



WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Paweł Dzedzic

Aplikacja internetowa do współdzielonego rysowania

Opiekun pracy:

dr inż. prof. PRz Mariusz Borkowski

Rzeszów, 2024

1.	Wstęp	3
2.	Przygotowanie projektu	3
3.	Opis projektu.....	4
3.1.	Strona główna	4
3.1.1.	Okno „Create Room”	4
3.1.2.	Okno „Join Room”.....	5
3.2.	Tablica interaktywna.....	7
3.2.1.	Logika tablicy	7
3.2.2.	Logika Rysowania.....	8
3.2.3.	Logika przybliżenia i oddalania	10
3.2.4.	Logika cofania (ctrl+z) i przewijania do przodu (ctrl+y)	11
3.2.5.	Logika wczytania pokoju	12
3.2.6.	Logika zmiany koloru i grubości narzędzi.....	12
4.	Podsumowanie	13

1. Wstęp

Celem niniejszego projektu było stworzenie zaawansowanej aplikacji internetowej do współdzielonego rysowania, wykorzystując nowoczesne technologie webowe, takie jak React. Aplikacja powstawała za pomocą oprogramowania Yarn stosowanego jako narzędzie do menadżera plików i zarządzania pakietami JavaScript. W projekcie wykorzystano również następujące biblioteki:

- Konva - biblioteka JavaScript stworzona do obsługi i manipulacji grafiką 2D w przeglądarce internetowej. Użyto jej głównie do utworzenia tablicy rysującej jak i samej logiki rysowania.
- Socket.io - biblioteka JavaScript, która umożliwia łatwą implementację komunikacji w czasie rzeczywistym między klientem a serwerem za pomocą WebSocketów.
- Router - biblioteka do obsługi routingu w aplikacjach React. Umożliwia tworzenie wielostronicowych aplikacji bez konieczności przeładowania strony, zarządzając nawigacją oraz renderowaniem odpowiednich komponentów w zależności od adresu URL.

2. Przygotowanie projektu

Na samym początku zainstalowano oprogramowanie yarn. Następnie utworzono projekt React z domyślnymi ustawieniami za pomocą oprogramowanie Vite, które pozwala na tworzenie szablonów aplikacji i uruchomienie lokalnego serwera. Poprzez środowisko Yarn zostały zaimplementowane do projektu wybrane biblioteki. Do uruchomienia utworzonego serwera użyto oprogramowania Node. Poniżej znajduje się sposób uruchomienia aplikacji.

- Uruchomienie serwera aplikacji

```
C:\Users\gosc\Documents\Whiteboard_app\backend>node server.js
```

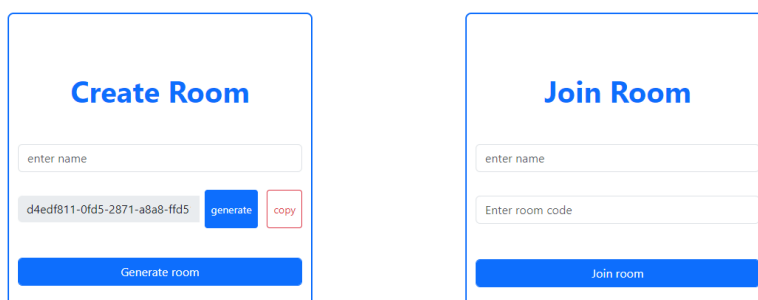
- Uruchomienie aplikacji

```
C:\Users\gosc\Documents\Whiteboard_app>yarn dev
```

3. Opis projektu

3.1. Strona główna

Na pierwszej stronie znajdują się dwa okna formularza. „Create Room” pozwala na utworzenie nowego pokoju do rysowania na podstawie generowanego kodu uuid. „Join room” pozwala na wejście do wybranego pokoju jeśli użytkownik posiada wybrany kod.



