Tic-Tac-Toe Step 7: Scoreboard

# Introduction:

This exercise will be the **last part** of this React course. We will wrap it up with a small but useful addition to our Tic-Tac-Toe, a **scoreboard** that keeps a track of the games won by both players (**X** & **O**).

# Scoreboard.jsx:

Start with creating a file called **Scoreboard.jsx**, make a base for the component and import React. Give the div element a className, “**scoreboard**”:

import React from "react";

export const Scoreboard = () => {

    return(

        <div className="scoreboard">

        </div>

    )

}

Let’s move to our base component and **import** the Scoreboard component:

import { Scoreboard } from "./Scoreboard";

The number of wins will be stored in a state, so we will create our one last state, named **scores**. Since the scoreboard will track the scores of both players, for the first time, our state will have **two variables**, called **xScore** and **oScore**. The **initial state of b**oth will obviously be **0**:

const [scores, setScores] = useState({ xScore: 0, oScore: 0})

# useEffect:

We will create a new useEffect hook that runs **every time** the status state is **updated**. Inside we will place the code that is responsible for counting the wins of the scoreboard. We also need to store the winner in a variable like in the previous useEffect:

useEffect(() => {

const winner = checkWinner();

}, [status])

We want the calculation to only happen, **when there is a winner**. We will add a simple if statement to check if the winner variable is **null**, if it is the function will **stop**:

useEffect(() => {

    const winner = checkWinner();

    if(winner === null){

        return;

    }

}, [status])

If it isn’t (**meaning one of the players has won the game**) the function will **continue** to the next if statement that we will add now.

Start with an **if statement** that checks if the winner is equal to **X**:

if (winner === "X"){

}

Inside, we will add one to the xScore every time the game is won **by X**:

setScores({ xScore: scores.xScore + 1, oScore: scores.oScore});

---------------------------

Because we have **two variables** inside our scores state, we will set the state we want to update to add one to it. The oScore will stay the same here, since it’s **not possible** to have two winners in Tic-Tac-Toe.

There are only **two options** for the winner, not including a draw, because the function has continued through the if statement that checks if the winner returns null, **meaning either X or O has won**. We can use **else** rather than make another condition. If the game isn’t won by **X** that automatically means **O** won. In that case, we will set the score state to add one to the oScore:

if (winner === "X") {

    setScores({ xScore: scores.xScore + 1, oScore: scores.oScore});

} else {

setScores({ xScore: scores.xScore, oScore: scores.oScore + 1 });

} --------------------------

# Scoreboard component:

Let’s render the scoreboard now, even though nothing is visible yet. It’s just easier to do at this time because we will have to pass the scores state to it. Place the component element below the game title, and above the board component:

<h1>Tic-Tac-Toe</h1>

------------------------------

<Scoreboard scores={scores} />

------------------------------

<Board gameState={gameState} onSquareClick={onSquareClick}/>

Now, move to the Scoreboard.jsx. Pass the scores state to the component as an argument, since its a state we will need to add **squirrely brackets** around it:

export const Scoreboard = ({ scores }) => {

    return(

        <div className="scoreboard">

        </div>

    )

}

Next, add two span elements inside the components div element and render the scores there, from our scores state that we passed to the component, like shown below:

export const Scoreboard = ({ scores }) => {

    return(

        <div className="scoreboard">

-------------------------------

            <span>X - {scores.xScore}</span>

            <span>O - {scores.oScore}</span>

-------------------------------

        </div>

    )

}

Now the scoreboard should be **visible**, open your browser and see if the scores are updating right:

Calendar

Description automatically generated

# clearScoreBoard:

Now we have an easy way to track the wins, but what if we want to **reset** the scoreboard? Right now, the only way is to **refresh the whole page**.

So, for the last thing we’ll add to our game, is a **button to set the scores to 0**. This will be one easy function inside our base component. Let’s go ahead and create a new function named **clearScoreboard**:

function clearScoreBoard(){

}

Here we will simply set the scores states variables **to 0**:

setScores({ xScore: 0, oScore: 0 })

Let’s also make the board and turn reset with the button, so if the scoreboard is cleared the **game resets automatically**:

function clearScoreboard(){

    setScores({ xScore:0, oScore:0 })

    setGameState(intialBoard)

    setIsXturn(true)

}

All that’s left to do is adding a button and assigning the function. Let’s place that directly **under** our scoreboard:

<Scoreboard scores={scores} />

<button onClick={clearScoreboard}>Clear scoreboard</button>

The button should be visible and functioning now, when the scoreboard is cleared the game will reset:

A picture containing calendar

Description automatically generatedTable

Description automatically generated

# Conclusion:

We have now reached the end of the game, and the course as a whole.

The goal of the course was to get to know the basics of React. To understand components and how they work, to understand the reusability of them, and understand the different uses they can have. We also got to know the useState and useEffect hooks, and the different ways to use them.

**Hopefully you have got the hang of it, and maybe gained the knowledge to start building your own applications and bigger projects with the great language!**