React Cheatsheet

26.03.2022 React-BootCamp

React Elements

React elements are written just like regular HTML elements. You can write any valid HTML element in React. We write React elements using a feature called JSX.

```
<h1>Header</h1>
<p>My Text</p>
<button>Delete</button>
```

But be careful of the single-tag elements. They must end in a forward slash.

```
<img src="img.png" />
<br />
<hr />
```

React Attributes

JSX is Javascript. We use camelCase naming in JavaScript. Attributes are written as camelCase.

```
 <div className="header"></div>
```

React Fragments

React requires that all returned elements be returned within a single “parent” component. If you don’t want to use a container element like a div, you can use a fragment.

```
function Component() {
  return (
    <React.Fragment>
      <h1>My header</h1>
      </p>My paragraph</p>
    <React.Fragment/>
  );
 }


function Component() {
  return (
    <>
      <h1>My header</h1>
      </p>My paragraph</p>
    </>
  );
 }
```

React Components

We can organize groups of elements into React components. Component names must start with a capital letter (that is, Header, instead of header). Components must return JSX.

React Functional Component

VSCode Shortcut => rfc

```
import React from 'react'

export default function App() {
 return (
   <div>App Component</div>
 )
}
```

React Arrow Function Component

VSCode Shortcut => rafc

```
import React from 'react'

export const App = () => {
 return (
   <div>App Component</div>
 )
}
```

React Props

React components can accept data passed to them called *props*. Props are passed from the parent component to a child component.

```
function Parent() {
  return <Child name="Hi Coders" />
}

function Child(props) {
  return <h1>Hello, {props.name} ! </h1>; // Hello, Hi Coders !
 }
```

```
function Parent() {
  return <Child name="Hi Coders" />
}

function Child({ name }) {
  return <h1>Hello, {name}!</h1>; // Hello, Hi Coders!
 }
```

React Conditionals

React components and elements can be conditionally displayed.

```
function App() {
      const isAuthUser = useAuth();

  if (isAuthUser) {

    return <AuthApp />;
  }

  return <UnAuthApp />;
}


OR


function App() {
      const isAuthUser = useAuth();

 return (
    <>
      <h1>My App</h1>
      {isAuthUser ? <AuthApp /> : <UnAuthApp />}
    </>
  )
 }
```

React Lists

Lists of React components can be output using the `.map()` function. Whenever you are looping over an array of data, you must include the *key* prop on the element or component over which you are looping. Additionally, this key prop must be given a unique value, not just an element index.

```
function StudentList() {
  const students = ["Fatih", "Sultan", "Mehmet"];

  return (
    <div>
      {students.map((student, index) => (
        <Student key={index} name={student} />
      ))}
    </div>
  );
 }
```

React Events

The most common event listeners are onClick for links/buttons and onSubmit for forms.

```
import React from "react";

export default function Hello() {
  function handleClick(event) {
    event.preventDefault();
    alert("Hello HiCoders");
  }

  return (
    <a href="/" onClick={handleClick}>
      Say Hi
    </a>
  );
  }
```

React Hooks

React hooks were introduced in React version 16.8 as a way to easily add reusable, stateful logic to React function components.

React useState Hook

useState is used instead of a simple variable because when state is updated, our component re-renders, usually to display that updated value. Like all hooks, we call `useState` at the top of our component and can pass it an initial value to put on its state variable.

```
import { useState } from 'react';

function Component() {

  const [stateValue, setStateValue] = useState(initialValue);
 }
```

```
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  function increase() {
    setCount(count + 1);
  }

  return <button onClick={increase}>Count is: {count}</button>;
 }
```

React useEffect Hook

If we want to interact with the “outside world”, such as using an API, we use the `useEffect` hook. useEffect is used to perform a side effect, which means to perform an operation that exists outside of our app that doesn’t have a predictable result. The basic syntax of useEffect requires a function as a first argument and an array as the second argument.

```
import { useEffect } from 'react';

function MyComponent() {
   useEffect(() => {
     // perform side effect here
   }, []);
 }
```

```
import { useEffect } from 'react';

function PostList() {
       const [posts, setPosts] = useState([]);

   useEffect(() => {
         fetch('https://jsonplaceholder.typicode.com/posts')
       .then(response => response.json())
       .then(posts => setPosts(posts));
   }, []);

   return posts.map(post => <Post key={post.id} post={post} />
 }
```

React Forms

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

export default function LoginForm() {
  let [username, setUsername] = useState("");
  let [password, setPassword] = useState("");

  function handleSubmit(event) {
    event.preventDefault();
    alert(`Loging in with ${username} and ${password}`);
  }

  function updateUsername(event) {
    setUsername(event.target.value);
  }

  function updatePassword(event) {
    setPassword(event.target.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" placeholder="Username" onChange={updateUsername} />
      <input type="password" placeholder="Password" onChange={updatePassword}
/>
      <input type="submit" value="Login" />
    </form>
  );
  }
```

Crud Operations

getAll

```
const getData = async () => {
   const dataURL = "http://localhost:8080/employee";
   const response = await fetch(dataURL);
   const data = await response.json();
   setState(data)
 };
```

getElementById

```
const getElementById = async (pElementId) => {
   const dataURL = `http://localhost:8080/employee/${pElementId}`;
   const response = await fetch(dataURL);
   const data = await response.json();
   setState(data)
 };
```

post

```
const createItem = async (pItem) => {
   const requestOptions = {
     method: "POST",
     headers: { "Content-Type": "application/json" },
     body: JSON.stringify(pItem),
   };
   await fetch("http://localhost:3002/employee", requestOptions);
 };
```

delete

```
const deletePerson = async (personId) => {
   await fetch(`http://localhost:8080/employee/${personId}`, {
     method: "DELETE",
   });
 };
```

put

```
const updatePerson = async (pId, pPerson) => {
     await fetch(`http://localhost:8080/employee/${pId}`, {
       method: "PUT",
       headers: { "Content-Type": "application/json" },
       body: JSON.stringify(pPerson),
     });
   };
```