REACT QUICK LABS AND CHALLENGES

QAREACTJS

Learn. To Change.

Contents

1. Quick Lab 1 – Installing and running REACT

1.1. Objectives

- To be able to install, run and manipulate react in VSCode.

1.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and **cd** into the folder.
3. Run the following code to install react.

```
npx create-vite@latest solution
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd solution
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser.
6. Inside your App.jsx file, remove the content between the **App** div and replace it with **<h1>**Hello World**</h1>**.

App.jsx

```
import './App.css';

function App() {
  return (
    <div className="App">
      <h1>Hello World</h1>
    </div>
  );
} export default App;
```

7. Check your browser output once again. It should now simply be the words 'Hello World' cantered in your webpage.

This is the end of Quick Lab 1

2. Quick Lab 2 – Create your first react components

2.1. Objectives

- To understand how to build a page using components.

2.2. Activity

1. In your **src** folder, create a new file called **FirstComponent.jsx**.
2. You need to create a stateless function in this file called **FirstComponent**.
3. This function needs to return a **h1** with the words 'My first Webpage' followed by a **p** tag with the content 'This will look Great'.
4. As all JSX requires there to be a containing tag, ensure it is all within a div with a className of 'content'.

```
const FirstComponent = () => {
    return (
        <div className="content">
            <h1>My First Webpage</h1>
            <p>This will look Great</p>
        </div>
     );
}
 export default FirstComponent;
```

5. Import your **FirstComponent** into **App.jsx** with the following line of code at the top of the file.

```
import FirstComponent from './FirstComponent';
```

6. Replace the **<h1>** tag with the function call **<FirstComponent />** inside the App div.
7. Save your files and check your webpage. It should have the content from your FirstComponent.jsx file rendered inside it.

App.jsx

```
import FirstComponent from './FirstComponent;
import './App.css';

function App() {
  return (
    <div className="App">
      < FirstComponent />
    </div>
  );
}
export default App;
```

8. Create a second JSX file in your **src** folder. Call this one **Image.jsx** .
9. Create a new stateless functional component, as before and name it Image.
10. Find an image of the react logo online and drop the source path into an **<img>** tag (don't forget the containing div). Set the **width** of the image to "200px".
11. Once saved, import it at the top of **App.jsx** and then call the Image import below the previous FirstComponent tag.
12. Save and check the output. The image should be at the bottom of the webpage.

Image.jsx

```
const Image = () => {
    return (
        <div className="image">
            <img src="YourURl.png"
                width="200px"
            />
        </div>
     );
}

export default Image;
```

App.jsx

```
import FirstComponent from './FirstComponent;
import Image from './Image';
import './App.css';

function App() {
  return (
    <div className="App">
      <FirstComponent />
      <Image />
    </div>
  );
}

export default App;
```

This is the end of Quick Lab 2

3. Quick Lab 3 – Props in react

3.1. Objectives

- To understand how to pass information to child components as props.

3.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and **cd** into the folder.
3. Run the following code to install react.

```
npx create-vite@latest props
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd props
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser.
6. Create a new **component** folder inside the **src** folder.
7. Inside that folder, create a new file called **Card.jsx**.
8. This file should contain a **function** which takes in a **parameter** called { name }.
9. The function should return a **h1** tag that outputs "This entry was from { name }".
10. In your **App.jsx** file, ensure that the default react content has been removed, **import Card from './components/Card'** and call the card in the **App** div .
11. Inside the call, add the parameter name="Dave".
12. Save and render and you should see: This Entry was from Dave" on your web browser.
13. Add 2 more **Card** calls with 2 more names and save your files – check the output.
14. Go back to **Card.jsx** and add **age** and **role** to the parameters accepted in the call. It should look like this – { name, age, role }.
15. Add them as outputs to the return.
16. In **App.jsx** add the new parameters to the calls (all inline) and save your files.
17. Go to **App.css,** delete all but the **.App {}** rule at the top and add some styles to improve the look of your cards (You can find some example styles at the bottom of this section).
18. Finally, add an image to the top of your **Card** return and ensure you have included a CSS rule for it for correct alignment.

3.3. If You Have Time

If finished, have a go at simply importing props in the call and then destructure them using dot(.) notation e.g. `props.name`.

Card.jsx

```
const Card = ( {name, age, role} ) => {
    return (
        <div className="card">

            <img src="YourImageURL"
             width="100px"
             />
            <h1>{name}</h1>
            <h2>{age}</h2>
            <h2>{role}</h2>

         </div>
     );
}
export default Card;
```

App.jsx

```
import Card from './components/Card';
import './App.css';

function App() {
  return (
    <div className="App">
      <p><Card name="Bill" age="35" role="DevOps" /></p>
      <p><Card name="Andy" age="27" role="Sales"/></p>
      <p><Card name="Dave" age="64" role="Catering"/></p>
    </div>
  );
}
export default App;
```

Instructions continue on the next page…

App.css

```css
.App {
  text-align: center;
}

.card {
  padding: 5px;
  border-bottom: 1px lightgray solid;
  max-width: 600px;
  margin: auto;
  text-align: left;

}
img {
  float: right;
  padding: 40px;
}
```

This is the end of Quick Lab 3

4. Quick Lab 4 – Conditional Rendering

Note: This lab is a continuation of Lab 3 so make sure you have your solution's folder open in Visual Studio Code.

4.1. Objectives

- To investigate how to control what is rendered based on the condition of the data.

4.2. Activity

1. In your props folder, find your **Card.jsx** file.
2. Add the parameter **isActive** to the call after **role**.
3. Back in the **App.jsx** file, find the Card calls and include a parameter **isActive** as a boolean { true } or { false }.
4. In **Card.jsx**, use conditional rendering to add a rosette icon (Recommend searching for Unicode symbols (https://symbl.cc/en/)) to appear next to the name of the employee if they are active.

Card.jsx

```
const Card = ({name, age, role, isActive}) => {
        return (
                <div className="card">
                    <img src="YourImageURL"
                    width="100px"
                    alt=""
                    />
                    <h1>{name} {isActive && '🏵'}</h1>
                    <h2>{age}</h2>
                    <h2>{role}</h2>
                </div>
            );
}
export default Card;
```

App.jsx

```
import Card from './components/Card';
import './App.css';

function App() {
  return (
    <div className="App">
```

```
      <p><Card name="Bill" age="35" role="DevOps" isActive={true} /></p>
      <p><Card name="Andy" age="27" role="Sales" isActive={false}/></p>
    </div>
  );
}
export default App;
```

This is the end of Quick Lab 4

5. Quick Lab 5 – Rendering External Data

5.1. Objectives

- To import data from a JSON file and render it on a webpage.

