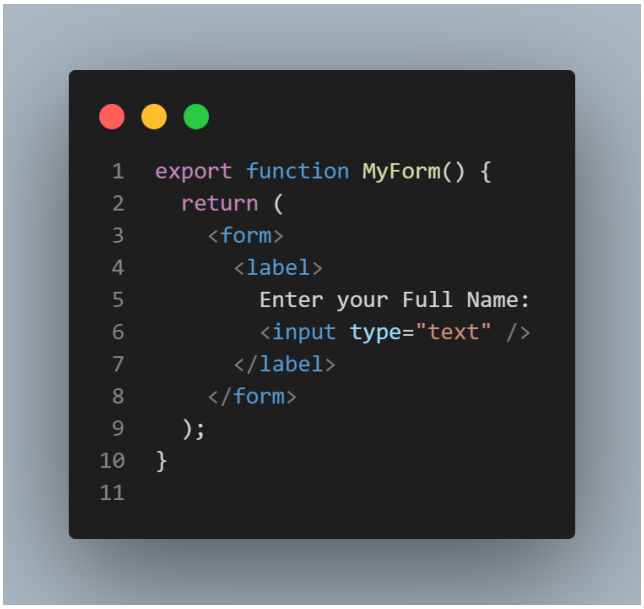
React Forms

Topics covered:

- How to add Forms in react?
  - Types of inputs
  - Adding a label to an input
  - To provide an initial point value for an input
- How to handle React forms?
  - Submitting form
  - Handling multiple inputs
- Control input with state variable
- RegEx in React
  - RegEx
  - Example

1. **Add Form in react component:**

   React allows you to add a form just like any other element:



```
export function MyForm() {
  return (
    <form>
      <label>
        Enter your Full Name:
        <input type="text" />
      </label>
    </form>
  );
}
```

   - The form will submit as expected, and the page will refresh.
   - However, with React, this is usually not what we want to happen.
   - In order to let React manage the form, we want to avoid this default behavior.

a. Type of inputs:

Render an input component in order to see the input. It will by default be a text input. For a checkbox, a radio button, or any other input type, you can pass type=" checkbox," type=" radio," or another input type.

```
export default function MyForm() {
  return (
    <div>
      <label>
        Text input: <input name="myInput" />
      </label>
      <hr />
      <label>
        Checkbox: <input type="checkbox" name="myCheckbox" />
      </label>
      <hr />
      <p>
        Radio buttons:
        <label>
          <input type="radio" name="myRadio" value="option1" />
          Option 1
        </label>
        <label>
          <input type="radio" name="myRadio" value="option2" />
          Option 2
        </label>
        <label>
          <input type="radio" name="myRadio" value="option3" />
          Option 3
        </label>
      </p>
    </div>
  );
}
```

Text input: Some initial value

Checkbox:

Radio buttons:
- Option 1
- Option 2
- Option 3

b. Adding a label to an input:

- Every <input> tag should be placed inside a <label> tag. This informs the browser that this label is linked to that input. The browser will immediately focus the input when the user clicks the label. It's also important for accessibility: when the user focuses on the related input, a screen reader will announce the label caption.
- If you can't nest <input> within a <label>, connect them by supplying the same ID to <input id> and <label htmlFor>. Use useId to generate such an ID to avoid conflicts between several instances of the same component.

```
import { useId } from "react";

export default function Form() {
  const ageInputId = useId();
  return (
    <div>
      <label>
        Your first name:
        <input name="firstName" />
      </label>
      <hr />
      <label htmlFor={ageInputId}>Your age:</label>
      <input id={ageInputId} name="age" type="number" />
    </div>
  );
}
```

Your first name:

Your age:

c. To provide an initial point value for an input:

