

Uniwersytet Bielsko-Bialski

LABORATORIUM

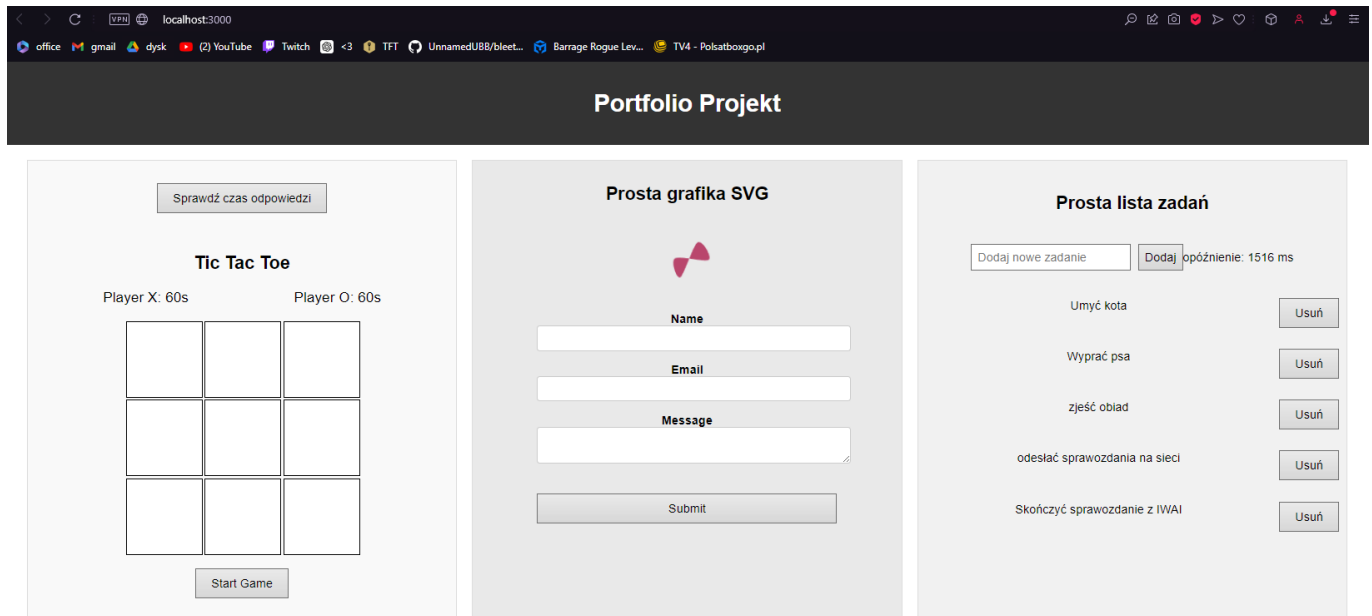
IWAI

Portfolio

GRUPA: 2B / SEMESTR: 6 / ROK: 3

Kacper Lizak / 59443

Jak wygląda :



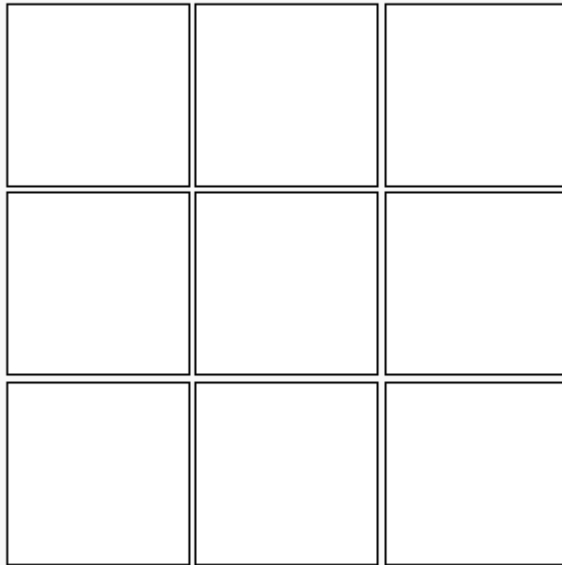
Z czego się składa :