5.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest external-data
```

   Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd external-data
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser.
6. Remove the content from **App.jsx** as before, ready for your components.
7. In the **src** folder, create a new file called **itemsData.json**. This is the data you will import into your webpage and display.

itemsData.json

```
[
    {
        "id": 1,
        "symbol": "1",
        "name": "Digit One",
        "unicodeVal": "U+0031"
},
    {
        "id": 2,
        "symbol": "2",
        "name": "Digit Two",
        "unicodeVal": "U+0032"
}
]
```

8. In the **src** folder, create a new folder called **components**.

9. Inside the components folder, create a new file called **itemCard.jsx**.
10. This file will contain the code that receives and standardises the entries in your JSON file.
11. Create a function in this file called **ItemCard.** it should accept the parameters **{symbol, name, unicodeVal}**.
12. In the return of this function, create a **div** with a className of **item-card**.
13. Inside that **div** create a **div** with a **className** of **symbol.** Rest the **{ symbol }** variable inside this **div**.
14. Underneath this **div**, create a **h3** with the **{ name }** variable.
15. Underneath this h3, create a **p** tag with **{ unicodeVal }** inside it.

itemCard.jsx

```
function ItemCard({ symbol, name, unicodeVal}) {
    return (
        <div className="item-card">
            <div className="symbol">{symbol}</div>
            <h3>{name}</h3>
            <p>{unicodeVal}</p>
        </div>
    )
}
export default ItemCard;
```

16. Back inside **App.jsx** import **ItemCard** from the **itemCard.jsx** component.
17. Import **itemsData** from the **itemsData.json** file.
18. In the App, create a **<main>** tag to hold everything inside.
19. Add an **h1** tag which says Unicode Characters.
20. Create a **div** with the **className** 'items-grid'.
21. Inside that **div**, create a **map** from the **itemsData** import. It should perform a callback function and with each item, should perform a call of **ItemCard** with the following parameters being passed as props:

```
key={ item.key }
symbol={ item.symbol }
name={ item.name }
unicodeVal={ item.unicodeVal }
```

22. Test the code to ensure that it is pulling the information out of the file and into the webpage.
23. Open up your **App.css** file and drop the following rules into it to style to content of your webpage.

App.css

```
.App {
  text-align: center;
}

h1 {
  margin-bottom: 1.5rem;
}

.items-grid {
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;
}

.item-card {
  margin-left: 10px;
  width: 200px;
  padding: 1rem;
  background-color: rgb(248, 248, 248);
  border-radius: 3px;
  border: solid 1px rgb(220, 220, 220);
}

.item-card .symbol {
  font-size: 2rem;
  text-align: center;
}

.item-card h3 {
  margin-bottom: 0.5rem;
}

.item-card p {
  margin-left: 0;
}
```

24. Add another couple of records to your Json file to check that the new records render as the others do.

App.jsx

```
import ItemCard from './components/itemCard';
import itemsData from './itemsData.json';
import './App.css';

function App() {

  return (
    <div className="App">
      <main>
        <h1>Unicode Characters</h1>
        <div className="items-grid">
          {itemsData.map((item) => (
            <ItemCard
            key={item.id}
            symbol={item.symbol}
            name={item.name}
            unicodeVal={item.unicodeVal}
            />
          ))}
        </div>
      </main>
    </div>

  );
}

export default App;
```

This is the end of Quick Lab 5

6. Quick Lab 6 – Routing in react

6.1. Objectives

- To create a simple "Single Page Application" (SPA) using the tools for routing in react.

6.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest routertest
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd routertest
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser.
6. You need to use npm to install the routing dependencies that allow for the creation of an SPA. In the terminal, ensure that your project development server is no longer running by pressing ctrl + c. Press 'y' and then enter to confirm.
7. You will need to type the following commands into the terminal, ensuring that you are focused on the folder that holds the react project (for learning purposes the second command uses an alternative "reduced syntax").

```
npm install --save react-router
```

```
npm i -S react-router-dom
```

8. When both of these have finished, open your **package.json** file and check the dependencies. You should now see both of the new packages are installed. These are the packages called when we run **npm -i** after downloading and preparing a new react project.
9. Remove the content from **App.jsx** as before, ready for your components.
10. Create two new folders in the **src** folder, one called **components,** the other called **pages**.
11. We will create the pages first. 2 very simple pages for use in our SPA. Inside the pages folder, create a new file called **HomePage.jsx**.
12. Inside this file you will need to create a new function called **HomePage**, which returns a surrounding div. This div should contain a **h2** tag which says 'This is the home page', and under that, a **p** tag which outputs 'Welcome to this website'.
13. Don't forget to export the function.

14. Create a second file in the **pages** folder, this time called **AboutPage.jsx.** This file will be similar to the home page. Ensure you create an exported function called **AboutPage** which has suitable **h2** and **p** tags for the page.
15. Inside the **components** folder, create a new file called **Navbar.jsx**. This file will contain the links on our SPA to the 2 page components.
16. At the top, write a import for **{ Link }** from **'react-router-dom'**.
17. Create a function called **Navbar**. This function should return the following:
    - Enclosing **<nav>** tags
    - **h1** tag which contains the text 'React App'
    - an unordered list with 2 **li** elements.
    - The first li will contain a Link to "/"
        - <Link to="/">Home</Link>
    - The second will contain a Link to "/about"
        - <Link to="/about">About</Link>
18. You need to prepare your **App.jsx** file to allow for effective routing. At the top of the page you will need to import **{ BrowserRouter, Routes, Route }** from **'react-router-dom'**.
19. You will also need to import the **Navbar**, **HomePage** and **AboutPage** components from their respective files.
20. The return for the App will need to contain an enclosing **<BrowserRouter>** tag. This will immediately be followed by the call for the **Navbar** component.
21. You can now start to register your routes. Create <**main>** and **<Routes>** opening and closing tags. The tags for the individual routes will be nested inside these.
22. You will create a separate **<Route>** tag for each component you wish to call. The following code will create a live view of the **HomePage** component:

```
<Route path="/" element={ <HomePage /> } />
```

You will remember that we put a <Link to="/"> tag in the Navbar component. This route tag links that button to the **HomePage** component.
The same will be done for the **AboutPage** component.