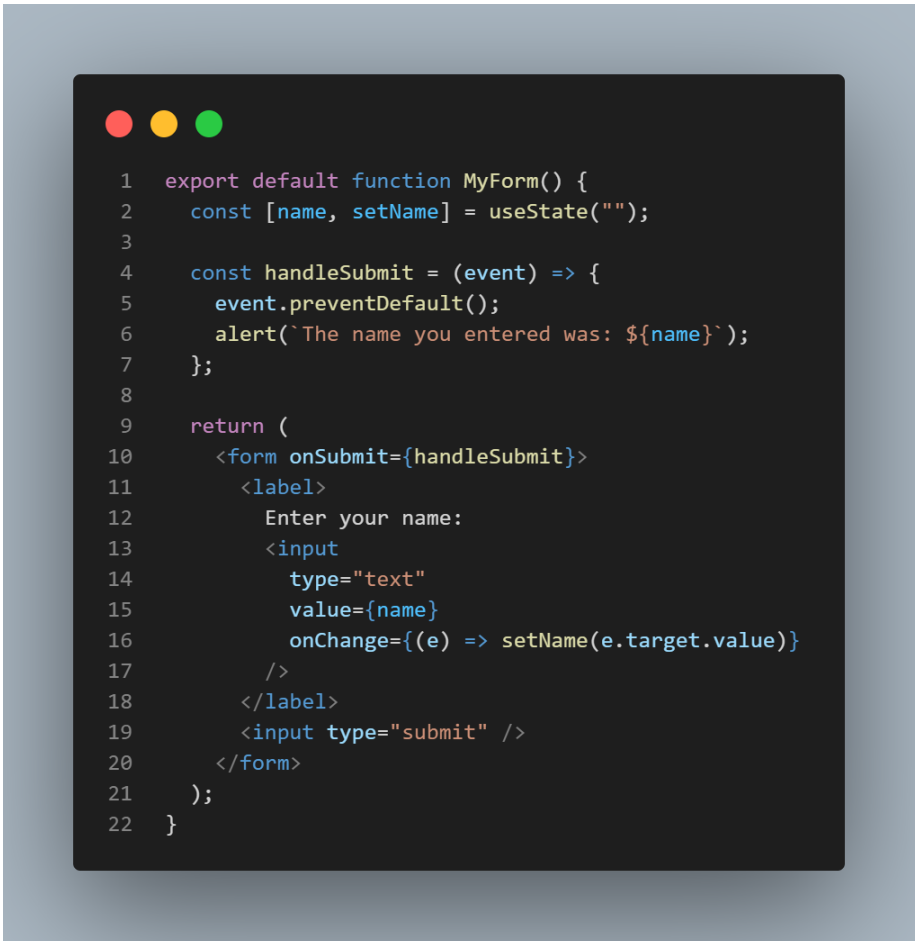
- Any input can have its initial value specified as an option. For text inputs, use it as the defaultValue string. Instead, checkboxes and radio buttons should use the defaultChecked boolean to specify the initial value.

```
export default function MyForm() {
  return (
    <div>
      <label>
        Text input: <input name="myInput" defaultValue="Some initial value" />
      </label>
      <hr />
      <label>
        Checkbox:
        <input type="checkbox" name="myCheckbox" defaultChecked={true} />
      </label>
      <hr />
      <p>
        Radio buttons:
        <label>
          <input type="radio" name="myRadio" value="option1" />
          Option 1
        </label>
        <label>
          <input
            type="radio"
            name="myRadio"
            value="option2"
            defaultChecked={true}
          />
          Option 2
        </label>
        <label>
          <input type="radio" name="myRadio" value="option3" />
          Option 3
        </label>
      </p>
    </div>
  );
}
```

2. **Handle React Forms:**
    - Handling forms refers to how you handle data when it changes or is submitted.
    - Form data is typically handled by the DOM in HTML.
    - In React, components handle form data frequently.
    - When data is handled by components, it is all kept in the component state.
    - Changes can be controlled by adding event handlers to the onChange attribute.
    - The useState Hook can be used to keep track of each input's value and offer a "single source of truth" for the entire application.

