Voting App - ReactJS

V1 December 2022

CONTENTS

[What you’ll be creating 3](#_Toc121997232)

[Create your React app using VSCode 4](#_Toc121997233)

[Create your first component 5](#_Toc121997234)

[Create the data 6](#_Toc121997235)

[Logic to build the webpage 7](#_Toc121997236)

[Add images 9](#_Toc121997237)

[Create a return for App.js 10](#_Toc121997238)

[Adding and styling a header 12](#_Toc121997239)

What you’ll be creating

In this lab, you will create a simple React app that allows the user to vote for their favourite languages (e.g. JavaScript or Python).

You will add some interactivity. When the “+1” buttons are pressed, the number of votes for the corresponding language will increase by 1.

Table

Description automatically generated

Create your React app using VSCode

Before you begin, please ensure you have VSCode and [node](https://nodejs.org/en/download/) installed!

1. Create a new folder and open it in VSCode.
2. Click ‘Terminal’ > ‘New Terminal’
3. When the terminal opens at the bottom, type the following command:

npx create-react-app voteapp

1. Focus on your new folder with the following command:

cd voteapp

1. run **npm start** in the terminal. You should see this window

Graphical user interface

Description automatically generated with medium confidence

Create your first component

1. Inside your **src** folder, create a new folder called **components**
2. Inside the **components** folder, create a file called **VotingCard.jsx**
3. We will create a card component that can be reused for each new language that the users can vote for, and will later call it in our React app.

Ensure you have the following code in the **VotingCard.jsx** file

import React from "react";

function VotingCard(props) {

    const { lang, incrementVoteCount } = props;

    return(

        <div className="card">

            <div className="logo">

                <img src={`/assets/images/${lang.logo}`} alt="" />

            </div>

            <div className="content">

                <p>Name: {lang.name}</p>

                <p>Rating: {lang.rating}</p>

            </div>

            <button onClick={() => incrementVoteCount(lang.\_id)}>

                +1

            </button>

            <p>Vote Count: {lang.votes}</p>

        </div>

    );

}

export default VotingCard;

This code builds the JSX to create a block of HTML based on the information we will provide in a JSON file. You can see that the process is based on the destructuring of the props passed to the VotingCard function (see {lang.name} and {lang.logo}).

It also includes a button for each language that calls a function to increment a number (representing the number of votes) that will be created as state when App.js is called. This allows us to update the number of votes live on the website. Under the hood, React uses the virtual DOM when updating the UI.

The className attributes you can see will be used to style the webpage later.

Create the data

1. Create a folder called **lib** inside your **src** folder
2. Create a new file called **lang.json**  inside the **lib** folder. Populate it with the following content (don’t worry about the image paths just yet).

[

    {

        "\_id": 1,

        "name": "Django",

        "logo": "django.png",

        "rating": "3rd",

        "votes":0

    },

    {

        "\_id": 2,

        "name": "JavaScript",

        "logo": "js.png",

        "rating": "8th",

        "votes":0

    },

    {

        "\_id": 3,

        "name": "Python",

        "logo": "python.png",

        "rating": "1st",

        "votes":0

    }

]

This code will form the content of our webpage, with each entry being used on its own card. It’s important to link here that {lang.name} which you saw in the VotingCard.jsx file corresponds to the “name” elements in the above JSON file. They will all be used and deconstructed when the website renders.

Logic to build the webpage

1. Find your **App.js** file and delete the contents.
2. Include the code at the top to set up the file for importing your component and data

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

import VotingCard from "./components/VotingCard";

import langsJson from "./lib/lang.json";

We have imported useEffect and useState – these will later allow us to control when the webpage re-renders and to be able to create state to keep track of the number of votes for each language.

The “import VotingCard” line allows us to call that component into the main app when rendered.

When we imported langsJson, it’s important to remember that we need to use that name to reference in the code. You can imagine that we imported that file **as** langsJson (with that specific name).

