Props, Event Handling, React Form Assignment:

Objective:

- Know what is props in React.
- Understand how to use React's props.
- Be able to understand event handling in React and how to use them.
- Be able to use form in React and understand what form controlled is.
- Be able to do form validation.

Props in React

React components use props to communicate with each other. Every parent component can pass some information to its child components by giving them props. Props might remind you of HTML attributes, but you can pass any JavaScript value through them, including objects, arrays, and functions.

(From react document https://react.dev/learn/passing-props-to-a-component)

1. Passing props to the child component

```
function App() {
  return (
    <div className="App">
      <Card header="This is title" body="Hello, World" number={99} />
    </div>
  );
}
```

NOTE: you can pass a class name to your JSX but it has to be in camel case "className"

In the above example we pass string and number to several props.

Let's see how we can access the props from the child component.

2. Access the props inside the child component

```jsx
import React from "react";
import "../styles/card.css";

function Card(props) {
  return (
    <div className="card-container">
      <h2 className="card-header">{props.header}</h2>
      <p className="card-body">{props.body}</p>
      <p>{props.number}</p>
    </div>
  );
}

export default Card;
```

NOTE: we use props follow by the name of your props that you pass down from the parent component (Your App component).

My card.css

```css
.card-container {
  width: 200px;
  height: 200px;
  padding: 10px;
  margin: 10px;
  border: 1px white solid;
  border-radius: 20px;

  display: flex;
  flex-direction: column;
  align-items: center;
  text-align: center;
}

.card-header {
  color: yellowgreen;
}

.card-body {
  color: lightsalmon;
}
```

The result:

3. Specify the props using destructuring syntax provided by javascript

(This MDN document talk about destructure syntax https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

```jsx
import React from "react";
import "../styles/card.css";

function Card({ header, body, number }) {
  return (
    <div className="card-container">
      <h2 className="card-header">{header}</h2>
      <p className="card-body">{body}</p>
      <p>{number}</p>
    </div>
  );
}

export default Card;
```

NOTE: You can see that we remove the props keyword and only use value provided in the function parameter.

4. You can set the default value to your props too!

```jsx
import React from "react";
import "../styles/card.css";

function Card({ header, body, number = 100 }) {
  return (
    <div className="card-container">
      <h2 className="card-header">{header}</h2>
      <p className="card-body">{body}</p>
      <p>{number}</p>
    </div>
  );
}

export default Card;
```

In this case if the number doesn't pass down to your component then the value will be 100.

5. Passing JSX as children

You can access your JSX that nested inside your component via props.children

```jsx
import React from "react";
import "../styles/card.css";

function Card({ header, children }) {
  return (
    <div className="card-container">
      <h2 className="card-header">{header}</h2>
      <div className="card-body">{children}</div>
    </div>
  );
}

export default Card;
```

```jsx
import Card from "./components/Card";

function App() {
  return (
    <div className="App">
      <Card header="This is title">
        <p>This is your children!</p>
        <ul>
          <li>hi!</li>
          <li>hi!</li>
          <li>hi!</li>
        </ul>
      </Card>
    </div>
  );
}

export default App;
```

Your “children” props contain a <p> tag and <ul> tag with its list items.

NOTE: Our component have closing tag just like normal HTML tag!!

6. How props change over time

A component may receive different props over time. Props are not always static!

However, props are immutable—a term from computer science meaning “unchangeable”. When a component needs to change its props (for example, in response to a user interaction or new data), it will have to “ask” its parent component to pass it different props—a new object! Its old props will then be cast aside, and eventually the JavaScript engine will reclaim the memory taken by them.

You can’t change props. When you need interactivity, you’ll need to set state.

Let’s see how we can change props using state!

7. Using useState with props concept

```jsx
import React from "react";

function Text({ color, text }) {
  return <p style={{ color: color }}>{text}</p>;
}

export default Text;
```

We have “Text” component that accept “color” and “text” as its props

```jsx
import { useState } from "react";
import Text from "./components/Text";

const Colors = ["red", "green", "blue", "yellow"];

function App() {
  const [index, setIndex] = useState(0);

  function handleChangeColor() {
    if (index >= 3) {
      setIndex(index % 3);
      console.log(index);
    } else {
      setIndex(index + 1);
    }
  }

  return (
    <div className="App">
      <button onClick={handleChangeColor}>Change color</button>
      <Text text={"Your text is here"} color={`${Colors[index]}`} />
    </div>
  );
}

export default App;
```

In your parent component (App component) we have a button that increase the index number. In the handleChangeColor function we have to check if the index is not > 3 because it will out of bound. In the Text component we pass “text” and color props. The color props is special because its value are depend on the value of index state.