    a. **Submitting form:**

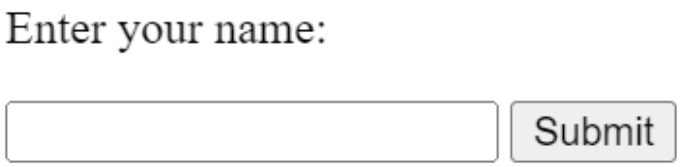
    You can control the submit action by adding an event handler to the form's onSubmit attribute:



```jsx
export default function MyForm() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`The name you entered was: ${name}`);
  };

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



b. Handling multiple inputs:

<input> should be enclosed in a <form> with a <button type="submit">. Your form's onSubmit event handler will be called. The browser will automatically send the form's data to the current URL and reload the page. Calling e.preventDefault() will allow you to alter that behavior. Use new FormData(e.target) to read the form data.

```jsx
export default function MyForm() {
  function handleSubmit(e) {
    // Prevent the browser from reloading the page
    e.preventDefault();

    // Read the form data
    const form = e.target;
    const formData = new FormData(form);

    // You can pass formData as a fetch body directly:
    fetch("/some-api", { method: form.method, body: formData });

    // Or you can work with it as a plain object:
    const formJson = Object.fromEntries(formData.entries());
    console.log(formJson);
  }

  return (
    <form method="post" onSubmit={handleSubmit}>
      <label>
        Text input: <input name="myInput" defaultValue="Some initial value" />
      </label>
      <hr />
      <label>
        Checkbox:{" "}
        <input type="checkbox" name="myCheckbox" defaultChecked={true} />
      </label>
      <hr />
      <p>
        Radio buttons:
        <label>
          <input type="radio" name="myRadio" value="option1" /> Option 1
        </label>
        <label>
          <input
            type="radio"
            name="myRadio"
            value="option2"
            defaultChecked={true}
          />{" "}
          Option 2
        </label>
        <label>
          <input type="radio" name="myRadio" value="option3" /> Option 3
        </label>
      </p>
      <hr />
      <button type="reset">Reset form</button>
      <button type="submit">Submit form</button>
    </form>
  );
}
```

Text input: Some initial value

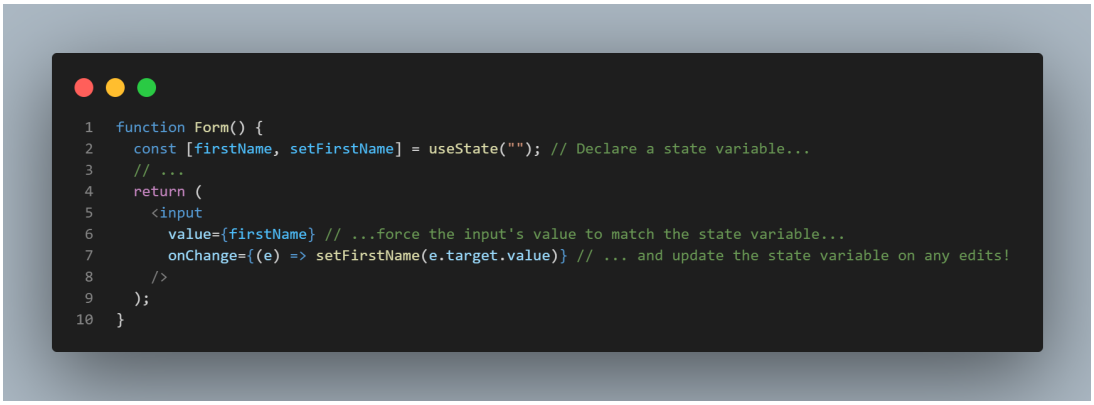
Checkbox: ☑

Radio buttons:

- ○ Option 1
- ◉ Option 2
- ○ Option 3

Reset form | Submit form

3. **Control input with state variable**
    - An uncontrolled input is <input />. Even if you specify an initial value, such as <input defaultValue="Initial text" />, your JSX will just specify the initial value. It has no say over what the value should be right now.
    - Pass the value prop to a controlled input to render it (or checked for checkboxes and radios). React will always force the input to have the value you gave. In most cases, you'll control an input by declaring a state variable:



```
function Form() {
  const [firstName, setFirstName] = useState(""); // Declare a state variable...
  // ...
  return (
    <input
      value={firstName} // ...force the input's value to match the state variable...
      onChange={(e) => setFirstName(e.target.value)} // ... and update the state variable on any edits!
    />
  );
}
```

If you needed state anyhow, for example, to re-render your UI on every edit, a controlled input makes sense.