Gra w kółko i krzyżyk :

Tic Tac Toe

Player X: 60s

Player O: 60s



Start Game

Kod dla gry:

```
import React, { useState, useEffect, useRef } from 'react';
import './TicTacToe.css';

function TicTacToe() {
  const initialBoard = Array(9).fill(null);
  const [board, setBoard] = useState(initialBoard);
  const [currentPlayer, setCurrentPlayer] = useState('X');
  const [timeLeft, setTimeLeft] = useState({ X: 60, O: 60 });
  const [gameOver, setGameOver] = useState(false);
  const [gameStarted, setGameStarted] = useState(false);
  const intervalRef = useRef();

  useEffect(() => {
    if (gameStarted && !gameOver) {
      intervalRef.current = setInterval(() => {
        setTimeLeft((prev) => {
          const newTime = { ...prev, [currentPlayer]: prev[currentPlayer] - 1 };
          if (newTime[currentPlayer] <= 0) {
            clearInterval(intervalRef.current);
            setGameOver(true);
            alert(`Player ${currentPlayer} ran out of time!`);
          }
          return newTime;
        });
      }, 1000);
    }
  });
}
```

```
    }, 1000));
  }

  return () => clearInterval(intervalRef.current);
}, [currentPlayer, gameOver, gameStarted]);

const handleClick = (index) => {
  if (gameStarted && !gameOver && !board[index]) {
    const newBoard = board.slice();
    newBoard[index] = currentPlayer;
    setBoard(newBoard);

    if (checkWinner(newBoard)) {
      setGameOver(true);
      clearInterval(intervalRef.current);
      alert(`Player ${currentPlayer} wins!`);
    } else if (newBoard.every((cell) => cell)) {
      setGameOver(true);
      clearInterval(intervalRef.current);
      alert('Draw!');
    } else {
      setCurrentPlayer(currentPlayer === 'X' ? 'O' : 'X');
    }
  }
};

const checkWinner = (board) => {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (board[a] && board[a] === board[b] && board[a] === board[c]) {
      return true;
    }
  }
  return false;
};

const startGame = () => {
  setGameStarted(true);
};

const restartGame = () => {
  setBoard(initialBoard);
  setCurrentPlayer('X');
  setTimeLeft({ X: 60, O: 60 });
};
```

```
    setGameOver(false);
    setGameStarted(false);
  };

  return (
    <div className="tic-tac-toe">
      <h2>Tic Tac Toe</h2>
      <div className="timers">
        <div>Player X: {timeLeft.X}s</div>
        <div>Player O: {timeLeft.O}s</div>
      </div>
      <div className="board">
        {board.map((cell, index) => (
          <div key={index} className="cell" onClick={() => handleClick(index)}>
            {cell}
          </div>
        ))}
      </div>
      {!gameStarted && !gameOver && (
        <button onClick={startGame}>Start Game</button>
      )}
      {gameOver && (
        <button onClick={restartGame}>
          Restart Game
        </button>
      )}
    </div>
  );
}

export default TicTacToe;
```

Co posiada ten komponent:

-Liczniki czasu dla graczy co pozwala na cięższą rozgrywkę, wartość czasu można bardzo łatwo zmienić w kodzie.

Przykład użycia:

Tic Tac Toe

Player X: 57s

Player O: 53s

X		X
X	O	
O		O

Komunikat ze strony localhost:3000

Player X wins!

OK

Tic Tac Toe

Player X: 57s

Player O: 53s

X	X	X
X	O	
O		O

Restart Game

Prosta grafika SVG wygenerowana przez stronę do tworzenia takich grafik :