```
<Route path="/about" element={ <AboutPage /> } />
```

23. Test your webpage for working links.
24. Finally, we need to put some styling in the **App.css** file to ensure the product looks a little more usable. All completed files have been pasted below, including the CSS styling – however, you should consider creating your own styles at this point.

Navbar.jsx

```
import { Link } from 'react-router-dom'
export default function Navbar() {
    return (
        <nav>
            <h1>React App</h1>
            <ul>
                <li>
                    <Link to="/">Home</Link>
                </li>
                <li>
                    <Link to="/about">About</Link>
                </li>
            </ul>
        </nav>
    )
}
```

AboutPage.jsx

```
export default function AboutPage() {
    return (
        <div>
            <h2>This is the about Page</h2>
            <p>Some helpful information</p>
        </div>
    )
}
```

HomePage.jsx

```
export default function HomePage() {
    return (
        <div>
            <h2>This is the Home Page</h2>
            <p>Welcome to the Website</p>
        </div>
    )
}
```

App.jsx

```
import {BrowserRouter, Routes, Route} from 'react-router-dom'
import Navbar from './components/Navbar';
import HomePage from './pages/HomePage';
import AboutPage from './pages/AboutPage';
import './App.css';

function App() {
  return (
  <BrowserRouter>
    <Navbar />
      <main>
            <Routes>
                  <Route path="/" element={<HomePage />} />
                  <Route path="/about" element={<AboutPage />} />
            </Routes>
      </main>
  </BrowserRouter>
  );
}
export default App;
```

App.css

```
nav {
  box-shadow: rgba(0, 0, 0, 0.08) 0 5px 8px;
}

nav h1 {
  text-align: center;
  margin: 0;
  padding: 1rem;
  background-color: #0b2025;
  color: #61dbfb;
}

nav ul {
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 1rem;
  margin: 0;
  padding: 0.5rem 0;
  list-style: none;
}

nav p,
a {
  margin: 0;
  color: #444444;
  text-decoration: none;
}

nav a:hover {
  text-decoration: underline;
}

main {
  margin: 1rem;
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 1rem;
  text-align: center;
}
```

This is the end of Quick Lab 6

7. Quick Lab 7 – Event Handling

7.1. Objectives

- To be able to use event handlers to control events in your SPAs

7.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest eventhandling
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd eventhandling
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.
6. You will need to install a dependency to allow this project to run. A whimsical bit of interactivity which can add a touch of joy to a webpage. In the terminal enter the following code.

```
npm install js-confetti
```

7. Create a new file called **AddConfetti.jsx**.
8. At the top, create an import for **JSConfetti** from 'js-confetti (note the capital letters).
9. Create a **const** called **jsConfetti** and set it equal to **new JSConfetti()**.
10. Under this, create a new **const** called **addConfetti**. This should be a stateless arrow function with the following line of code:

```
jsConfetti.addConfetti({ emojis: ['⚛️', '⭐'], confettiNumber: 16  })
```

11. The emojis are taken from the same Unicode site in previous labs. You can choose any you like.
12. Don't forget to export the **addConfetti const**.
13. In **App,js**, create an import for **addConfetti** from **'./AddConfetti**.
14. Create an **h1** which contains the text 'The confetti Button'.
15. Create a button with the **onClick** parameter for **{ addConfetti }** and let the button say 'Confetti Time'.
16. Save and test your webpage.

AddConfetti.jsx

```
import JSConfetti from 'js-confetti'

const jsConfetti = new JSConfetti()

const addConfetti = () =>
    jsConfetti.addConfetti({ emojis: ['⚛️', '⭐'], confettiNumber: 16  })


export default addConfetti;
```

App.jsx

```
import './App.css';
import addConfetti from './AddConfetti';

function App() {
  return (
    <div className="App">
      <h1>The Confetti Button</h1>
      <button onClick={addConfetti}>Confetti Time!!</button>
    </div>
  );
}

export default App;
```

This is the end of Quick Lab 7

8. Quick Lab 8 – State in react

8.1. Objectives

- To be able to add State to your webpages.

8.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest state
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd state
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.
6. In **App.jsx**, import **{ useState }** from 'react'.
7. At the very top of the function, create State as below:

```
const [ temperature, setTemperature ] = useState(20)
```

8. Directly underneath the **const**, create a function called **increaseTemperateure()**.
9. This function should call the **setTemperature** function from State. You need to take the temperature State and add one to it:

```
setTemperature( temperature + 1)
```

10. Do the same for **decreaseTemperature()**, Taking one away this time.
11. Inside the return, put an **h1** tag that has the text 'React Thermostat'.
12. Create an **h2** which calls the **{ temperature }** state and a °C symbol.
13. In a **<div>** underneath, add 2 buttons. One should have a plus, the other a minus.
14. Each button should have an onClick. The + button should call **increaseTemperature()**, the minus should call **decreaseTemperature()**.
15. Your temperature should go up or down based on which button you click.
16. Add another State below the first:

```
const [ count, setCount ] = useState(1)
```

17. Add another function called **double()**. This should call the **setCount** function and then take count * 2.
18. Back in the return, add another **h1** at the bottom and add the text 'Powers off 2".
19. In an **h2** under this add the following:

```
{ count.toLocaleString() }
```

20. This will change the format of the State in the webpage.
21. Finally, add a button with an **onClick** which calls **{ double }** and has X2 on it.
22. Test your code.

App.jsx

```
import React, { useState } from 'react'
import './App.css'

export default function App() {
  const [temperature, setTemperature] = useState(20)
  const [count, setCount] = useState(1)

  function increaseTemperature() {
    setTemperature(temperature + 1)
  }

  function decreaseTemperature() {
    setTemperature(temperature - 1)
  }
function double() {
  setCount(count * 2)
}

  return (
    <div>
      <h1>🌡️ React Thermostat</h1>
      <h2>{temperature}°C</h2>
      <div>
        <button onClick={increaseTemperature}>+</button>
        <button onClick={decreaseTemperature}>-</button>
      </div>
      <div>
        <h1>Powers of 2</h1>
        <h2>{count.toLocaleString()}</h2>
        <button onClick={double}>X2</button>
      </div>
    </div>
  )
}
```

This is the end of Quick Lab 8

9. Quick Lab 9 – Inverse Dataflow

9.1. Objectives

- To be able to demonstrate how data can flow back up the tree.

9.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest inverse
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd inverse
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.
6. Create anew folder in **src** called **components**.
7. Inside this, create 3 files, CounterDisplay.jsx, IncrementButton.jsx and ResetButton.jsx.
   - **CounterDisplay.jsx** should contain a function which takes the prop of **{ count }**. It should return an **h1** which simply contains **{ count }**.
   - **IncrementButton.jsx** should contain a function which takes the prop of **{increment}**. It should return a **button** which has an **onClick** which calls **{increment}** and says 'Increment the Counter'.
   - **ResetButton.jsx** should contain a function which takes the prop of **{ reset }**. It should return a **button** which has an **onClick** which calls **{ reset }** and says 'Reset to 0'.
8. In **App.jsx** import **{ useState }** from 'react'.
9. Import **CounterDisplay** from './components/CounterDisplay'.
10. Import **ResetButton** from './components/ResetButton'.
11. Import **IncrementButton** from '/components/IncrementButton'.
12. At the top of the App function, create State called count, set to 0.
13. Inside the return, create calls for the three components:

```
<CounterDisplay count={count} />
<IncrementButton />
<ResetButton />
```

14. Underneath the creation of the count State, create a function called **handleIncrement()**. This should setCount to previous + 1. Previous is a block of code which takes the current State and can change it. Your code should look like this:

```
setCount((previous) => previous +1)
```

15. Create another function called **handleReset()**. This should simply setCount to 0.
16. Back in the return, in the call for **IncrementButton**, add a prop of:

```
increment = {handleIncrement}
```

17. In the call for **ResetButton** , add a prop of:

```
reset = { handleReset }
```

18. Save and test your code. The display number should start at 0, go up by one when you click increment, and return to 0 when the rest button is clicked.

    The thing to remember is that the State is being passed back up the tree and then back down again to any "interested" components.

CounterDisplay.jsx

```
export default function CounterDisplay({ count }) {
    return <h1>{count}</h1>
  }
```

IncrementButton.jsx

```
export default function IncrementButton({ increment }) {
    return (
      <button onClick={increment}>
        Increment the counter!
      </button>
    )
  }
```

ResetButton.jsx

```
export default function ResetButton({ reset }) {
    return (
      <button onClick={reset}>
        Reset to 0
      </button>
    )
  }
```

App.jsx

```
import { useState } from 'react';
import CounterDisplay from './components/CounterDisplay';
import ResetButton from './components/ResetButton';
import IncrementButton from './components/IncrementButton';
import './App.css';

export default function App() {
  const [count, setCount] = useState(0)

  function handleIncrement() {
    setCount((previous) => previous + 1)
  }

  function handleReset() {
    setCount(0)
  }

  return (
    <main>
      <CounterDisplay count={count} />
      <IncrementButton increment={handleIncrement} /> &nbsp;
      <ResetButton reset={handleReset} />
    </main>
  )
}
```

This is the end of Quick Lab 9

10. Quick Lab 10 - useEffect

10.1. Objectives

- To be able to use the useEffect hook.

10.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and **cd** into the folder.
3. Run the following code to install react.

```
npx create-vite@latest effect
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd effect
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.
6. You will need to install a dependency to allow this project to run. A whimsical bit of interactivity which can add a touch of joy to a webpage. In the terminal enter the following code.

```
npm install js-confetti
```

7. Create a new file called **AddConfetti.jsx**.
8. At the top, create an import for **JSConfetti** from 'js-confetti (note the capital letters).
9. Create a **const** called **jsConfetti** and set it equal to **new JSConfetti()**.
10. Under this, create a new **function** called **addConfetti**. This should take in the prop of { text }. Add the following code to the function:

```
js.Confetti.addConfetti({
     emojis: [ text ],
     confettiNumber: 12,
     emojiSize: 50,
})
```

11. Don't forget to export the function at the bottom.
12. Back in **App.jsx**, create a State called **count** which is set to 0.
13. Create a State called **isConfettiEnabled** which is set to **true**.
14. In the return, create a containing **<main>** tag.
15. Inside there, create an **h1** tag which reads 'Confetti Counter'.
16. Create a button with an onClick which has an anonymous function. This function should change the **setIsConfettiEnabled** to the opposite Boolean value:

```
<button onClick= { () => setIsConfettiEnabled (( prev ) => !prev  )}
```

```
</button>
```

17. Use Conditional rendering and the ternary to change the Text on the button.

```
Confetti: { isConfettiEnabled ? 'On': 'Off' }
```

18. Under this, create a **h2** tag which outputs the **{ count }** State.
19. Create a **button** that has an **onClick** which changes the State of count:

```
<button onClick={() => setCount((previous) => previous + 1)}
+1
</button>
```

20. Add a **useEffect** which checks **if (isConfettiEnabled)** is **true** every time the count state is changed. It should run the following code:

```
addConfetti( {text: count.toString() } )
```

21. Save and test your code.

AddConfetti.jsx

```
import JSConfetti from 'js-confetti'

const jsConfetti = new JSConfetti()
function addConfetti ({ text }) {
  jsConfetti.addConfetti({
    emojis: [text],
    confettiNumber: 12,
    emojiSize: 50,
  })
}

export default addConfetti
```

App.jsx

```
import {useState, useEffect} from 'react'
import addConfetti from './AddConfetti';
import './App.css';

function App() {
  const [count, setCount] = useState(0)
  const [isConfettiEnabled, setIsConfettiEnabled] = useState(true)

  useEffect(() => {
    // This should run whenever "count" changes.
    if (isConfettiEnabled) {
      addConfetti({ text: count.toString() })
    }
  }, [count])

  return (
    <main>
      <h1>🎉 Confetti Counter</h1>

      <button onClick={() => setIsConfettiEnabled((prev) => !prev)}>
        Confetti: {isConfettiEnabled ? 'on' : 'off'}
      </button>

      <h2>{count}</h2>

      <button onClick={() => setCount((previous) => previous + 1)}>
   +1
</button>
    </main>
  );
}

export default App;
```

This is the end of Quick Lab 10

11. Quick Lab 11 – useEffect to Collect data

11.1. Objectives

- To be able to use the useEffect hook to collect data on the initial render.