```
function Form() {
  const [firstName, setFirstName] = useState('');
  return (
    <>
      <label>
        First name:
        <input value={firstName} onChange={e => setFirstName(e.target.value)} />
      </label>
      {firstName !== '' && <p>Your name is {firstName}.</p>}
      ...

```

It's also useful if you want to provide multiple ways to change the input state (such as by clicking a button):

```jsx
function Form() {
  // ...
  const [age, setAge] = useState('');
  const ageAsNumber = Number(age);
  return (
    <>
      <label>
        Age:
        <input
          value={age}
          onChange={e => setAge(e.target.value)}
          type="number"
        />
        <button onClick={() => setAge(ageAsNumber + 10)}>
          Add 10 years
        </button>
```

Controlled components should not be given undefined or null values. If the initial value must be empty (as in the firstName field below), set your state variable to an empty string (' ').

```jsx
import { useState } from "react";

export default function Form() {
  const [firstName, setFirstName] = useState("");
  const [age, setAge] = useState("20");
  const ageAsNumber = Number(age);
  return (
    <div>
      <label>
        First name:
        <input
          value={firstName}
          onChange={(e) => setFirstName(e.target.value)}
        />
      </label>
      <label>
        Age:
        <input
          value={age}
          onChange={(e) => setAge(e.target.value)}
          type="number"
        />
        <button onClick={() => setAge(ageAsNumber + 10)}>Add 10 years</button>
      </label>
      {firstName !== "" && <p>Your name is {firstName}.</p>}
      {ageAsNumber > 0 && <p>Your age is {ageAsNumber}.</p>}
    </div>
  );
}
```

Initial Output:

First name:

Age: 20 Add 10 years

Your age is 20.

Output after applying changes:

First name: Palak Rukhaya

Age: 20 Add 10 years

Your name is Palak Rukhaya.

Your age is 20.

4. **RegEx in React (Regular Expression):**
   a. **RegEx(or Regular Expressions):**
      - A string is examined to see if it contains the provided search pattern using a RegEx, or regular expression, which is a series of characters that defines a search pattern.
      - Additionally, it is used to validate strings that contain email, passwords, and other data.
   b. **Example:**
      In this example, we'll create a React application for authentication that asks the user for their email and password and determines whether or not they've been validated.

      For our application's email and password validation, we have Regex.js, which contains all the regular expressions.

Regex.js:

```
export const validEmail = new RegExp(
  "^[a-zA-Z0-9._:$!%-]+@[a-zA-Z0-9.-]+.[a-zA-Z]$"
);
export const validPassword = new RegExp("^(?=.*?[A-Za-z])(?=.*?[0-9]).{6,}$");

```

App.js

```
import React, { useState } from 'react';
import { validEmail, validPassword } from './regex.js';

const App = () => {
   const [email, setEmail] = useState('');
   const [password, setPassword] = useState('');
   const [emailErr, setEmailErr] = useState(false);
   const [pwdError, setPwdError] = useState(false);
   const validate = () => {
      if (!validEmail.test(email)) {
         setEmailErr(true);
      }
      if (!validPassword.test(password)) {
         setPwdError(true);
      }
   };
   return (
      <div>
         <input
            type="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
         />
         <input
            type="password"
            placeholder="Password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
         />
         <div>
            <button onClick={validate}>Validate
         </div>
         {emailErr && <p>Your email is invalid</p>}
         {pwdError && <p>Your password is invalid</p>}
      </div>
   );
};
export default App;
```

When the user clicks the Validate button in the above example, the email and password are validated and the result is displayed.

Output:
This will result in the following outcome: