하였다.

이제 canvas는 조작하려는 canvas 태그를 가리킨다.

useRef 혹은 변수를 생성하고 해당 변수를 참조하는 리액트 컴포넌트를 만들 때 사용된다.

초기값으로 **null**을 주는 이유는 **useRef** 혹은 생성된 변수를 참조하기 위한 용도로 사용되기 때문이다.

초기값으로 **null**을 주면 이후에 해당 변수에 원하는 값을 할당할 수 있다.

useRef 혹은 리액트 컴포넌트의 렌더링과 관련된 데이터를 유지하고 업데이트하는 데 사용된다.

useRef를 사용하면 컴포넌트의 렌더링과 관련되지 않는 데이터를 저장하고 사용할 수 있고,

컴포넌트의 재렌더링이 발생해도 해당 변수의 값은 변하지 않는다.

따라서, **useRef(null)**은 초기값으로 **null**을 주어 리액트 컴포넌트 내에서 참조할 변수를 생성하기 위한 것이다.

이후에 해당 변수에 원하는 값을 할당하거나 참조하여 사용할 수 있다.

useRef()를 호출하고 초기값을 주지 않고 사용하는 것도 가능하다.

이렇게 초기값을 주지 않고 **useRef()**를 호출하면 **ref** 변수에는 초기값으로 **undefined**가 할당되는데,

```
const ref = useRef();
```

이후에 해당 변수에 원하는 값을 할당하거나 참조하여 사용할 수 있다.

다만, 초기값을 주지 않고 **useRef()**를 사용하는 경우에는 주의해야 할 점이 있다.

ref.current에 접근할 때 **undefined**일 수 있으므로 적절한 체크가 필요하다.

예를 들어, 다른 코드에서 **ref.current**를 사용하기 전에 유효성 검사를 수행하여 **undefined**인지 확인해야 한다.

```
if (ref.current) {
  // ref.current를 사용하는 코드
}
```

따라서, 초기값을 주지 않고 **useRef()**를 사용할 때는 주의하여 사용해야 하며, 변수의 유효성을 검사하는 코드를 추가해야 한다.

초기값을 명시적으로 지정하는 것이 가독성과 안전성 측면에서 더 좋은 방법이다.

⇒ 초기값으로 **null**을 넣어주는 이유는, 명시적으로 빈 레퍼런스를 생성했음을 알려주기 위함이다.

+++

일반적으로 컴포넌트가 재랜더링될 때는 컴포넌트의 상태(state)나 속성(props)이 변경되었을 때 발생하는데

이때 `useState`를 사용하여 상태를 관리하면 상태가 변경될 때마다 컴포넌트가 재랜더링 되지만,

`useRef`로 선언된 변수는 재랜더링과 무관하게 이전 값을 유지한다.

따라서, `useRef`를 사용하면 컴포넌트가 재랜더링되어도 값이 변하지 않고 유지되기 때문에 특정 데이터를 컴포넌트의 라이프사이클 동안 계속해서 사용하고자 할 때 유용하다.

예를 들어, 이전의 값을 참조하거나 비교해야 하는 상황이나, 컴포넌트 내부에서 계속해서 갱신되는 값을 보관하고자 할 때 `useRef`를 활용할 수 있다.

`useEffect`는 한 번만 실행되는 초기 설정 작업을 수행하고, 재랜더링시 다시 실행하지 않기 위해 사용했다.

특히, `useEffect`의 의존성 배열이 `[]`로 설정되어 있어서 컴포넌트가 처음으로 렌더링될 때만 실행된다.



`useEffect` 내부에서는

1. `canvasRef`를 사용하여 실제 DOM 요소인 캔버스에 접근하고, 캔버스의 너비와 높이 설정
2. `canvas.getContext("2d")`를 사용하여 2D 그래픽 컨텍스트를 가져온 후, 해당 컨텍스트를 `ctx.current`에 할당.

이를 통해 그래픽(2d) 컨텍스트를 참조

3. `context.strokeStyle`와 `context.lineWidth`를 사용하여 그림 그리기에 필요한 속성을 설정.



주의할 것(`useEffect`)

실행 직전에 **"ctx.current.beginPath is not a function"**이라는 오류가 발생했는데

이유는 `useEffect` 내부에서 `ctx.current` 값들을 초기화 해주는데

`useEffect`는 컴포넌트가 마운트 된 직후(모든 엘리먼트가 렌더링 된 후)에 실행되기 때문에

처음 컴포넌트가 DOM에 렌더링되기 이전에 `useRef`로 생성한 참조 객체(`canvasRef`와 `ctx`)에

`ctx.current`가 `null` 값으로 초기화 되어서 당연히 `beginPath` 함수를 호출할 수 없다.

해결 방법으로는 `ctx.current`가 `null`이 아닌 경우에만 `beginPath` 함수를 호출하도록 수정해야 한다.

이전 코드에서 `ctx.current`가 `null`인 경우에는 함수 실행을 건너뛰는 로직을 추가해서

`ctx.current`가 `null`이 아닌 경우에만 해당 함수를 실행하도록 변경했다.

그 다음에

```
<canvas
  ref={canvasRef}
  className={classes.canvas}
```

캔버스 태그에 마우스에 관한 이벤트 리스너를 모두 달아준다.

```
onMouseDown={onMouseDown}
onMouseMove={onMouseMove}
onMouseUp={onMouseUp}
onMouseLeave={onMouseUp}
></canvas>
```

각 함수는 현재 마우스의 상황을 drawing함수에 파라미터로 넘겨주면서 실행한다.

직접 그림을 그리는 동작은 수행하지 않는다.

대신 isDrawing.current를 통해 drawing함수를 실행 할지 말지를 결정한다.

onMouseDown 함수에서는

isDrawing.current = true;로 바꿔주고

시작점을 drawing 함수로 전송해준다.

onMouseMove 함수에서 밑에있는

current.current.x = offsetX;

current.current.y = offsetY;의 역할은

그리기가 완료되면 현재 위치(`current.current.x` , `current.current.y`)를 업데이트하여

다음 이벤트에서 이전 위치를 현재 위치로 사용해서 선이 부드럽게 이어지게 한다.

onMouseUp 함수에서는

isDrawing.current = false;만 실행해주면 그림 그리기가 중단된다.

drawing함수 내부에서 그림 그리는 동작을 수행하고,

동시에 socket.current.emit으로 서버에 "drawing"이라는 이벤트 이름으로

현재 x,y좌표와 움직인 후의 x,y좌표 그리고 stroke 색깔, 두께 까지 모두 전송한다.