3.1.1. Okno „Create Room”

W oknie „Create Room” użytkownik może wygenerować kod za pomocą przycisku „generate” który uruchamia wybraną funkcję:

```
const uuid = () => {  
  let S4 = () => {  
    return (((1 + Math.random()) * 0x10000) | 0).toString(16).substring(1);  
  };  
  return (S4() + S4() + "-" + S4() + "-" + S4() + "-" + S4() + "-" + S4() + S4() + S4());  
};
```

Po wygenerowaniu kodu i wpisaniu nazwy użytkownika i wciśnięciu przycisku „create room” następuje przekierowanie na stronę z możliwością rysowania.

```
const CreateRoomForm = ({ uuid, socket, setUser }) => {
  const [roomId, setRoomId] = useState(uuid());
  const [name, setName] = useState("");

  const navigate = useNavigate();

  const copy = () => {
    navigator.clipboard.writeText(roomId);
  }

  const handleCreateRoom = (e) => {
    e.preventDefault();

    const roomData = {
      name,
      roomId,
      userId: uuid(),
      host: true,
    };
    setUser(roomData);
    navigate(`/${roomId}`);
    socket.emit("userJoined", roomData);
  };
};
```

Powyższy kod uruchamia się właśnie przy wciśnięciu przycisku. Generuje on użytkownika i wysyła na serwer zmienną pod którą znajdują się informacje o nim.

3.1.2. Okno „Join Room”

Proces dołączenia do pokoju działa w podobny sposób jak generowanie. Różni się on jedynie implementacją wyjątku odpowiedzialnego za wykrycie błędu w przypadku dołączenia nie istniejącego pokoju. Za każdym dołączeniem użytkownika na stronę główną, pobierana jest z serwera lista zapamiętująca unikalne identyfikatory pokoi.

```
io.on("connection", (socket) => {
  console.log('A user connected:', socket.id);

  // Lista utworzonych już pokoi wysłana
  socket.emit("createdRooms", createdRooms);
});
```

Poniższy proces znajduje się po stronie klienta i nasłuchuje powyższego sygnału serwera.

```
useEffect(() => {
  socket.on("createdRooms", (createdRooms) => {
    setCreatedRooms(createdRooms);
  });
}, [socket]);
```

Następnie porównany jest wpisany identyfikator w pole tekstowe z tym znajdującym się w zapisanej tablicy. W przypadku błędnej identyfikacji zwracany jest błąd.

```
const handleRoomJoin = (e) => {
  e.preventDefault();

  const roomData = {
    name,
    roomId,
    userId: uuid(),
    host: false,
  };

  if (!createdRooms[roomData.roomId]) {
    setRoomError(true);
    return;
  }

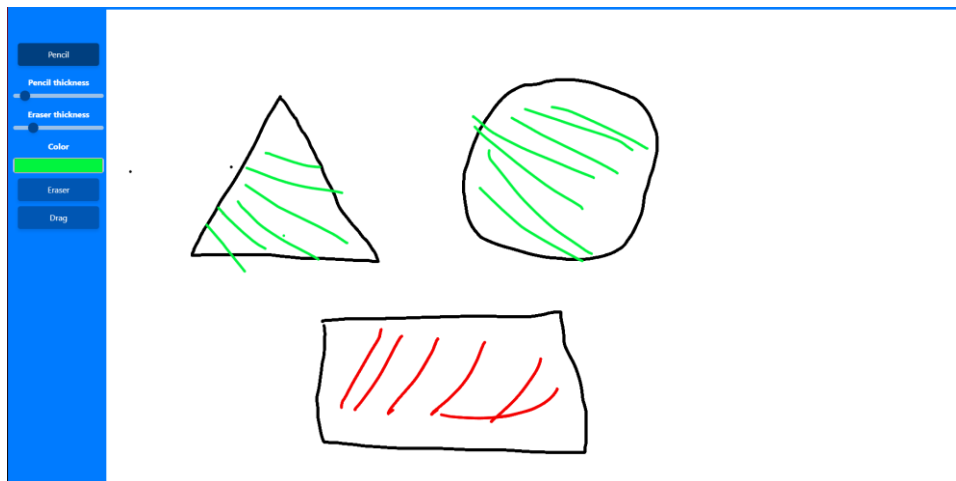
  setUser(roomData);
  navigate(`/${roomId}`);
  socket.emit("userJoined", roomData);
};
```

Join Room

Pokój nie istnieje

3.2. Tablica interaktywna

Utworzona została nieskończona tablica do rysowania, którą można przesuwać, oddalać lub przybliżać. Posiada ona podstawowe funkcje wyboru grubości pędzla i koloru, oraz wyboru grubości gumki.