11.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest onload
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd onload
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.
6. In **App.jsx**, import { useState, useEffect } from 'react'.
7. Before the App function, create a const called **apiUrl** which is equal to:

```
'https://dog.ceo/api/breeds/image/random'
```

8. Directly inside the App function, initialise State called **imageUrl** which should be set to an empty string ('').
9. In the return, create an enclosing **<main>** tag. Inside this tag, create a **h1**, with the text 'Go Fetch'.
10. Under that, an **img** tag with the following parameters:

```
Width={300} src={imageUrl} alt=''
```

11. Finally, above the return, create a useEffect. This should only run on first render and should include the following fetch:

```
Fetch(apiUrl)
   .then( (response) => response.json() )
   .then( (data) => setImageUrl(data.message) )
```

12. Save and test your code.

App.jsx

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

const apiUrl = 'https://dog.ceo/api/breeds/image/random'

function App() {
  const [imageUrl, setImageUrl] = useState('')

  useEffect(() => {
    fetch(apiUrl)
      .then((response) => response.json())
      .then((data) => setImageUrl(data.message))
  }, [] )

  return (
    <div className="App">
      <main>
        <h1>Go Fetch</h1>
        <img width={300} src={imageUrl} alt=""/>
      </main>
    </div>
  );
}

export default App;
```

This is the end of Quick Lab 11

12. Quick Lab 12 – Editing Json in react

12.1. Objectives

- To be able to start a json server and add data to it .

12.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest simplejson
```

Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd simplejson
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.
6. Create a new folder in the root of **simplejson** called data. In this folder, create a file called **db.json** This file should contain some simple resourced json. e.g.

```
{
  "users": [
    {
      "firstName": "Andy",
      "lastName": "Smith",
      "title": "Dr",
      "id": 1
    },
    {
      "firstName": "Maggie",
      "lastName": "May",
      "title": "Mrs",
      "id": 2
    }
  ]
}
```

7. Start a new terminal window in VSCode, ensure you are focused on the root of the react project and type the following code:

```
npx json-server --watch data/db.json --port 8000
```

This will spin up a local json server, that watched the file you just created. It can also serve requests to access the data. If you open a browser window, you can navigate to:

```
Localhost/8000/users
```

and you will see the json data stored there. You can even add an id to the end of the url e.g. /1 or /2 and you will see that particular record in the browser.

8. Create a new component called **Create.jsx**. This is where you will create the form that adds users to your json file.
9. Import **{ useState }** from 'react' at the top.
10. Create a stateless function called **Create**.
11. Immediately inside the function, create State for **first**, **second** and **title**, all set to (''). These will be how we identify the form inputs.
12. In the return, create a **<form>** tag. Within the tag we will need two text inputs and a select box.

```
<label>First Name: </label>
<input type="text"
required value={first}
onChange={(e) => setFirst(e.target.value)}
     />

<label>Second Name: </label>
<input type="text"
required value={second}
onChange={(e) => setSecond(e.target.value)}
     />

    <label>Title:</label>
    <select
     value={title}
     onChange = {(e) => setTitle(e.target.value)} >
          <option value="mr">Mr</option>
          <option value="mrs">Mrs</option>
<option value="dr">Dr</option>
     </select>
```

The (e) is the event object. In this case the event is the changing of the content of the input box which simultaneously updates the State and the textbox.

13. Add a button before the closing **</form>** tag which reads "add user".
14. When the form is filled in and submitted, an event needs to be raised to stop the form doing what it usually does. Create a function above the return called **handleSubmit.** It should take in the (e) parameter.
15. Inside the function, you need to prevent the default action of the event with:

```
e.preventDefault()
```

16. Directly underneath this, create a **const** called user and set it equal to **{first, second, title}**. These are now the contents of the text boxes and select box.
17. Now that the data has been collected from the user inputs, a **fetch()** can be invoked and that data POSTED to the json file via the running json server. Inside the **handlesubmit()** function, type the following code:

```
fetch('http://localhost:8000/users', {
    method: 'POST',
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(user)
})
```

This takes the user **const**, opens the json file and performs a 'POST', which tells the server to put the information into the file.

18. At the bottom of the function, call all 3 of the setState functions passing an empty string ('') to each of them. This clears the form ready for new data.
19. Finally, in the opening form tag add the following event listener:

```
onSubmit={handleSubmit}
```

20. With your json file open in VSCode, add a user and submit. The form should clear and you should see the new data appear in the Json file.

App.jsx

```
import Create from './Create';
import './App.css';

function App() {
  return (
    <div className="App">
        <h2>Add new Records</h2>
        <Create />
    </div>
  );
}

export default App;
```

Create.jsx

```
import { useState } from "react";
const Create = () => {

    const [first, setFirst] = useState('');
    const [second, setSecond] = useState('');
    const [title, setTitle] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        const user = { first, second, title }

        fetch('http://localhost:8000/users', {
            method: 'POST',
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify(user)
        })

        setFirst('');
        setSecond('');
        setTitle('');
    }

    return (
        <form onSubmit={handleSubmit}>
            <label>First Name:</label>
            <input type="text"
             required
             value={first}
             onChange={(e) => setFirst(e.target.value)}
            />
            <br />
            <label>Last Name:</label>
            <input type="text"
             required
             value={second}
             onChange={(e) => setSecond(e.target.value)}
            />
            <br />
            <label>Title:</label>
            <select
            value={title}
            onChange={(e) => setTitle(e.target.value)}>
                <option value="mr">Mr</option>
                <option value="mrs">Mrs</option>
                <option value="dr">Dr</option>
            </select>
            <br />
            <button>Add User</button>
        </form>
    );
 }
export default Create;
```

This is the end of Quick Lab 12

13. Quick Lab 13 – Context

13.1. Objectives

- To be able create simple examples of using Context in react. This builds on the content from Quick Lab 5 where you dragged information in from a Json file. This time, we will build a fruit stall.

13.2. Activity

1. Create an empty folder in your documents and open it in VSCode.
2. Right click this file in VSCode and open a terminal in it, or open a terminal and cd into the folder.
3. Run the following code to install react.

```
npx create-vite@latest context
```

   Select React and then JavaScript when prompted.

4. Once your install has finished, type the following commands.

```
cd context
```

```
npm install
```

```
npm run dev
```

5. Check that the react page has opened in a browser, remove the default from **App.jsx**.

   In this Lab, we will create and add context to our react app, ensuring that it only passes the state to the children components that need it.

6. In your **src** folder create a file called **itemsData.json**. Drop the following information into it:

```
  {
    "id": 1,
    "symbol": "🍎",
    "name": "Apple",
    "price": 0.3
  },
  {
    "id": 2,
    "symbol": "🍍",
    "name": "Pineapple",
    "price": 1
  },
  {
    "id": 3,
    "symbol": "🍉",
    "name": "Watermelon",
    "price": 4
  },
  {
    "id": 4,
    "symbol": "🥝",
    "name": "Kiwi",
    "price": 0.5
  },
  {
    "id": 5,
    "symbol": "🍊",
    "name": "Orange",
    "price": 0.3
  },
  {
    "id": 6,
    "symbol": "🍋",
    "name": "Lemon",
    "price": 0.2
  }
]
```

7. In your **src** folder, create a folder called **components**. Inside here create 3 files, **CartList.jsx**, **ItemCard.jsx** and **ItemList.jsx**.
8. In your **src** folder, create a folder called **context**. Inside here create a file called **cart-context.jsx**.
9. In **ItemCard**, We will do something similar to what we did in Lab 05. Import **React** at the top and create a default export called **ItemCard** which takes in **{id, symbol, name, price}** as parameters. Notice that these correspond to the keys in the json file.
10. Create a function called **handleAddToCart**(). This should contain a call to the **addToCart** function which needs the **id** of the clicked element to be passed to it. The **addToCart** function won't be recognised at the moment. We will solve this issue later.

```
function handleAddToCart() {
    addToCart(id)
}
```

11. In the return for **ItemCard**, drop the following code:

```
return (
    <div className="item-card">
      <div className="symbol">{symbol}</div>
      <h3>{name}</h3>
      <p>£{price.toFixed(2)}</p>
      <button onClick={handleAddToCart}>Add to cart</button>
    </div>
  )
}
```

This puts a **button** on each card that will invoke the handleAddToCart function.