서버는 받은 데이터를 socket.broadcast.emit("drawing", data);로 뿌려준다.(나중에는 방마다 분할해서 뿌려야한다.)

그러면 Draw.jsx가 socket.current.on("drawing", onDrawingEvent);로 다시 받아서 onDrawingEvent 함수를 실행한다.

```
const onDrawingEvent = (data) => {
  drawing(data.x0, data.y0, data.x1, data.y1, data.color, data.lineWidth);
};
```

이 함수는 내가 마우스를 직접 조작하지 않아도 drawing 함수를 실행해서 그림을 그려준다.

즉, 내가 그리는 그림이 상대방 화면에도 똑같이 그려진다.

▼ 전체 코드(Draw.jsx, server.js)

```
(Draw.jsx)

import React, { useRef, useEffect } from "react";
import io from "socket.io-client"; // Client Socket

import Tools from "../Tools/Tools";
import Chat from "../Chat/Chat";

import classes from "../Draw.module.css";

// import { useSelector } from "react-redux";

const Draw = () => {
```

```

const canvasRef = useRef(null); // 1. useRef()를 사용하여 canvasRef라는 Ref 객체를 만들었다.
const ctx = useRef(null);
const isDrawing = useRef(false);
// const colorsRef = useRef([]);
const current = useRef({
  color: "black",
  lineWidth: 1,
});
const socket = useRef(null);
socket.current = io();

useEffect(() => {
  // 3. useEffect를 통해, 페이지가 렌더가 될때 canvas에 canvasRef라는 ref 객체의 .current라는 값 할당하였다.
  // 이제 canvas는 조작하려는 canvas 태그를 가리킨다.
  const canvas = canvasRef.current;
  canvas.width = 1100;
  canvas.height = 770;
  const context = canvas.getContext("2d");
  // console.log(context);
  context.strokeStyle = current.current.color;
  context.lineWidth = current.current.lineWidth;
  // console.log(contextRef.current);
  ctx.current = context;
}, []);
// console.log("ctx :", ctx);

const onDrawingEvent = (data) => {
  drawing(data.x0, data.y0, data.x1, data.y1, data.color, data.lineWidth);
};

const drawing = (x0, y0, x1, y1, color, lineWidth, emit) => {
  if (!ctx.current) return; // => useRef(null); 했을 때 여기서 중요!!!

  ctx.current.beginPath();
  ctx.current.moveTo(x0, y0);
  ctx.current.lineTo(x1, y1);
  ctx.current.strokeStyle = color;
  ctx.current.lineWidth = lineWidth;
  ctx.current.stroke();
  ctx.current.closePath();

  if (!emit) return;

  socket.current.emit("drawing", {
    x0: x0,
    y0: y0,
    x1: x1,
    y1: y1,
    color: color,
    lineWidth: lineWidth,
  });

  // socket.current.on("drawing", onDrawingEvent);
};

const onMouseDown = ({ nativeEvent }) => {
  const { offsetX, offsetY } = nativeEvent;
  isDrawing.current = true;
  current.current.x = offsetX;
  current.current.y = offsetY;
};

const onMouseMove = ({ nativeEvent }) => {
  const { offsetX, offsetY } = nativeEvent;
  if (!isDrawing.current) return;
  drawing(
    current.current.x,
    current.current.y,
    offsetX,
    offsetY,
    current.current.color,
    current.current.lineWidth,
    true
  );
  current.current.x = offsetX;
  current.current.y = offsetY;
};

const onMouseUp = () => {
  isDrawing.current = false;
};

```

```

    });

    socket.current.on("drawing", onDrawingEvent);

    return (
      <main className={classes.draw}>
        { /* 2. 해당 객체를 DOM으로 조작하고 싶은 canvas 태그의 Ref 값으로 설정하였다. */ }
        <canvas
          ref={canvasRef}
          className={classes.canvas}
          onMouseDown={onMouseDown}
          onMouseMove={onMouseMove}
          onMouseUp={onMouseUp}
          onMouseLeave={onMouseUp}
        ></canvas>
        <aside className={classes.side}>
          <Tools />
          <Chat />
        </aside>
      </main>
    );
  });
};

export default Draw;

```

```

(server.js)

io.on("connection", (socket) => {
  console.log("연결되었습니다");
  socket.on("drawing", (data) => {
    socket.broadcast.emit("drawing", data);
  });
});

```

day9.



/draw페이지 만들기

1. ~~그림판 그림 socket으로 실시간 공유~~ ⇒ 방을 나누는 것은 chatting까지 개발 후에 하기로 했다.
2. tools 각 기능 개발(그림판과 연계됨) ⇒ -개발중-
3. chatting 기능 ⇒ socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.
4. 방 나누기

```

(Tools.jsx)
-- 위아래 나머지 부분은 너무 길어서 생략했다--

const Tools = (props) => {
  const [activeButton, setActiveButton] = useState(null);
  const [lineWidth, setLineWidth] = useState(false);
  const [chooseColor, setChooseColor] = useState(false);

  const handleButtonClick = (buttonId) => {
    setActiveButton(buttonId);
    if (buttonId !== "stroke") {
      setLineWidth(false);
    }
    if (buttonId !== "palette") {

```

tooltip과 아이콘, 누른 아이콘에만 따로 주는 class가 있어야해서 코드가 좀 길고 보기가 좋지 않지만 어쩔 수 없음

pen과 eraser는 코드가 간단해서 따로 파일을 분할 하지는 않았다.

draw.jsx에서 props로 변경함수를 받아서 연필과 지우개를 실행했

```

        setChooseColor(false);
    }
};

const lineWidthToggle = () => {
    if (!lineWidth) {
        setLineWidth(true);
    } else {
        setLineWidth(false);
    }
};

const chooseColorToggle = () => {
    if (!chooseColor) {
        setChooseColor(true);
    } else {
        setChooseColor(false);
    }
};

return (
    <nav className={classes.toolbox}>
        <h2>친구와 그림을 공유하세요!</h2>
        <section className={classes.tools}>
// 연필
            <div
                className={` ${classes.parts} ${
                    activeButton === "pen" ? classes.selected : null
                }`}
                onClick={() => {
                    handleButtonClick("pen");
                    props.onStationery("pen");
                }}
            >
                <Tooltip title="pen" arrow placement="top">
                    <BorderColorOutlinedIcon sx={{ fontSize: 45 }} />
                </Tooltip>
            </div>
// 지우개
            <div
                className={` ${classes.parts} ${
                    activeButton === "eraser" ? classes.selected : null
                }`}
                onClick={() => {
                    handleButtonClick("eraser");
                    props.onStationery("eraser");
                }}
            >
                <Tooltip title="eraser" arrow placement="top">
                    <AutoFixHighIcon sx={{ fontSize: 45 }} />
                </Tooltip>
            </div>
// 선긋기
            <div
                className={` ${classes.parts} ${
                    activeButton === "stroke" ? classes.selected : null
                }`}
                onClick={() => {
                    handleButtonClick("stroke");
                    lineWidthToggle();
                }}
            >
                <Tooltip title="stroke" arrow placement="top">
                    <HorizontalSplitIcon sx={{ fontSize: 45 }} />
                </Tooltip>
                {lineWidth ? (
                    <Stroke onChangeStrokeWidth={props.onChangeStrokeWidth} />
                ) : (
                    false
                )}
            </div>
// 팔레트
            <div
                className={` ${classes.parts} ${
                    activeButton === "palette" ? classes.selected : null
                }`}
                onClick={() => {
                    handleButtonClick("palette");
                    chooseColorToggle();
                }}
            >

```

다.

stroke와 palette는 방법이 비슷했다.

대신 stroke는 기본 태그인 input을 썼고,

palette는 react-color 라이브러리를 사용해서, ChromePicker를 가져왔다.

둘 다 값을 추출한 다음 draw.jsx로 보내서 current 값을 변경 시켜줬다.

문제점이 있는데

둘 다 클릭하는 div내부에 있어서 누른 채로mousedown하고 있지 않으면 띄워 놓은 picker가 사라진다는 점(둘 다 같은 문제)

ChromePicker는 색을 고를때 ChromePicker내부 ui가 움직이지 않는 것이다.

내일 마저 고치고

tools박스 내부의 나머지 기능들도 개발해 봐야겠다.

```

    }}
  >
  <Tooltip title="palette" arrow placement="top">
    <ColorLensIcon sx={{ fontSize: 45 }}></ColorLensIcon>
  </Tooltip>
</div>
{chooseColor ? (
  <Palette onChangeStrokeColor={props.onChangeStrokeColor} />
) : (
  false
)}
</section>

```

```

(Draw.jsx)
...윗부분 생략...

// 연필, 지우개 함수
const stationeryHandler = (stationery) => {
  if (stationery === "pen") {
    current.current.color = "black";
  } else {
    current.current.color = "white";
  }
};

// 선굵기 함수
const changeStrokeWidth = (e) => {
  current.current.lineWidth = e.target.value;
};

// 선 색깔 함수
const changeStrokeColor = (color) => {
  console.log(color.hex);
  current.current.color = `${color.hex}`;
};

socket.current.on("drawing", onDrawingEvent);

return (
  <main className={classes.draw}>
    { /* 2. 해당 객체를 DOM으로 조작하고 싶은 canvas 태그의 Ref 값으로 설정하였다. */ }
    <canvas
      ref={canvasRef}
      className={classes.canvas}
      onMouseDown={onMouseDown}
      onMouseMove={onMouseMove}
      onMouseUp={onMouseUp}
      onMouseLeave={onMouseUp}
    ></canvas>
    <aside className={classes.side}>
      // 여기서 함수를 실행해서 current값을 바꾸는게 편하기 때문에 props로 함수를 넘겨 주었다.
      <Tools
        onStationery={stationeryHandler}
        onChangeStrokeWidth={changeStrokeWidth}
        onChangeStrokeColor={changeStrokeColor}
      />
      <Chat />
    </aside>
  </main>
);
};

export default Draw;

```

day.10(230601)



/draw페이지 만들기

1. 그림판 그림 ~~socket으로 실시간 공유~~ ⇒ 방을 나누는 것은 chatting까지 개발 후에 하기로했다.
2. tools 각 기능 개발(그림판과 연계됨) ⇒ -문제점 수정중-



문제 해결:

누른 채로 mousedown하고 있지 않으면 띄워 놓은 picker가 사라지는 문제

⇒ picker div박스 자체를 아이콘 div박스로 뺐다.

ChromePicker내부 ui가 움직이지 않는 문제

⇒ ChromePicker말고 SwatchesPicker로 바꿨다. 내 코드 안에서 움직이는데 있고 안움직이는데 있는데 정확한 이유를 찾아보고 할 수 있는 시도는 다 해봤지만 움직이게하는데 실패했다.

3. chatting 기능 ⇒ socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.
4. 방 나누기

```
(Draw.jsx)

useEffect(() => {
  ...
  socket.current.on("imageUploaded", handleImageUpload);

  return () => {
    // 컴포넌트가 언마운트될 때 이벤트 핸들러를 정리, 메모리 누수 차단
    socket.current.off("imageUploaded", handleImageUpload);
  };
}, []);
// console.log("ctx :", ctx);

const handleImageUpload = (imageData) => {
  const image = new Image();
  image.src = imageData;
  image.onload = function () {
    ctx.current.drawImage(image, 50, 50, 300, 300);
  };
};

const onUploadFile = (e) => {
  const file = e.target.files[0];
  const reader = new FileReader();
  // 파일 읽기 완료 후 수행할 동작
  reader.onload = (event) => {
    const imageData = event.target.result; // 이미지 데이터
    socket.current.emit("uploadImage", imageData);
  };
  reader.readAsDataURL(file); // 파일 읽기 시작
};

const onDownload = () => {
  const url = canvasRef.current.toDataURL();
  const a = document.createElement("a");
  a.href = url;
  a.download = "myDrawing.png";
  a.click();
};
```

: nomad coders paintjs 강의 참고

사진 업로드 기능과, 그림판에 그린 그림을 다운로드하는 기능을 추가했다.

업로드 할 때 원래는 이미지 url을 만들어서 drawing함수에 매개 변수로 넘겨 주어야하나 라고 생각했지만 drawing는 이미 매개변수가 많고 내부에서 어떻게 처리해야할지 몰라서 이미지는 따로 코드를 만들었다.

서버에서도 그림 공유와 이미지 공유는 따로 처리한다.



이미지 공유

1. 클라이언트에서 이미지를 업로드한 후, 이미지 데이터를 서버로 전송
2. 서버 측에서 socket으로 이미지 데이터를 수신하여 모든(나 포함) 클라이언트에게 전달
3. 클라이언트에서 socket으로 이미지 데이터를 수신하여 canvas에 그리기


```
URL.createObjectURL();

.onload

.click() 같은 처음보는 함수도 많았는데
핵심은 html 태그 내부 속성을 js로 다룰
수 있다는 것이다.
```

다음은 초대하기와 방 나가기 기능인데 둘 다 방 나누기 기능과 연계되어 있어서 방 나누기 먼저 기능 개발한 후에 만들어야겠다.