3.2.1. Logika tablicy

Tablica renderowana jest za pomocą biblioteki Konva. Do obiektu „Stage” odpowiadającym za tablicę przekazywane są metody takie jak pisanie lub przybliżanie i oddalanie.

```
<Stage
  width={width}
  height={height}
  draggable={mode === 'drag'}
  onMouseDown={handleMouseDown}
  onMouseMove={handleMouseMove}
  onMouseUp={handleMouseUp}
  onWheel={handleWheel}
  ref={stageRef}
>
```

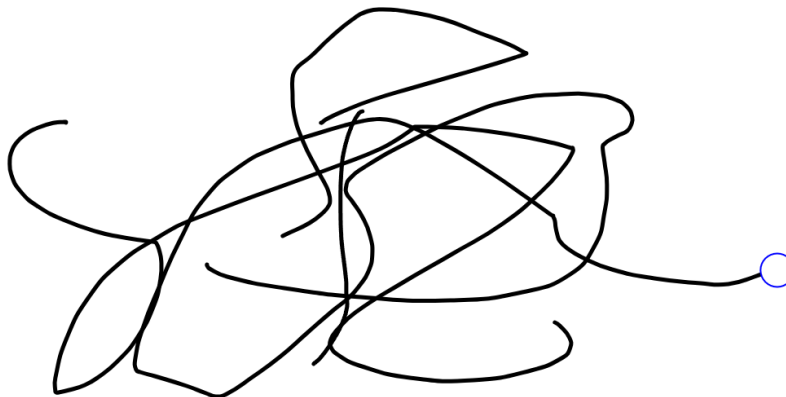
Element „Layer”, grupuje kształty i działa jak warstwa na płótnie. W tym obiekcie mapowane są punkty w celu renderowania linii na płótnie. Linie są reprezentowane przez komponent biblioteki Konva „Line”. Obiekt posiada atrybuty wskazujące na położenie punktów rysujących, kolor i grubość pędzla lub gumki.

```

<Layer>
  {drawingLines.map((line, i) => (
    <Line
      key={i}
      points={line.points}
      stroke={line.tool === "eraser" ? "#FFFFFF" : line.strokeColor}
      strokeWidth={
        line.tool === "eraser" ? line.eraserThickness : line.pencilThickness
      }
      tension={0.5}
      lineCap="round"
      globalCompositeOperation={
        line.tool === "eraser" ? "destination-out" : "source-over"
      }
    />
  ))}
  {isDrawing && mode === "eraser" && (
    <Circle
      x={eraserPosition.x}
      y={eraserPosition.y}
      radius={eraserThickness / 2}
      fill="rgba(255,255,255,0.5)"
      stroke="blue"
      strokeWidth={2}
    />
  )}
)}

```

Poniżej znajduje się obiekt „Circle”, którego celem jest zwizualizowanie obszaru wymazywania przez gumkę elementów, które zostały narysowane.



3.2.2. Logika Rysowania

Przy wyborze wybranej opcji do rysowania zmieniany jest status zmiennej mode.

```

const handleModeChange = (newMode) => {
  setMode(newMode);
};

```

W chwili gdy użytkownik wciśnie lewy przycisk myszy sprawdzany jest typ elementu jaki wybrał. Następnie ustawiany jest stan „isDrawing” co na „true” co wskazuje że użytkownik zaczął rysowanie. Pobierana jest referencja do obiektu sceny, która jest przekazywana do funkcji pobierającej pozycję wskaźnika myszy względem tablicy. Na tej

podstawie tworzona jest nowa linia i atrybutami takimi jak grubość ołówka lub gumki. Po tym aktualizowany jest stan tablicy zawierającej informacje na temat utworzonych linii.

```
const handleMouseDown = (e) => {
  if (mode === "pencil" || mode === "eraser") {
    setIsDrawing(true);
    const stage = e.target.getStage();
    const pos = getRelativePointerPosition(stage);
    const newLine = {
      points: [pos.x, pos.y],
      tool: mode,
      pencilThickness,
      eraserThickness,
      strokeColor,
    };
    setDrawingLines((prevLines) => [...prevLines, newLine]);
  }
};
```