12. In **ItemList.jsx** we need to complete a couple of imports. Firstly, **ItemCard** and secondly **itemsData.json**.
13. Create (and export) a function called called **ItemsList** and give it a **return** statement.
14. Within the **return** statement and inside a containing **<div>** create an **<h2>**which outputs 'Products'.
15. Create a new **<div>** with a **className** of "items-grid". Inside here you need to map the **itemsData** as a component call with the following state passed as props:

```
<div className="items-grid">
        {itemsData.map((item) => (
          <ItemCard
            key={item.id}
            id={item.id}
            symbol={item.symbol}
            name={item.name}
            price={item.price}
          />
        ))}
      </div>
```

16. Next we will create the **context**. In **cart-context.js**, we need to import **createContext** and **useState** from **'react'** and **itemsData** from the json file.

    Drop the following code into the file:

```
export const CartContext = createContext()

export function CartProvider({ children }) {
  const [itemsInCart, setItemsInCart] = useState([])

  function addToCart(itemId) {
    const item = itemsData.find((item) => item.id === itemId)
    setItemsInCart((prev) => [...prev, item])
  }

  const contextValue = {
    itemsInCart,
    addToCart
  }

  return (
    <CartContext.Provider value={contextValue}>
        {children}
    </CartContext.Provider>
  )
}
```

Here, we are creating **CartContext** and setting up a context provider. It will pass the context to the consumers (children) directly, without having to pass it through all other components.

The **addToCart()** function simply adds the **id** of the item clicked so that the symbol for that item can be rendered on the webpage.

Finally, we create an array called **contextValue**, which is passed from the provider, to be collected and used by the consumer, just as any other prop.

17. Now we can add our context to the **ItemCard.jsx** file. First, import **{ useContext }**, then import **{ cartContext }** from the relevant folder.
18. Finally, add a single line of code inside the **ItemCard** function that will allow the button click to update the context. To do this add the following line of code directly underneath the function definition:

```
const { addToCart } = useContext(CartContext)
```

19. Now we have context, we can properly build the **CartList** which will show us the result of our clicks. Inside **CartList.jsx import { useContext },** then **{ CartContext }**.
20. Create a function called **CartList()** and immediately create a **const** for **{itemsInCart}** and set it equal to **useContext(CartContext)**.
21. In the **return** for this function, inside a containing **<div>**add an **<h2>** which outputs "Cart".
22. Create a new**<div>** with a **className** "cart-wrapper".
23. Inside this **<div>** create a simple **map** from the **itemsInCart const** and within a **<span>** output **{ item.symbol }**: it should look like this:

```
  return (
    <div>
      <h2>Cart</h2>
      <div className="cart-wrapper">
        {itemsInCart.map((item) => (
          <span>{item.symbol}</span>
        ))}
      </div>
    </div>
  )
}
```

24. Don't forget to default export.
25. We can finally call all context and components into **App.jsx**. As is always the case, the imports are what makes this stage possible. Import **CartList**, **ItemsList** and **{CartProvider}** from their paths.
26. Create an import for **'./style.css'** too, we will create the CSS shortly for styling of our app. Delete the link to **App.css**.
27. Inside the **return**, ensure you have an enclosing **<main>**.

28. Inside the enclosing tags, create a **<h1>** which reads "React Fruit Market".
29. You need to now initialise the Context Consumer. Create opening and closing **<CartProvider>** tags.
30. Between them, call <CartList /> and <ItemsList />.
31. Finally, create a file called **style.css** inside the **src** folder. Drop the following code into it.

```
body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
Oxygen,
    Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
  margin: 1.5rem;
}

h1 {
  margin-bottom: 1.5rem;
}

.items-grid {
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;
}

.item-card {
  width: 200px;
  padding: 1rem;
  background-color: rgb(248, 248, 248);
  border-radius: 3px;
  border: solid 1px rgb(220, 220, 220);
}

.item-card .symbol {
  font-size: 2rem;
  text-align: center;
}

.item-card h3 {
  margin-bottom: 0.5rem;
}

.item-card p {
  margin: 0 0 0.5rem;
}
```

```
.item-card button {
  width: 100%;
}

.cart-wrapper {
  display: flex;
  align-items: center;
  flex-wrap: wrap;
  padding: 1rem;
  min-height: 1rem;
  border: solid 1px rgb(204, 204, 204);
  border-radius: 3px;
}
```

You now have everything set up for context to be passed between the parent and its relevant children. The website should render and add the symbol of the fruit you clicked in the cart at the top.

32. Save and run your app in a browser.

Here are the files should you get stuck:

cart-context.js

```
import React, { createContext, useState } from 'react'
import itemsData from '../itemsData.json'

export const CartContext = createContext()

export function CartProvider({ children }) {
  const [itemsInCart, setItemsInCart] = useState([])

  function addToCart(itemId) {
    const item = itemsData.find((item) => item.id === itemId)
    setItemsInCart((prev) => [...prev, item])
  }

  const contextValue = {
    itemsInCart,
    addToCart
  }

  return (
    <CartContext.Provider
value={contextValue}>{children}</CartContext.Provider>
```

```
  )
}
```

CartList.jsx

```
import React, { useContext } from 'react'
import { CartContext } from '../context/cart-context'

export default function CartList() {
  const { itemsInCart } = useContext(CartContext)

  return (
    <div>
      <h2>Cart</h2>
      <div className="cart-wrapper">
        {itemsInCart.map((item) => (
          <span>{item.symbol}</span>
        ))}
      </div>
    </div>
  )
}
```