`const reader = new FileReader()`

`FileReader`는 웹 브라우저에서 제공하는 API 중 하나로, 비동기적으로 파일의 내용을 읽을 수 있게 해주는 객체이다.

`FileReader`를 사용하면 클라이언트 측에서 파일을 선택하고 해당 파일의 내용을 읽을 수 있다.

`FileReader`를 사용하여 파일을 읽는 과정은:

1. `FileReader` 객체를 생성: `const reader = new FileReader();`
2. `FileReader` 객체의 `onload` 이벤트 핸들러 후 설정 파일 읽기가 완료되면 이 이벤트 핸들러가 호출된다.

```
javascriptCopy code
reader.onload = function(event) {
  // 파일 읽기 완료 후 수행할 동작
  const fileContent = event.target.result; // 파일 내용은 event.target.result에 저장된다.
  // 파일 내용을 활용하여 추가적인 작업을 수행할 수 있다.
};
```

3. `File` 객체를 선택하고, `FileReader` 객체의 `readAsDataURL()` 메서드를 호출하여 파일을 읽는다.

```
javascriptCopy code
const file = inputElement.files[0]; // 파일 선택을 위해 input 요소를 사용하거나 다른 방법으로 File 객체를 얻는다.
reader.readAsDataURL(file); // 파일 읽기 시작
```

`readAsDataURL()` 메서드는 선택한 파일을 데이터 URL로 읽어온다.

데이터 URL은 파일의 내용을 Base64 인코딩된 문자열로 표현한 것이다.

4. 파일 읽기가 완료되면 `onload` 이벤트 핸들러가 호출된다. 이 핸들러 내부에서 파일의 내용을 활용할 수 있다.

`FileReader`를 사용하면 클라이언트 측에서 선택한 파일의 내용을 비동기적으로 읽을 수 있으며, 읽은 내용을 활용하여 원하는 작업을 수행할 수 있다.

Start.jsx에서 방을 만들 때 서버에서 db에 저장된 id를 받아서 socket으로 넘겨 주어야한다.

socket은 받은 id에 해당하는 방에만 그림이나 채팅을 broadcast해주어야 한다.

방 안에서 그림을 그리다가 멈추면 그 그림은 DB에 저장된다.

저장된 그림은 새로운 유저가 방에 접속하면 불러와서 보여진다.



/draw페이지 만들기

1. ~~그림판 그림 socket으로 실시간 공유~~ ⇒ 방을 나누는 것은 chatting까지 개발 후 할 것
2. ~~tools 각 기능 개발(그림판과 연계됨)~~ ⇒ 초대하기와 방 나가는 방 나누기 개발 후 할 것
3. 방 나누기
4. chatting 기능 ⇒ socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.
5. 방 참가자 표시(멤버 이름)

Start컴포넌트에서 axios를 날린 후 dispatch를 통해 store.js에 있는 state값(_id, 방id, 멤버)을 업데이트 하면

Draw 컴포넌트로 이동 후 자동으로 state값이 업데이트되도록 하려고 했지만,

다른 유저가 접속했을 때는 또 다른 store가 생성되기 때문에 같은 state를 공유할 수 없었다.

따라서 방에 접속중인 멤버들을 실시간으로 보여주려면 Redux가 아닌 db(MongoDB change stream)나 socket을 이용해야겠다.



Redux state

Redux에서 상태 값을 업데이트하면 **Draw** 컴포넌트에서도 자동으로 해당 상태 값이 업데이트됩니다. 이는 Redux의 핵심 원리 중 하나인 "단일 소스 진리 (Single Source of Truth)" 개념에 기반합니다.

Redux에서 상태는 단일 스토어에 저장되며, 컴포넌트들은 이 스토어에서 상태 값을 선택하고 업데이트될 때마다 이를 구독합니다. 따라서 다른 컴포넌트에서 **dispatch**를 통해 상태 값을 업데이트하면, **Draw** 컴포넌트에서도 자동으로 업데이트된 상태 값을 수신합니다.

이를 가능하게 하는 핵심은 **useSelector** 훅입니다. **useSelector** 혹은 Redux 스토어의 상태 값을 선택하고, 해당 상태 값이 변경될 때 컴포넌트를 리렌더링합니다. 따라서 다른 컴포넌트에서 상태 값을 업데이트하면, **useSelector**를 사용하여 해당 상태 값을 구독하는 **Draw** 컴포넌트도 업데이트됩니다.

방 참가중인 멤버나 방 내부 채팅이나 뭘 하든 방 정보를 가져와서 해당 방 id에 따라 실시간 broadcast해야한다는 걸 이제 알았다.
방 정보 가져오기 먼저 해야겠다.

처음에는 방 정보를 위 처럼 store에 저장 후 /draw로 이동한 뒤에 다시 가져오거나,
/draw에서 다시 axios를 날려야 하나 라고 생각했다.

근데 의외로 가까운 곳에 방법이 있었다.

```
// Start 컴포넌트
import { useNavigate } from "react-router-dom";
```

navigate함수를 사용해서 페이지를 이동할 때

```
const Start = () => {
  const navigate = useNavigate();
  const joinRoomHandler = async () => {
    // ...
    if (response.data === "") {
      alert("해당 ID로 된 방이 없습니다. 방을 새로 생성하세요.");
    } else {
      console.log("join", response.data);
      const roomData = response.data; // 전달할 데이터
      navigate("/draw", { state: roomData }); // /draw 페이지로 이동하면서 데이터 전달
    }
  };
};
```

받아온 정보를 같이 넘길 수 있었다.

state 객체로 함께 넘길 수 있었다.

```
// Draw 컴포넌트

import { useLocation } from "react-router-dom";

const Draw = () => {
  const location = useLocation();
  const roomData = location.state; // 전달된 데이터
  // 사용할 데이터를 이용하여 로직 수행
  // ...
};
```

그리고 `/draw` 페이지에서는 `location.state`를 통해 데이터를 받을 수 있다. `useLocation` 혹은 사용하여 현재 경로의 `location` 객체에 접근하고,

`location.state`를 통해 데이터를 가져올 수 있다.

`roomData`를 이용해 하고싶은 작업을 하면 된다.