Funkcja „getRelativePointerPosition” pozwala na zidentyfikowanie gdzie znajduje się kursor w trakcie wykonywania operacji rysowania.

```
const getRelativePointerPosition = (node) => {
  const transform = node.getAbsoluteTransform().copy();
  transform.invert();
  const pos = node.getStage().getPointerPosition();
  return transform.point(pos);
};
```

Poniższa funkcja jest wywoływana w trakcie ruchu myszą przy jednoczesnym trzymaniu lewego przycisku myszy. Pobierana jest pozycja kursora a następnie sprawdzany jest rodzaj narzędzia. W przypadku gumki ustawiana jest jej pozycja. W zmiennej „lastLine” zapisywana jest narysowana linia (czyli ta utworzona w funkcji „handleMouseDown”). Do tej zmiennej będącej tablicą dodawane są nowe współrzędne. Ostatni element tablicy „drawingLines” jest zastępowany zaktualizowaną linią. Na koniec aktualizowana jest tablica „drawingLines” o ostatnią zaktualizowaną linię. Po wszystkim ostatnia narysowana linia jest wysyłana na serwer wraz z numerem identyfikacyjnym pokoju, aby zsynchronizować rysunek z innymi użytkownikami w tym samym pokoju.

```
const handleMouseMove = (e) => {
  if (!isDrawing) return;

  const stage = e.target.getStage();
  const pos = getRelativePointerPosition(stage);
  if (mode === "eraser") {
    setEraserPosition(pos);
  }
  let lastLine = drawingLines[drawingLines.length - 1];
  lastLine.points = lastLine.points.concat([pos.x, pos.y]);
  drawingLines.splice(drawingLines.length - 1, 1, lastLine);
  setDrawingLines(drawingLines.concat());

  socket.emit("drawing", { roomId: user.roomId, line: lastLine });
};
```

Po opuszczeniu lewego przycisku myszy pobierana jest ostatnia linia i wysyłana na serwer. Następnie wyłączony zostaje tryb rysowania zmieniając status na false.

```
const handleMouseUp = () => {
  if (isDrawing) {
    const lastLine = drawingLines[drawingLines.length - 1];
    socket.emit("drawing", { roomId: user.roomId, line: lastLine });
  }
  setIsDrawing(false);
};
```

3.2.3. Logika przybliżenia i oddalania

Pierwsza linia poniższego kodu zapobiega przewijaniu strony w przeglądarce. Pobierana jest referencja do sceny, aktualna skala, pozycja kursora, współczynnik skali. Następnie obliczana jest nowa skala na podstawie kierunku przewijania. Obliczane jest względne położenie kursora względem sceny i aktualnej skali. Na tej podstawie ustawiana jest nowa skala dla sceny na obu osiach. Na koniec obliczana jest nowa pozycja sceny, która jest aktualizowana i następuje ponowne rysowanie elementów na tablicy.

```

const handleWheel = (e) => {
  e.evt.preventDefault();
  const stage = stageRef.current;
  const oldScale = stage.scaleX();
  const pointer = stage.getPointerPosition();
  const scaleBy = 1.05;
  const newScale = e.evt.deltaY > 0 ? oldScale / scaleBy : oldScale * scaleBy;

  const mousePointTo = {
    x: (pointer.x - stage.x()) / oldScale,
    y: (pointer.y - stage.y()) / oldScale,
  };

  stage.scale({ x: newScale, y: newScale });

  const newPos = {
    x: pointer.x - mousePointTo.x * newScale,
    y: pointer.y - mousePointTo.y * newScale,
  };

  stage.position(newPos);
  stage.batchDraw();
};