ItemCard.jsx

```
import React, { useContext } from 'react'
import { CartContext } from '../context/cart-context'

export default function ItemCard({ id, symbol, name, price }) {
  const { addToCart } = useContext(CartContext)

  function handleAddToCart() {
    addToCart(id)
  }

  return (
    <div className="item-card">
      <div className="symbol">{symbol}</div>
      <h3>{name}</h3>
      <p>£{price.toFixed(2)}</p>
      <button onClick={handleAddToCart}>Add to cart</button>
    </div>
  )
}
```

ItemList.jsx

```
import React from 'react'
import ItemCard from './ItemCard'
import itemsData from '../itemsData.json'

export default function ItemsList() {
  return (
    <div>
      <h2>Products</h2>
      <div className="items-grid">
        {itemsData.map((item) => (
          <ItemCard
            key={item.id}
            id={item.id}
            symbol={item.symbol}
            name={item.name}
            price={item.price}
          />
        ))}
      </div>
    </div>
  )
}
```

App.jsx

```
import React from 'react'
import CartList from './components/CartList'
import ItemsList from './components/ItemsList'
import { CartProvider } from './context/cart-context'
import './style.css'

export default function App() {
  return (
    <main>
      <h1>React Fruit Market</h1>
      <CartProvider>
        <CartList />
        <ItemsList />
      </CartProvider>
    </main>
  )
}
```

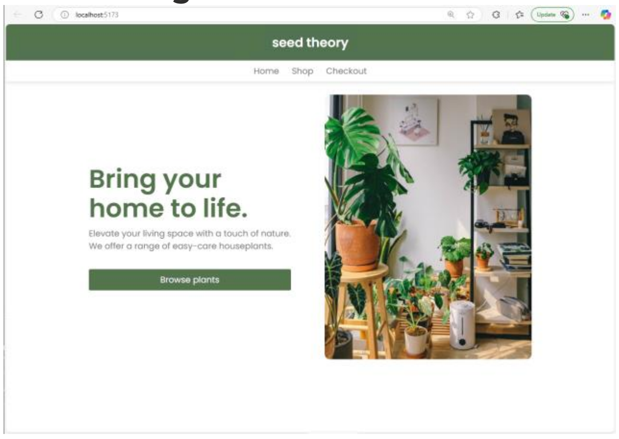
This is the end of Quick Lab 13

14. React Challenge 1

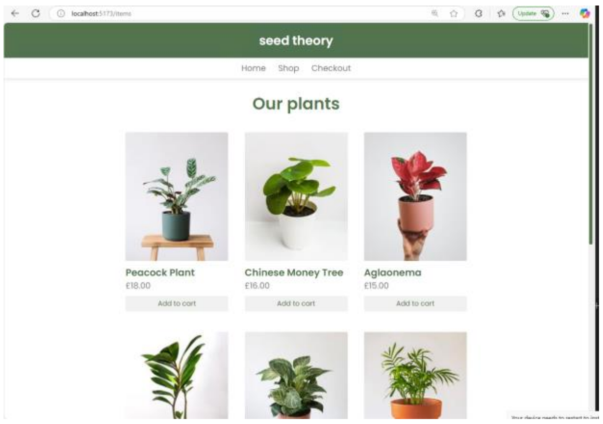
14.1. Task summary

Build a static website using React based around an online shop that sells potted plants. Ultimately the site needs three pages:

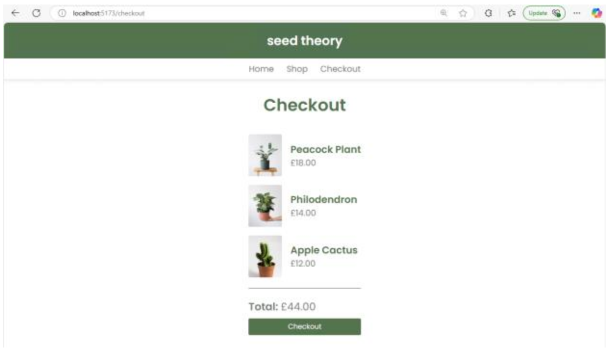
- **A home Page**


localhost:5173
seed theory
Home Shop Checkout
Bring your home to life.
Elevate your living space with a touch of nature. We offer a range of easy-care houseplants.
Browse plants

- **A products page** that allows users to select plants to put in their baskets.



- **A Checkout page** that will (ultimately) allow users to review the content of their basket.



Don't worry about matching the visual design perfectly. Resources have been provided to help speed up your work.

14.2. Concepts

- HTML/CSS/JS fundamentals
- React components, props, rendering lists
- React Router

14.3. Resources to help you

- Homepage image (link)
- Items data (link) *

14.4. Creating a React app

Follow the steps taken in the previous lab (6) to create a new React project that supports routing.

14.5. Tips

- Instead of viewing the designs as a whole, break down the problem by identifying the different elements (components) that make up each page.
- Make sure to create a reusable component for the repeated items on the products page.
- The "Add to cart" buttons should not do anything at this stage.
- The checkout page Can either show no items or all the items held in the json file.

Hard-code this data into your app by placing it in a file called `itemsData.json` within your project. To render this data on the shop page, you should be familiar with using `Array.map` to render lists of React elements.

This is the end of Challenge 1

15. React Challenge 2

15.1. Concepts

- Handling events
- React state (useState)
- Conditional rendering

15.2. Task summary

On the potted plant app's items page, each item component has an "Add to cart" button that currently does nothing. Change this so that clicking the buttons toggles the text back and forth between "Add to cart" and "In cart". Click this link for a live demo.

At this stage, there's no need to store the cart data - simply change the appearance of the buttons.

15.3. Tips

- To change the visual appearance of the buttons, you can change which CSS classes are applied to the button element.

You can also use this time to focus on finishing anything from Challenge 1

16. React Challenge 3

16.1. Concepts

- useEffect hook
- Using external data
- State (useState)
- Context

16.2. Task summary

In this challenge we return to the potted plant shop app. Currently, the items data is hard coded into the app in a JSON file. You need to change the implementation so the app fetches the data from an external source when the site is first loaded.

The JSON data is hosted here.

Alternatively, if you are developing locally, you can use json-server to host the data on a fake API on your development machine.

16.3. Tips

- **Do not forget the dependency array** in your useEffect.

16.4. Extensions

16.4.1. Keep track of cart content via context

Think about adding context to the project that keeps track of what plants are added and removed from the cart. Given the context will be hosting the list of cart items, it should be possible to add functionality to the cart that allows clients to remove, insert, get, check to see if items are in the cart and calculate the total value of all the items in the cart.