Assignment 1:

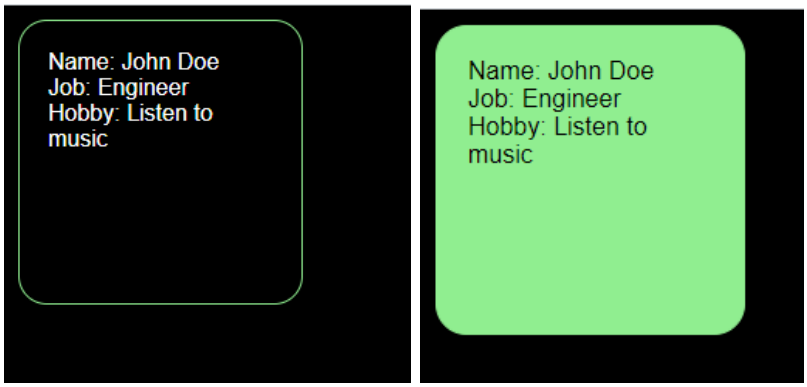
Objective:

- Know how to pass a props.

Create components folder inside your src folder for storing this assignment components.

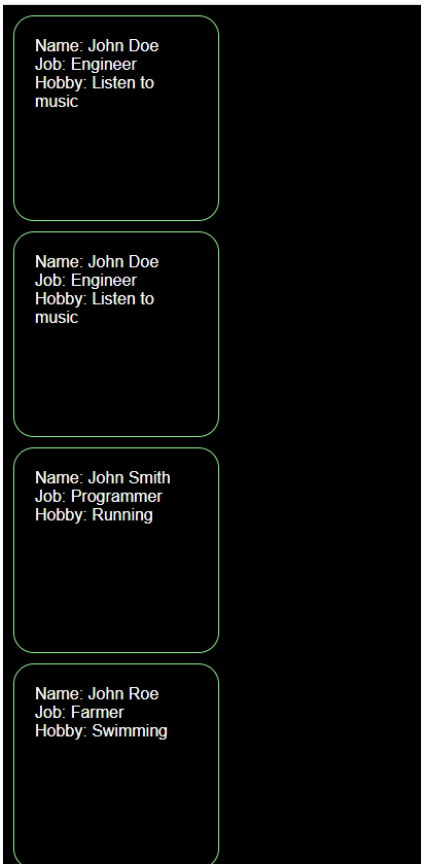
1. Create Card component inside components folder. Your component takes only one prop called “user” which is an Object that has name, job and hobby as its properties. Your Card component will render each of them inside <p> tag (3 <p> tags) respectively. Your Card component should have:
    - Width of 200 pixels
    - Height of 200 pixels
    - Border of 1 pixel, color lightgreen, solid style
    - Border radius of 20 pixels
    - Padding of 20 pixels
    - Margin of 10 pixels
    - Transition of background 1 second and text color 1 second

    When hovering over your Card component it should change smoothly by using transition to lightgreen background and text color of black



    You must create your own “user” object with its own value.

2. Create List component inside components folder. Your component takes only one prop which called “users”. It is an array that contain objects (at least 3 objects). It is the same object from previous assignment but have different information. Your job is taking the information from array and render each of them to Card component from previous assignment.



Event Handling in React

React lets you add event handlers to your JSX. Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on.

1. To add an event handler, you will first define a function and then pass it as a prop to the appropriate JSX tag.

For example, here is a button that doesn't do anything yet:

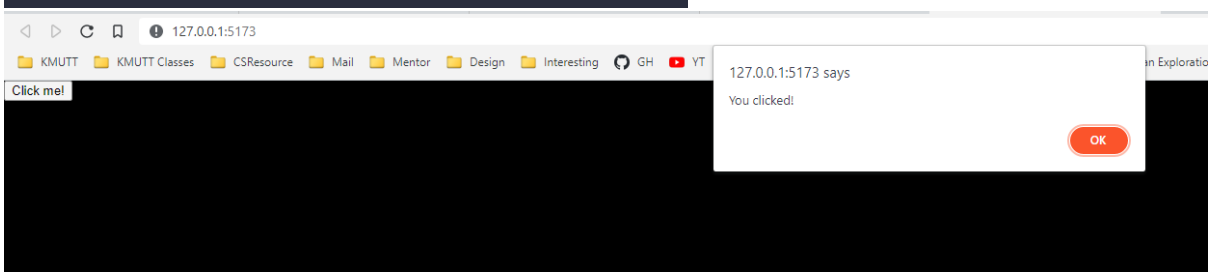
```
function App() {

  return (
    <div className="App">
      <button>Click me!</button>
    </div>
  );
}
```

You can pass function that handle clicking event to the button's onClick props

```
function App() {
  function handleClick() {
    alert("You clicked!");
  }

  return (
    <div className="App">
      <button onClick={handleClick}>Click me!</button>
    </div>
  );
}
```



NOTE!!!: Functions passed to event handlers must be passed, not called.