```

3.2.4. Logika cofania (ctrl+z) i przewijania do przodu (ctrl+y)

Gdy użytkownik postanowi skorzystać z cofnięcia lub przewinięcia zmian naniesionych na scenę, informacje o tym jest przesyłana na serwer wraz z numer identyfikacyjnym pokoju przypisanym pod użytkownika.

```

useEffect(() => {
  const handleUndoRedo = (e) => {
    if (e.ctrlKey && e.key === "z") {
      e.preventDefault();
      socket.emit("undo", user.roomId);
    } else if (e.ctrlKey && e.key === "y") {
      e.preventDefault();
      socket.emit("redo", user.roomId);
    }
  };

  window.addEventListener("keydown", handleUndoRedo);
  return () => window.removeEventListener("keydown", handleUndoRedo);
}, [socket, user.roomId]);

```

Po stronie serwera sprawdzana jest lista zawierająca informacje o narysowanych liniach danego pokoju. W razie gdy taki element w liście istnieje to jest on usuwany z od końca, a następnie dodawany do listy zapamiętującej cofniętą linię danego pokoju. Na koniec informacja jest rozsyłana do wszystkich użytkowników pokoju.

```

socket.on("undo", (roomId) => {
  if (roomDrawings[roomId] && roomDrawings[roomId].length > 0) {
    const undoneLine = roomDrawings[roomId].pop();
    if (!undoStack[roomId]) {
      undoStack[roomId] = [];
    }
    undoStack[roomId].push(undoneLine);
    io.to(roomId).emit("drawingHistory", roomDrawings[roomId]);
  }
});

```

W przypadku przewijania do przodu sprawdzana jest lista elementów, które zostały cofnięte z listy „roomDrawings”. Ostatni element tej listy jest następnie usuwany z niej i przeniesiony na koniec listy początkowej z której to został na początku cofnięty. Na koniec informacja jest wysyłana do wszystkich użytkowników pokoju.

```

socket.on("redo", (roomId) => {
  if (undoStack[roomId] && undoStack[roomId].length > 0) {
    const redoneLine = undoStack[roomId].pop();
    roomDrawings[roomId].push(redoneLine);
    io.to(roomId).emit("drawingHistory", roomDrawings[roomId]);
  }
});

```

3.2.5. Logika wczytania pokoju

Gdy nowy użytkownik dołączy do stworzonego wcześniej pokoju to narysowane już wcześniej linie przez poprzednich użytkowników zostają odtworzone na tablicy nowego członka.

```

if (roomDrawings[roomId]) {
  socket.emit("drawingHistory", roomDrawings[roomId]);
}
socket.emit("userIsJoined", { success: true });
});

```

3.2.6. Logika zmiany koloru i grubości narzędzi.



Gdy użytkownik zmieni ustawienia dowolnego narzędzia to jego wartość jest aktualizowana przez poniższą funkcję.

```
const ChangeAttribute = (attribute) => {  
  if (attribute.target.className === "pencil-thickness") {  
    setPencilThickness(attribute.target.value);  
  } else if (attribute.target.className === "eraser-thickness") {  
    setEraserThickness(attribute.target.value);  
  } else if (attribute.target.className === "color") {  
    setStrokeColor(attribute.target.value);  
  }  
};
```

4. Podsumowanie

Projekt stworzenia w pełni funkcjonalnej aplikacji umożliwiającej jednocześnie pisanie w czasie rzeczywistym okazał się wyzwaniem. Kwestią pierwszorzędą w dalszym rozwoju aplikacji jest optymalizacja logiki pisania ze względu na wykorzystanie pamięci przez przechowywanie informacji na temat rysowania linii. Kolejną kwestią jest lepsze zabezpieczenie procesu dołączenia do pokoju i rozwiązanie wszelkich możliwych problemów z nimi występujących. Aktualnie po wyłączeniu serwera, wszystkie utworzone pokoje są usuwane z pamięci, dlatego kluczowym elementem jest zaimplementowanie bazy danych do projektu, która będzie przechowywać informacje na temat utworzonych pokoi.

Jak można zauważyć projekt wymaga sporo pracy nad nowymi funkcjonalnościami i naprawą widocznych problemów. Mimo to udało się utworzyć pewien zarys tego jak ta aplikacja będzie w przyszłości wyglądała. Ponadto podstawowe działanie aplikacji, czyli rysowanie w czasie rzeczywistym działa poprawnie i pozwala na podstawowe wykonywanie zamierzonych celów.