드디어 socket으로 방 나누기 성공

```
// Draw.jsx 추가한 코드
...

const location = useLocation();
const roomData = location.state; // 전달된 데이터

...

socket.current.emit("joinroom", roomData.roomid);

...

socket.current.emit("drawing", {
  x0: x0,
  y0: y0,
  x1: x1,
  y1: y1,
  color: color,
  lineWidth: lineWidth,
  roomNum: roomData.roomid,
});

...

const onUploadFile = (e) => {
  const file = e.target.files[0];
  const reader = new FileReader();
  // 파일 읽기 완료 후 수행할 동작
  reader.onload = (event) => {
    const imageData = { // 이미지 데이터
      image: event.target.result,
      roomNum: roomData.roomid,
    };
    // 서버로 이미지 데이터 전송
```

`joinroom`이라는 이벤트로 페이지 이동 시에 받아온 `roomid`를 서버에 넘겨주었다.

첫 렌더링 시에 유저가 접속한 `roomid`로 방에 접속 될 것이고 `drawing` 함수가 실행 할 때도 해당 `roomid`를 가진 방에만 그림이 공유될 것이다.

이미지 데이터도 해당 `roomid`를 가진 방에만 공유될 것이다.

사실 클라이언트 측에서는 `roomid`를 넘겨주고,

서버에서 `io.to().emit()`로 방을 나눠주는 것이다.

```

        socket.current.emit("uploadImage", imageData);
    };
    reader.readAsDataURL(file); // 파일 읽기 시작
};

```

```

// server.js

io.on("connection", (socket) => {
  socket.on("joinroom", (data) => {
    console.log("방 참가 되었어요", data);
    socket.join(`${data}`);
  });
  socket.on("drawing", (data) => {
    // 현재 소켓을 제외한 모든 연결된 클라이언트에게 이벤트와 데이터를 전송
    // 그림을 그리는 나는 내가 그리는 그림을 중복해서 볼 필요없음
    // socket.broadcast.emit("drawing", data);
    io.to(`${data.roomNum}`).emit("drawing", data);
  });
  socket.on("uploadImage", (imageData) => {
    // 나 포함 모든 클라이언트에게 이미지 데이터 전송
    // 나도 공유되는 이미지가 보여야한다.
    io.to(`${imageData.roomNum}`).emit("imageUploaded", imageData.image);
  });
  // socket.on("joinroom", (data) => {
  //   socket.join();
  // });
});

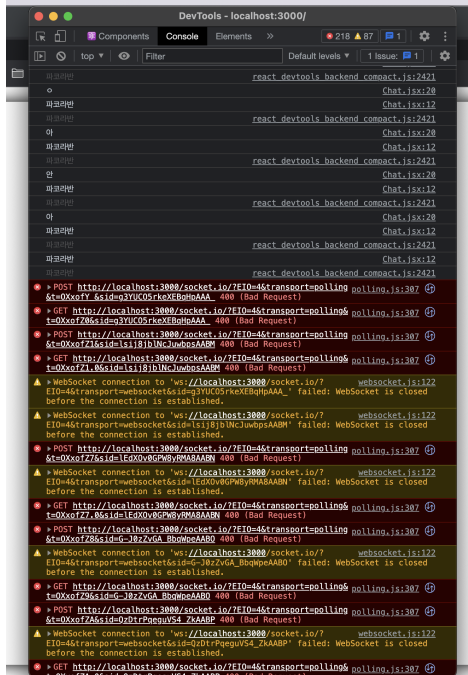
```

첫 렌더링 시에 받은 joinroom 이벤트에 대한 동작으로 방을 생성해준다.

- data에는 roomid가 들어있다.

drawing과 uploadImage 이벤트에 대한 동작으로 to() 메서드에 인자로 받은 roomid를 넣어주어서 해당 roomid를 가진 방에만 데이터를 emit해준다.

큰 기능 중 마지막인 채팅기능을 개발하고 있지만 잘 안된다.



뭔가 잘 되는듯하다가 에러가 300개씩 떠버린다.

이유는 아직 찾지 못했다.

내일 다시 해봐야지

day12 230603

방 나누기라는 큰 산을 넘었다.



/draw페이지 만들기

1. 그림판 그림 ~~socket으로 실시간 공유~~ ⇒ 방을 나누는 것은 chatting까지 개발 후 할 것
2. ~~tools~~ 각 기능 개발(그림판과 연계됨) ⇒ 초대하기와 방 나가기는 방 나누기 개발 후 할 것
3. 방 나누기
 - start에서 방 번호, 참여자가 설정한 닉네임을 draw와 chat에 넘겨주고 socket에 담아 서버와 통신한다.
 - 서버는 받은 데이터 중에 방id를 구별해서 해당id를 가진 방에만 그림과 chat을 공유한다.
4. chatting 기능 ⇒ ~~socket으로 구현, db에서 방, 유저 정보 가져와야함~~, 여러명일 수 있기 때문에 상대 구별해야함.
 - db에서 가져오지 않고 처음 입력 후 페이지 이동시 props로 직접 넘겨주었다.
 - 상대방을 구별하지만 채팅시 닉네임이 뜨지 않아서 누가 입력한 채팅인지 모르는 문제 수정해야함
5. 방 참가자 표시(멤버 이름) ⇒ 공간이 없어서 초대하기 버튼을 누르면 모달로 띄우고 모달 하단에 초대링크를 표시해야겠다.

어제 생긴 오류를 해결 했다.

socket을 draw에 선언해서 만들어 놓고, chat에 하나 더 선언했던 것이다.

```
// draw
// 선언 후
// 서버가 만들어 놓은 socket에 접속

const socket = useRef(null);
socket.current = io();
```

```
// chat
// 선언 후
// 서버가 만들어 놓은 socket에 접속

const socket = useRef(null);
socket.current = io();
```

두 군데에 똑같은 걸 선언해 놔다.

서버에서 데이터를 받긴하는데 다시 돌려줄 때 어느 곳으로 돌려줄지 몰라서 그랬던 것 같다.

그래서 draw에 만들어 놓은 socket을 props로 chat에 넘겨주었다.

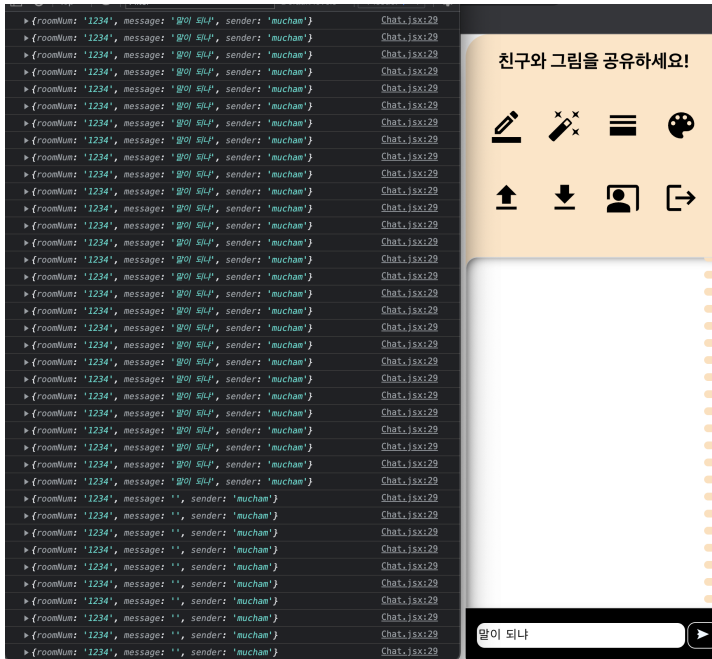
```
<Chat
  roomid={roomData.roomid}
  user={roomData.user}
  socket={socket.current}
/>
```

근데 문제는

버튼을 눌렀는데 너무 많이 출력된다.

⇒ useEffect 내부에 넣어서 해결

아마 렌더링 문제인 것 같다.



```
const userSend = (e) => {
  setMessage(e.target.value);
};

const onSendMessage = () => {
  // console.log(message.current);
  // 빈 채팅 보내는 것 막기
  if (message !== "") {
    props.socket.emit("chatting", {
      roomNum: props.roomid,
      message: message,
      sender: sender,
    });
  }
  setMessage(""); // 메시지 전송 후에 message 상태 초기화
};

const onAddMessage = (chatData) => {
  console.log(chatData); // 받은 chatData 출력
  setMessages((prevMessages) => [...prevMessages, chatData]);
};

useEffect(() => {
  props.socket.on("chatting", onAddMessage);
  return () => {
    // 컴포넌트가 언마운트될 때 이벤트 핸들러 정리
    props.socket.off("chatting", onAddMessage);
  };
}, [props.socket]);

...

<div className={classes.send}>
  <input
    id="chatting"
    type="text"
    placeholder="채팅을 입력하세요."
    value={message}
    onChange={userSend}
  />
  <button onClick={onSendMessage}>></button>
</div>
```

useEffect는 컴포넌트가 렌더링된 후에 비동기적으로 실행되는 효과를 가지고 있다.

이는 **useEffect** 내부의 코드가 컴포넌트의 렌더링이 완료된 이후에 실행된다는 것을 의미하는데

socket.current.on("chatting", (chatData) => { ... }) 함수가 **useEffect** 내부에 있으면, 해당 이벤트 리스너가 컴포넌트가 렌더링되고 난 후에 등록되기 때문에 메시지가 한 번만 출력된다.

이는 **useEffect**의 의존성 배열이 비어 있기 때문에 최초 렌더링 시에만 실행되기 때문이다.

반면에, **socket.current.on("chatting", (chatData) => { ... })** 함수를 **useEffect** 내부에 넣지 않으면, 해당 이벤트 리스너가 컴포넌트가 렌더링될 때마다 매번 등록되기 때문에 메시지가 여러 번 출력되게 된다.

엔터키 입력도 감지해서 메시지를 보낼 수 있게했다.

handleOnKeyPress 함수 추가.

처음에는 **click** 이벤트를 만들었을 때처럼 **button** 태그에 이벤트를 등록해

야 될거라고 생각했다. 안되는 게 당연하다.

사실 우리가 보통 로그인할 때를 생각해보면 아이디/비밀번호를 입력한다음에 엔터를 누른다.

즉, 키보드이벤트는 사용자로부터 입력 받은 값을 `keyCode`에 저장해서, 그게 `enter`의 키코드 값과 일치하는지 확인하는 것이기 때문에 `input` 태그에 등록해야한다.

전체 코드

```
import React, { useEffect, useState } from "react";

import classes from "./Chat.module.css";

const Chat = (props) => {
  const [message, setMessage] = useState("");
  const [messages, setMessages] = useState([]); // 채팅 메시지들을 배열에 담아 상태로 관리
  const sender = props.user;
  // console.log(sender);

  const userSend = (e) => {
    setMessage(e.target.value);
  };

  const onSendMessage = () => {
    // 빈 채팅 보내는 것 막기
    if (message !== "") {
      props.socket.emit("chatting", {
        roomNum: props.roomid,
        message: message,
        sender: sender,
      });
    }
    setMessage(""); // 메시지 전송 후에 message 상태 초기화
  };

  const handleOnKeyPress = (e) => {
    if (e.key === "Enter") {
      onSendMessage(); // Enter 입력이 되면 클릭 이벤트 실행
    }
  };

  const onAddMessage = (chatData) => {
    console.log(chatData); // 받은 chatData 출력
    setMessages((prevMessages) => [...prevMessages, chatData]);
  };

  useEffect(() => {
    props.socket.on("chatting", onAddMessage);
    return () => {
      // 컴포넌트가 언마운트될 때 이벤트 핸들러 정리
      props.socket.off("chatting", onAddMessage);
    };
  }, [props.socket]);

  return (
    <div className={classes.chatbox}>
      <div className={classes.chatting}>
        <div className={classes.bubbles}>
          {messages.map((chatData, i) => (
            <div
              key={chatData.roomNum + i}
              className={` ${classes.bubble} ${
                chatData.sender === sender ? classes.bubbleMe : null
              }`}
            >
              <div style={{ fontSize: "2px" }}>{chatData.sender}</div>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
};
```

```

        {chatData.message}
      </div>
    )}}
    {/* <div className={classes.bubble}>상대방 채팅</div>
    <div className={` ${classes.bubble} ${classes.bubbleMe}`}>내 채팅</div> */}
  </div>
  <div className={classes.send}>
    <input
      id="chatting"
      type="text"
      placeholder=" 채팅을 입력하세요."
      value={message}
      onChange={userSend}
      onKeyDown={handleOnKeyPress}
    />
    <button onClick={onSendMessage}>>></button>
  </div>
</div>
</div>
);
};

export default Chat;

```

거의 다 개발하고 수정했는데

또 css가 먹통임;; 채팅창 스크롤이 되지 않는다.

반드시 해결 해야함.

justify-content: flex-end; 를 없애니까 되긴된다.

근데 새로운 메시지가 오면 밑에 생성되는데 스크롤이 따라가질 않는다.



채팅 구현시 스크롤 아래로 내리기

장장 두 시간이 넘는 삽질끝에 해결했다.

scrollIntoView라는 함수까지 있는 것보면 애초에 css로는 해결되지 않는 문제인 듯하다.

한시간 반 정도는 어떻게든 css로 해결하려고 했다. ⇒ 실패

gpt에도 물어봄, 근데 jsx코드 안에서 const로 변수 선언함. ⇒ 실패

결국 구글링으로 해결 :

<https://velog.io/@wlv199/React로-채팅앱-구현-시-스크롤-맨-아래로-내리는-법>

1. 빈 div태그를 제일 밑에 깔고 ref값을 설정해준다.
2. useEffect에 messages를 종속성 배열에 넣고, 아래처럼 작성해준다. `scrollIntoView()` 메소드는 자신이 호출된 요소가 사용자에게 표시되도록 상위 컨테이너를 스크롤한다.

```
const messageEndRef = useRef(null);

const onAddMessage = (chatData) => {
  console.log(chatData); // 받은 chatData 출력
  setMessages((prevMessages) => [...prevMessages, chatData]);
};

useEffect(() => {
  props.socket.on("chatting", onAddMessage);
  messageEndRef.current.scrollIntoView({ behavior: "smooth" });
  return () => {
    // 컴포넌트가 언마운트될 때 이벤트 핸들러 정리
    props.socket.off("chatting", onAddMessage);
  };
}, [props.socket, messages]);

...

<div ref={messageEndRef}></div>
```

방 참가자 표시(멤버 이름)

멤버 버튼을 누르면 방 참가자 모달이 뜬다.

draw에서 입력받은 유저를 넘겨주는 것까지 했는데, 아마 배열에 저장한 다음 모달 내부에서 렌더링 해야할 것 같다.

day13 230605



/draw페이지 만들기

1. 그림판 그림 ~~socket으로 실시간 공유~~

- 방을 나누는 것은 chatting까지 개발 후 할 것 ⇒ 완료

2. ~~tools~~ 가 기능 개발(그림판과 연계됨)

- 초대하기와 방 나가는 방 나누기 개발 후 할 것 ⇒ 완료

3. 방 나누기

- start에서 방 번호, 참여자가 설정한 닉네임을 draw와 chat에 넘겨주고 socket에 담아 서버와 통신한다. ⇒ 완료
- 서버는 받은 데이터 중에 방id를 구별해서 해당id를 가진 방에만 그림과 chat을 공유한다. ⇒ 완료

4. ~~chatting~~ 기능 ⇒ ~~socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.~~

- db에서 가져오지 않고 처음 입력 후 페이지 이동시 props로 직접 넘겨주었다. ⇒ 완료
- 상대방을 구별하지만 채팅시 닉네임이 뜨지 않아서 누가 입력한 채팅인지 모르는 문제 수정해야함 ⇒ 수정 완료

5. 방 참가자 표시(~~멤버 이름~~)

- 공간이 없어서 초대하기 버튼을 누르면 모달로 띄우고 모달 하단에 초대링크를 표시해야겠다. ⇒ 완료

6. 방 나가기 클릭 시

- 진짜 나갈건지 모달로 묻기, 나가면 start로 이동하면서 이전 경로 폐기 or 저장 X ⇒ 완료
- db에서 멤버 삭제 ⇒ 완료

7. 방 중도 입장 시 그리고 있던 그림, 채팅 불러오기

- 채팅은 db에 따로 저장
- 이미지도 그림 데이터를 넘겨 주었던 것처럼 데이터를 db에 저장한 다음 접속하면 불러와야 할 것 같다.

현재 url정보 가져오기 :

<https://velog.io/@nemo/useLocation>

```

{pathname: '/lab/1', search: '', hash: '', state: undefined}
  hash: ""
  pathname: "/lab/1"
  search: ""
  state: undefined
  ▶ [[Prototype]]: Object

```

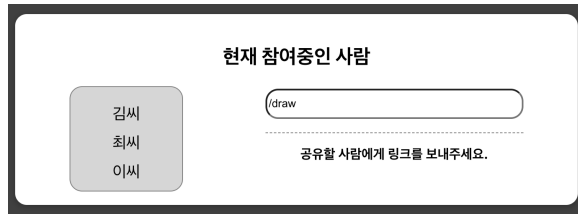
useLocation을 이용하면 이런 정보들을 얻을 수 있다.

이전 페이지에서 state에 데이터를 넘겨줄 수도 있고, replace를 이용해 뒤로가지를 막을 수도 있다.

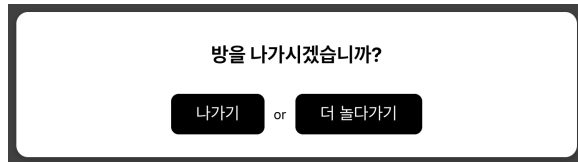
```

navigate("/draw", { state: roomData,
replace: true });

```



참가자를 확인하고, 방 링크를 복사 가능하게해서 공유할 수 있게 만들었다.

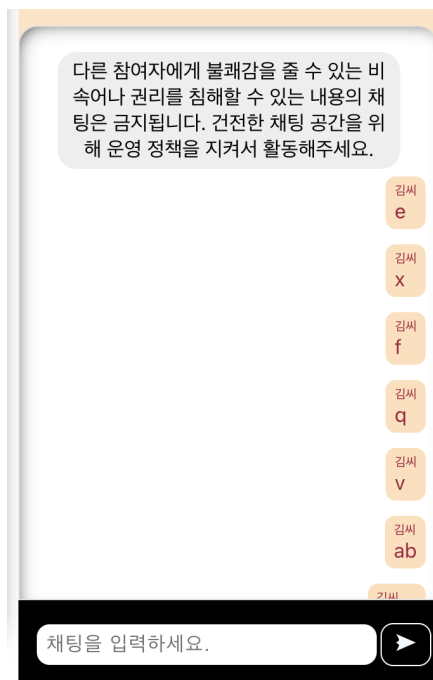


exit 버튼도 만들었다.

더 놀다가기를 클릭하면 exit모달은 다시 사라지고, 나가기를 클릭하면 start 페이지로 이동하게 했다.

이동 후에는 뒤로가기를 막아서 나갔던 방에 다시 접속하는 것을 막았다.

경고 메시지도 추가해봤다.



문제가 있는데 참가자가 들어오면 실시간으로 참가자 현황이 바뀌어야 하지만 바뀌지 않는다. 아마 처음 draw페이지로 들어올 때만 db에서 member를 가져오고, 그 후에는 따로 불러오지 않아서 그런것 같다. 내일 다시 해볼 것.

day14. 230606



/draw페이지 만들기

1. ~~그림판 그림 socket으로 실시간 공유.~~

- 방을 나누는 것은 chatting까지 개발 후 할 것 ⇒ 완료

2. ~~tools 가 기능 개발(그림판과 연계됨)~~

- 초대하기와 방 나가기는 방 나누기 개발 후 할 것 ⇒ 완료

3. ~~방 나누기~~

- start에서 방 번호, 참여자가 설정한 닉네임을 draw와 chat에 넘겨주고 socket에 담아 서버와 통신한다. ⇒ 완료
- 서버는 받은 데이터 중에 방id를 구별해서 해당id를 가진 방에만 그림과 chat을 공유한다. ⇒ 완료

4. ~~chatting 기능 ⇒ socket으로 구현, db에서 방, 유저 정보 가져와야함, 여러명일 수 있기 때문에 상대 구별해야함.~~

- db에서 가져오지 않고 처음 입력 후 페이지 이동시 props로 직접 넘겨주었다. ⇒ 완료
- 상대방을 구별하지만 채팅시 닉네임이 뜨지 않아서 누가 입력한 채팅인지 모르는 문제 수정해야함 ⇒ 수정 완료

5. ~~방 참가자 표시(멤버 이름)~~

- 공간이 없어서 초대하기 버튼을 누르면 모달로 띄우고 모달 하단에 초대링크를 표시해야겠다. ⇒ 완료

6. ~~방 나가기 클릭 시~~

- 진짜 나갈건지 모달로 묻기, 나가면 start로 이동하면서 이전 경로 폐기 or 저장 X ⇒ 완료
- db에서 멤버 삭제 ⇒ 완료

7. 방 중도 입장 시 그리고 있던 그림, 채팅 불러오기

- 채팅은 db에 따로 저장
- 이미지도 그림 데이터를 넘겨 주었던 것처럼 데이터를 db에 저장한 다음 접속하면 불러와야 할 것 같다.



chat컴포넌트 useEffect설명

```
const onAddMessage = useCallback((chatData) => {
  // console.log(chatData);
  setMessages((prevMessages) => [...prevMessages, chatData]);
}, []);

useEffect(() => {
  props.socket.on("chatting", onAddMessage);
  messageEndRef.current.scrollToView({ behavior: "smooth" });
  return () => {
    // 컴포넌트가 언마운트될 때 이벤트 핸들러 정리
    props.socket.off("chatting", onAddMessage);
  };
}, [props.socket, onAddMessage, messages]);
```

`useEffect` 혹은 컴포넌트가 렌더링될 때마다 실행된다.

이 혹은 `props.socket` 과 `onAddMessage` 를 의존성 배열로 갖고 있으므로, “이 값”들이 변경되었을 때에만 혹 내부의 코드가 실행된다. `props.socket.on("chatting", onAddMessage)` 을 호출하여 소켓 이벤트 리스너를 등록하고, `messageEndRef` 를 사용하여 스크롤을 채팅 메시지의 끝으로 이동시킨다.

이후, 컴포넌트가 언마운트될 때 `props.socket.off("chatting", onAddMessage)` 를 호출하여 이벤트 리스너를 제거한다.

Q. 여기서 “이 값”은 `props.socket`과 `onAddMessage`함수의 `return` 값을 말하는건가?

A. “이 값”은 `props.socket` 과 `onAddMessage` 함수의 현재 값(참조)을 의미한다.

`useEffect` 혹은 의존성 배열에 포함된 값들이 변경되었을 때에만 혹 내부의 코드가 실행된다.

`props.socket` 은 컴포넌트의 속성으로 전달되는 소켓 객체이며, `onAddMessage` 는 `useCallback` 혹은 통해 생성된 함수이다.

의존성 배열 `[props.socket, onAddMessage]` 은 이 두 값의 변경을 감지하고, 값이 변경되었을 때에만 `useEffect` 혹 내부의 코드가 실행된다.

이 값들이 변경되었을 때에만 `useEffect` 내부의 코드가 실행되는 이유는, `useEffect` 가 불필요한 실행을 방지하기 위해 의존성 배열을 모니터링하고 변경을 감지하기 때문이다. 만약 의존성 배열이 빈 배열(`[]`)로 설정되어 있다면, `useEffect` 혹은 컴포넌트가 처음 렌더링될 때에만 한 번 실행되고, 그 이후로는 다시 실행되지 않는다. 의존성 배열에 특정 값을 포함시키면 해당 값이 변경될 때마다 `useEffect` 혹의 코드가 실행되게 된다.

따라서, `useEffect` 혹은 의존성 배열 내부의 값이 변경되었을 때에만 내부의 코드를 실행하며,

`props.socket` 과 `onAddMessage` 함수의 리턴 값이 아닌 현재 값(참조)을 의미한다.

근데 해당 참조값들은 거의 변하지 않음 즉, `useEffect`가 거의 동작하지 않음

⇒ `messages`를 의존성 배열에 추가해서 동작하게 만들었다.

참가자가 들어오면 실시간으로 참가자 현황이 바뀌어야 하지만 바뀌지 않는다.

아마 처음 draw 페이지로 들어올 때만 db에서 member를 가져오고, 그 후에는 따로 불러오지 않아서 그런것 같다.

⇒ 해결

제일 처음 넘겨주는 것에는 한계가 있어서 InviteModal을 띄울 때마다 db에서 불러오게 했다.
이제 멤버 버튼을 클릭하면 모달이 뜨면서 db에 있는 최신 참여자 정보를 가져올 수 있다.
!! 서버에서 응답(res.send()) 두 번 보내지 말 것

```
// InviteModal.jsx

const Invite = (props) => {
  let [participants, setParticipants] = useState([]);

  useEffect(() => {
    axios
      .post("http://localhost:8080/participate", {
        body: { roomData: props.roomData },
      })
      .then((response) => {
        console.log(response.data);
        setParticipants([...response.data.member]);
      })
      .catch((err) => {
        console.log(err);
      });
  }, [props.roomData]);
```

```
// server.js

app.post("/participate", (req, res) => {
  resived = req.body.body;
  console.log("line140", resived);
  db.collection("drawroom").findOne(
    { roomid: resived.roomData.roomid },
    (err, result) => {
      if (err) return console.log(err);
      console.log("participate result", result);
      if (result) {
        res.send(result);
      }
    }
  );
});
```

day15. 230607

방 중도 입장 시 다른 유저들이 그리던 그림과, 채팅은 불러오지 않기로 했다.

끝