Prosta grafika SVG



Kod grafiki :

```
<svg
  className="rotating-svg"
  xmlns="http://www.w3.org/2000/svg"
  version="1.1"
  xmlnsXlink="http://www.w3.org/1999/xlink"
  xmlnsSvgjs="http://svgjs.dev/svgjs"
  viewBox="0 0 600 600"
  width="100"
  height="100"
>
  <path
    d="M183.76963806152355
      208.9005064639751C161.6492156982423
      170.41882972650765 305.4973805745444
      93.97905108067431 325.916229248047
      124.86910387608447C346.3350779215496
      155.75915667149462 284.1623077392579
      355.75914649896856 306.2827301025392
      394.24082323643603C328.4031524658204
      432.7224999739035 479.0576121012371
      386.6492170962993 458.6387634277345
      355.75916430088915C438.21991475423187
      324.869111505479 205.8900604248048
      247.3821832014425 183.76963806152355
      208.9005064639751C161.6492156982423
      170.41882972650765 305.4973805745444
      93.97905108067431 325.916229248047
      124.86910387608447"
    fill="hsl(340, 45%, 50%)"
  ></path>
</svg>
```

Grafika ma również css dzięki któremu kręci się w koło dopóki na nią nie najedziemy :

```
#svg-container {
  text-align: center;
  margin: 20px;
}

.rotating-svg {
  transition: transform 0.5s ease-in-out;
  animation: rotate 5s linear infinite;
}

.rotating-svg:hover {
  animation-play-state: paused;
}

@keyframes rotate {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}
```

Mamy również pseudo formularz kontaktowy z walidacją:

Name

Email

Message

Submit

Name

Name is required

Email

Message

Message is required

Email

! Uwzględnij znak „@” w adresie e-mail. W adresie „kacp” brakuje znaku „@”.

Przykładowy kod walidatora:

```
const validateForm = () => {  
  let formErrors = {};  
  if (!name) formErrors.name = 'Name is required';  
  if (!email) {  
    formErrors.email = 'Email is required';  
  } else if (!/\S+@\S+\.\S+/.test(email)) {  
    formErrors.email = 'Email address is invalid';  
  }  
  if (!message) formErrors.message = 'Message is required';  
  return formErrors;  
};
```

Kolejnym elementem jest ToDoList:


Prosta lista zadań

opóźnienie: 1516 ms

Umyć kota	<input type="button" value="Usuń"/>
Wyprać psa	<input type="button" value="Usuń"/>
zjeść obiad	<input type="button" value="Usuń"/>
odesłać sprawozdania na sieci	<input type="button" value="Usuń"/>
Skończyć sprawozdanie z IWA	<input type="button" value="Usuń"/>

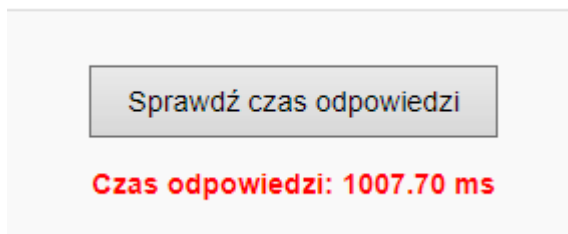
ToDo posiada serwer który jest odpowiedzialny za zapisywanie oraz odczytywanie zadań, jest również funkcja która liczy opóźnienie.

Przycisk wchodzi w "bufforowanie" na czas otrzymania odpowiedzi od serwera :

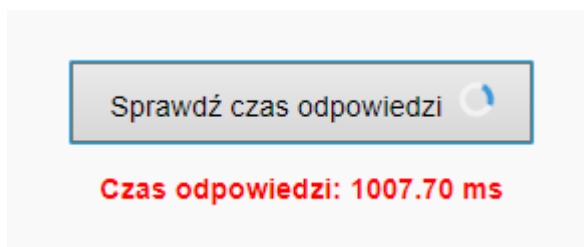
 opóźnienie: 1516 ms

Czas zostaje sztucznie przedłużony poprzez dodanie 1000ms żeby można było wyłapać bufforowanie, od czasu który jest wyświetlany należy odjąć więc 1000ms co w tym przypadku daje nam 16 ms od wysłania requesta do otrzymania responsa od serwera.

Ostatnim elementem jest pseudo sprawdzanie czasu odpowiedzi :



Tutaj dokładnie jak w przykładzie wyżej czas został sztucznie dopakowany aby umożliwić zobaczenie wczytywania się :



Kod do całego projektu znajduje się tutaj :

<https://github.com/KLizakk/PortfolioIWA>