<button onClick={handleClick}> This is correct way to pass the function

<button onClick={handleClick()}> This is wrong!!!

2. You can read props in event handlers too!

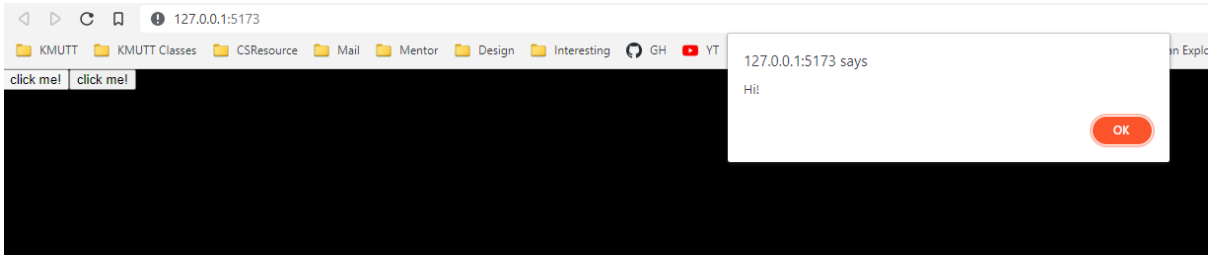
Because event handlers are declared inside of a component, they have access to the component's props. Here is a button that, when clicked, shows an alert with its message prop:

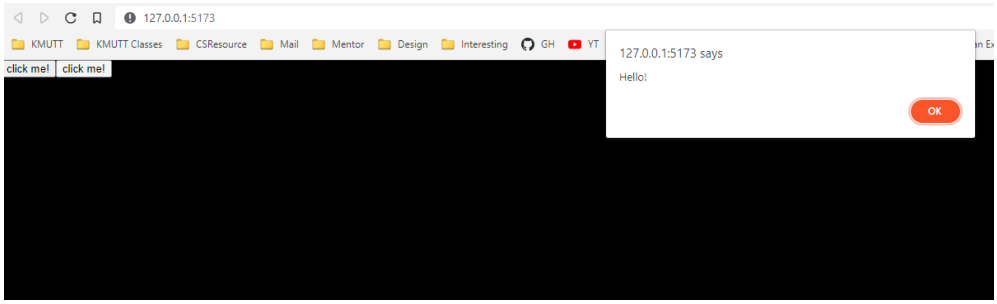
```
function App() {
  return (
    <div className="App">
      <AlertButton message={'Hi!'} />
      <AlertButton message={'Hello!'} />
    </div>
  );
}

function AlertButton({ message }) {

  function handleClick() {
    alert(message);
  }

  return (
    <button onClick={handleClick}>click me!</button>
  );
}
```





3. You can also pass the event handler as props!!

```
function App() {

  function handlePlay() {
    alert("Playing");
  }

  function handleStop() {
    alert("Stop!");
  }

  return (
    <div className="App">
      <AlertButton message={"play"} onClick={handlePlay} />
      <AlertButton message={"stop"} onClick={handleStop} />
    </div>
  );
}

function AlertButton({ message, onClick }) {
  return (
    <button onClick={onClick}>{message}</button>
  );
}
```

Now your AlertButton component going to act differently from each other because they don’t have the same event handler.

4. Because event handler also consider as props you can name it whatever you like.

```
  return (
    <div className="App">
      <AlertButton message={"play"} onAction={handlePlay} />
      <AlertButton message={"stop"} onAction={handleStop} />
    </div>
  );
}

function AlertButton({ message, onAction }) {
  return (
    <button onClick={onAction}>{message}</button>
  );
}
```

NOTE: the onClick at button still cannot be change because it is not your custom component!!

5. preventDefault()

In form element when you are hitting the submit button the default behavior of the <form> forces the page to reload. Sometimes you do not want that to happened.

event.preventDefault()

you must pass “event” object to the event handler to use it.

(More detail about preventDefault()
https://www.w3schools.com/jsref/event_preventdefault.asp)

```
function App() {

  function handleSubmit(event) {
    event.preventDefault();
  }

  return (
    <div className="App">
      <form onSubmit={handleSubmit}>
        <input type="text" />
        <button type="submit">Submit</button>
      </form>
    </div>
  );
}
```

Assignment 2:

Objective:

- Be able to understand and how to use event handling in React.

1. Create TypeMe component inside your components folder. Your component will render type “text” <input> tag inside it (wrapped with). When the text input is out of focus there should be a red paragraph appear “Please type in here!”

HINT: the event is called “Focus”

Form Controlled in React

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with setState().

We can combine the two by making the React state be the “single source of truth”. Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a “controlled component”.

Form Validation in React

Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This is called client-side form validation, and helps ensure data submitted matches the requirements set forth in the various form controls.

(https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation)

Example of Form Controlled in React

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function MyForm() {
  const [name, setName] = useState("");

  return (
    <form>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
    </form>
  )
}
```

In the picture above we add onChange to handle changing of user's input in the <input> element by calling the setName function which is your set state function. Your value of input is set to the the value of state. Now your form is using the value and setting value by using useState hook provided by React.

Your React state become "single source of truth" for your form.

NOTE: Most of your form element should be controlled using this way. That means if you have more <input> you also have to create more state and event handler for your <input>

Example: you have email input and username input

```
function MyForm() {
const [username, setUsername] = useState("");
const [email, setEmail] = useState("");

function handleUsernameChange(e) {
        setUsername(e.target.value);
}

function handleEmailChange(e) {
        setEmail(e.target.value);
}
Return (
<>
<input type="text" value={username} onChange={handleUsernameChange} />
<input type="email" value={email} onChange={handleEmailChange} />
</>
);
}
```

Example of Form Validation

```
<input type="text" id="country_code" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country
```

In HTML, the <input> element come with pattern attribute which you can pass Regular Expression to match with user's input if they don't match then it will alert user that they have to enter the input correctly.

In React, you can validate the data within your handleChange function which give you more control.

Assignment 3:

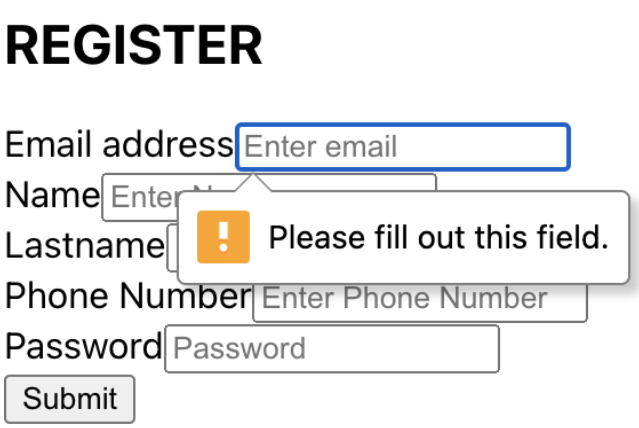
Objective:

- Be able to use form in React and understand what form controlled is.
- Be able to do form validation.

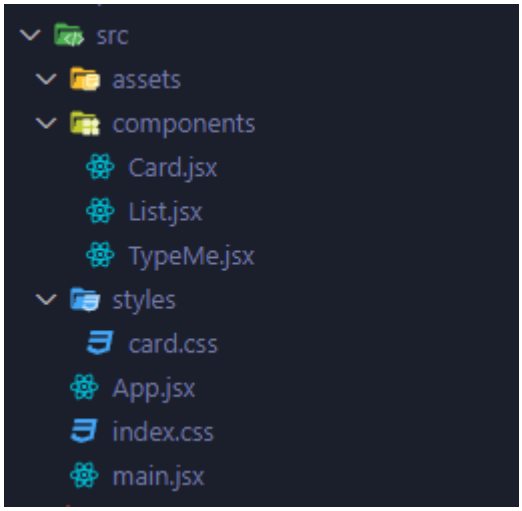
In your components folder create MyForm component. Your component should have <form> element that has email input, first name input, last name input, phone number input, password input and a submit button.

All field need to be filled. Your email, first name and last name must accept only text. Your phone should only accept 10 numbers of 0-9. Your password should not show text (use input type correctly). Your form should have onSubmit event handler which will alert user their information from the form.

If you click the submit button the page should not reload.



Your project structure might look like this:



P.S.: In your components folder, you should have MyForm.jsx too!

Finally: Upload your project to your github repository