16.4.2. Enable the Checkout page's Checkout button

To do this you may find the following steps of some use:

- Create a data folder and an empty db.json file
- Launch JSON Server on port 8000 to monitor the **data/db.json** file
- Extend the **handleCheckout** function.
  - Create an empty collection of order_items.
  - Loop around the itemIds collection (from the context)
    - Use getItemById to get items and push them onto the order_items collection.
  - Create an order object that contains the order_items and the order_total (calculated by calling the context's calculateTotal function).
  - Use a fetch operation to POST the order to the db.json api
  - Empty the cart after an order has been processed.

- Needs a new function to be added to the cart context that sets the itemIds useState to an empty array.
- Call the navigate function to return to the home page.

17. React Challenge 4 – Blogs

17.1. Concepts

- useEffect hook
- Updating external data
- Forms to collect data

17.2. Task summary

You will be given the starting files that take information from a json file, via the json server. The basic set up will allow you to display existing Blogs onto the home page, it will also provide a Navbar and links to a page that allows you to create a new blog entry.

Your job will be to add functionality to the webApp that link to an individual blog view, add a new blog, and delete any blogs you no longer want.

17.3. Base Files:

NavBar.jsx

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

const Navbar = () => {
  return (
    <nav className="navbar">
      <h1>Andy's Blog</h1>
      <div className="links">
        <Link to="/">Home</Link>
        <Link to="/create">New Blog</Link>
      </div>
    </nav>
  );
}

export default Navbar;
```

Notice the links to New Blog and home.

BlogList.jsx

```
import { Link } from 'react-router-dom';

const BlogList = ({ blogs }) => {
  return (
    <div className="blog-list">
      {blogs.map(blog => (
        <div className="blog-preview" key={blog.id} >
            <Link to={`/blogs/${blog.id}`}>
            <h2>{ blog.title }</h2>
            <p>Written by { blog.author }</p>
            </Link>
        </div>
      ))}
    </div>
  );
}

export default BlogList;
```

Here, the links are set up to be able to click the blog and open the individual blogs. It takes the props from the component call in the HomePage.

HomePage.jsx

```
import { useState, useEffect } from "react";
import BlogList from "./BlogList";

const HomePage = () => {

    const [blogs, setBlogs] = useState(null)
    const [url, setUrl] = useState('http://localhost:8000/blogs')

    useEffect(() => {
        fetch(url)
            .then((res) => res.json())
            .then((data) => setBlogs(data))
    }, [url])

    return (
        <div className="content">
            { blogs && <BlogList blogs={blogs} />}
        </div>
```

```
     );
}
export default HomePage;
```

Notice, the Use Effect which calls the blogs from the Json server. It only allows refreshes of the original page when the url state changes. There is also some conditional rendering which waits for the blogs to exist before putting them on the home page. This can be used to present a temporary 'Loading…….' Message. The fetch is a promise which returns the data or tells you it can't complete the task.

Db.json

```
{
  "blogs": [
    {
      "title": "Starting Data",
      "body": "This is simply some text which will serve as a placeholder for the initial content of the blogs. It is pointless and meaningless other than serving a rendering purpose.",
      "author": "Dave",
      "id": 1
    },
    {
      "title": "My Second Blog",
      "body": "This is simply some text which will serve as a placeholder for the initial content of the blogs. It is pointless and meaningless other than serving a rendering purpose.",
      "author": "Andy",
      "id": 2
    }

  ]
}
```

This requires a Json server to be running on port 8000 with the following file being accessed.

17.4. Challenge

Create a page with an HTML form that collects and updates State related to a new blog post. It should prevent the default behaviour and instead use a fetch to add the form entry to the **db.json** file.

A placeholder for the **BlogDetails** page already exists. You need to add code to it, so it displays full details of the Blog whose id has been appended to the url that routes the user to the page. The page also needs a delete button which should delete the blog from the **db.json** file.

A place holder for the **CreatePage** also exists which needs to be fleshed out with appropriate code that allows users to enter a blog title, body and author into text boxes and via the click of a button add the blog to the **db.json** file.

17.5. Tips

17.5.1. The BlogDetails component

The component uses a useParams hook that allows access the parameters in the URL (in this case a number that represents the blog **id**).

The following code will delete the blog with the specified id from the **db.json** and navigate the user back to the home page.

```
fetch('http://localhost:8000/blogs/' + blog.id, {
    method: 'DELETE'
}).then(() => {
    navigate('/');
})
```

The navigate function needs to be declared (and imported) as a useNavigate hook.

```
import { useNavigate, useParams } from "react-router-dom";
const navigate = useNavigate();
```

17.5.2. The CreatePage component

You may well want to make use of the navigate functionality as described above after a new blog is added to the **db.json** file.

17.6. Solutions

BlogDetails.jsx

```
import { useNavigate, useParams } from "react-router-dom";
import { useState, useEffect } from "react";

const BlogDetails = () => {
  const { id } = useParams();
  const [blog, setBlog] = useState(null)
  const navigate = useNavigate();

  const handleClick = () => {
    fetch('http://localhost:8000/blogs/' + blog.id, {
      method: 'DELETE'
    }).then(() => {
      navigate('/');
    })
  }

  useEffect(() =>{
    fetch('http://localhost:8000/blogs/' + id)
      .then(res => res.json())
      .then(data =>setBlog(data) )
  }, [])

  return (
    <div className="blog-details">
      { blog && (
        <article>
          <h2>{ blog.title }</h2>
          <p>Written by { blog.author }</p>
          <div>{ blog.body }</div>
          <button onClick={handleClick}>delete</button>
        </article>
      )}
    </div>
  );
}

export default BlogDetails;
```

CreatePage.jsx

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

const Create = () => {
  const [title, setTitle] = useState('');
  const [body, setBody] = useState('');
  const [author, setAuthor] = useState('');
  const navigate = useNavigate();

  const handleSubmit = (e) => {
    e.preventDefault();
    const blog = { title, body, author };

    fetch('http://localhost:8000/blogs/', {
      method: 'POST',
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(blog)
    }).then(() => {
      navigate('/');
    })
  }

  return (
    <div className="create">
      <h2>Add a New Blog</h2>
      <form onSubmit={handleSubmit}>
        <label>Blog title:</label>
        <input
          type="text"
          required
          value={title}
          onChange={(e) => setTitle(e.target.value)}
        />
        <label>Blog body:</label>
        <textarea
          required
          value={body}
          onChange={(e) => setBody(e.target.value)}
        ></textarea>
        <label>Blog author:</label>
        <input
            type="text"
           value={author}
          onChange={(e) => setAuthor(e.target.value)}
        />

        <button>Add Blog</button>
      </form>
    </div>
  );
}

export default Create
```

QA

Learn. To Change.

QA.com