1. We need to create our state (variables if you like) at the top of our code. In the body of function App(), you need to include:

const [langs, setLangs] = useState([]);

This allows us to store the languages in state so that they can be updated.

1. In the body of the App component, create a new function which allows the number of votes to be incremented when the user click on the add button.

function handleIncrementVoteCount(langId) {

    const updatedLangs = langs.map((lang) => {

      if (lang.\_id === langId) {

        lang.votes = lang.votes + 1;

      }

      return lang

    });

    setLangs(updatedLangs)

  }

This function takes one argument, “langId” – this is the \_id property of the card that was clicked. Using the Array map() method, the function iterates through each language (originally from the JSON file but now held in state), and on each iteration checks if the language’s ID matches the ID of the language that was voted for. If the IDs match, then the number of votes for that language will be incremented.

The state value “langs” is then updated via “setLangs”. React will notice that the state has been updated, and the UI will update through a process of diffing and reconciliation.

Add images

1. In your **public** folder, create an **assets** folder.
2. Create an **images** folder inside your **assets** folder
3. Find logos online (small if possible) of each of your languages. If you look back at the JSON file, you can see we had **django.png**, **js.png** and **python.png.** These need to be put inside your **images** folder.

You will notice in the <img> tag in **VotingCard.jsx** we typed the following:

<img src={`/assets/images/${lang.logo}`} />

Which mirrors the path we just constructed.

Create a return for App.js

We need to create a return for the main App.js file. In this return, we specify how to lay out the content when the app is rendered

1. Under your handleIncrementVoteCount() function, include the following code in the return() statement:

 return (

    <div>

      {langs.map((lang) => {

        return (

          <VotingCard

lang={lang}

incrementVoteCount={handleIncrementVoteCount}

key={lang.\_id}

          />

        )

      })}

    </div>

  );

}

export default App;

This code creates a new <VotingCard> component for each entry in the JSON file and passes the language data into the component via props. By also passing the incrementVoteCount function as a prop, it allows for the passing back up of information. This allows for the button to update the state in the parent (App) component, recording a vote and thus updating the page.

The key={lang.\_id} prop ensures that each child of the map can be uniquely identified by React.

Save your changes.

This gives us each entry set out on our page like so:

A picture containing graphical user interface

Description automatically generatedLogo

Description automatically generatedGraphical user interface, text, chat or text message

Description automatically generated

Which isn’t optimal for viewing. So, let’s add some CSS to the **index.css** file.

\* {

  box-sizing: border-box;

}

img {

  width: 60px;

}

.card {

  float: left;

  border: 1px black solid;

  width: 33%;

  text-align: center;

  padding-top: 10px;

  margin: 1.1px;

}

button {

  border-color: green;

  border-radius: 4px;

  background-color: white;

  color: green;

  height: 30px;

  width: 50px;

}

Saving and reloading the webpage gives us this:

Graphical user interface, application

Description automatically generated

Much better!

Adding and styling a header

1. In your **components** folder, create a new file called **Header.jsx**
2. Include the following code in your **Header.jsx** file

import React from "react";

function Header() {

    return (

        <header className="head">

            <p className="head-title">Language Voting App</p>

        </header>

    );

}

export default Header;

1. Back in **App.js** import your new component under the original imports

import Header from "./components/Header";

1. Call the Header component into your **App.js**. We will need to wrap everything inside a <div>, as React components are only valid if they return a single containing element.

return (

  <div>

    <Header />

    <div>

      {langs.map((lang) => {

        return (

          <VotingCard

lang={lang}

          incrementVoteCount={handleIncrementVoteCount}

key={lang.\_id}

          />

        )

      })}

    </div>

  </div>

  );

}

1. Finally, some CSS to style the content, to be added in **index.css**.

.head-title {

    font-size: 40pt;

    padding-left: 50px;

}

.head {

  font-family: Verdana, Geneva, Tahoma, sans-serif;

  height: 200px;

  background-color: lightgrey;

  padding-top: 50px;

}

Save your changes.

Table

Description automatically generated

End of React Lab