前端網站開發人員證書課程
(二) 進階網絡程式設計--專業React.js應用

5. Work with Forms

Presented by Krystal Institute

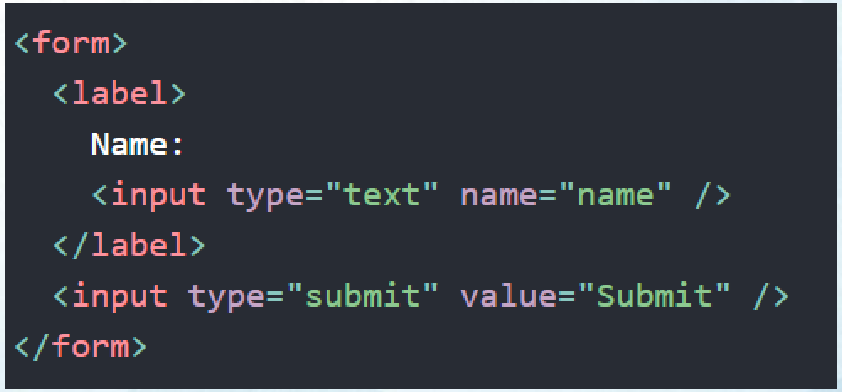
Lesson Outline

- Introduce forms that are used to get the inputs from the user and make the react application more dynamic and interactive
- Discuss how to create the forms and what are the types of form inputs in detail

5.1 Forms

5.1.1 Forms

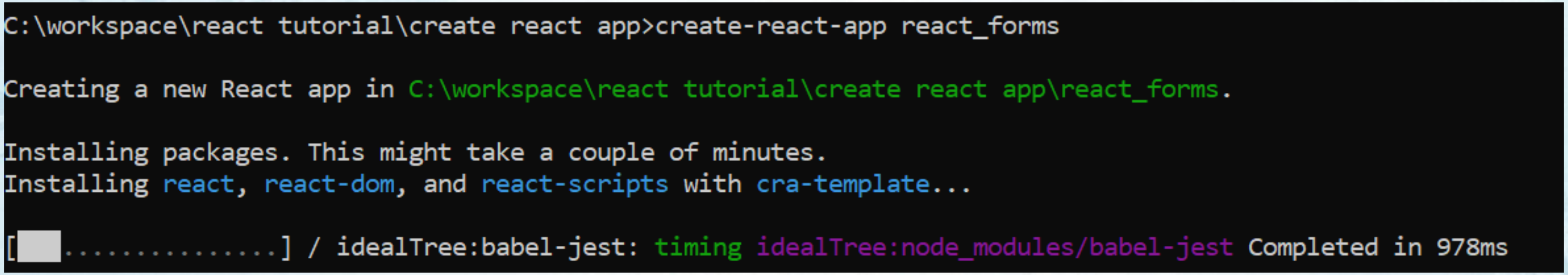
- Forms are an integral part of any modern web application.
- Forms allow the users to interact with the application and gather information from the users.
- Forms can perform many tasks that depend on your business requirements and logic such as user authentication, adding users, searching, filtering, booking, ordering, etc.
- A form can contain text fields, buttons, checkboxes, radio buttons, etc.
- HTML form elements work a bit differently from other DOM elements in React because form elements naturally keep some internal state.



```
<form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```

5.1.2 Create Forms

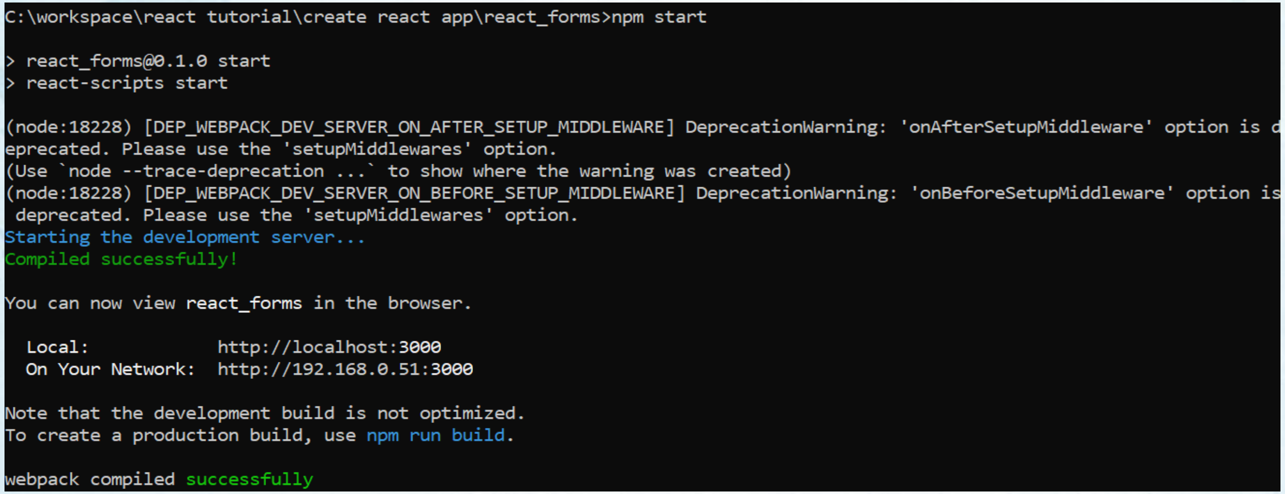
- We are going to build a simple contact form.
- Create a react app.



```
C:\workspace\react tutorial\create react app>create-react-app react_forms

Creating a new React app in C:\workspace\react tutorial\create react app\react_forms.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[██..............] / idealTree:babel-jest: timing idealTree:node_modules/babel-jest Completed in 978ms
```

5.1.2 Create Forms

- Get into the app folder and run the app.

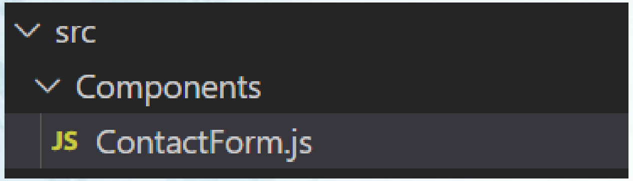


```
C:\workspace\react tutorial\create react app\react_forms>npm start

> react_forms@0.1.0 start
> react-scripts start

(node:18228) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is d
eprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:18228) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is
 deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view react_forms in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.51:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

- Open the app created in the Visual Studio Code.

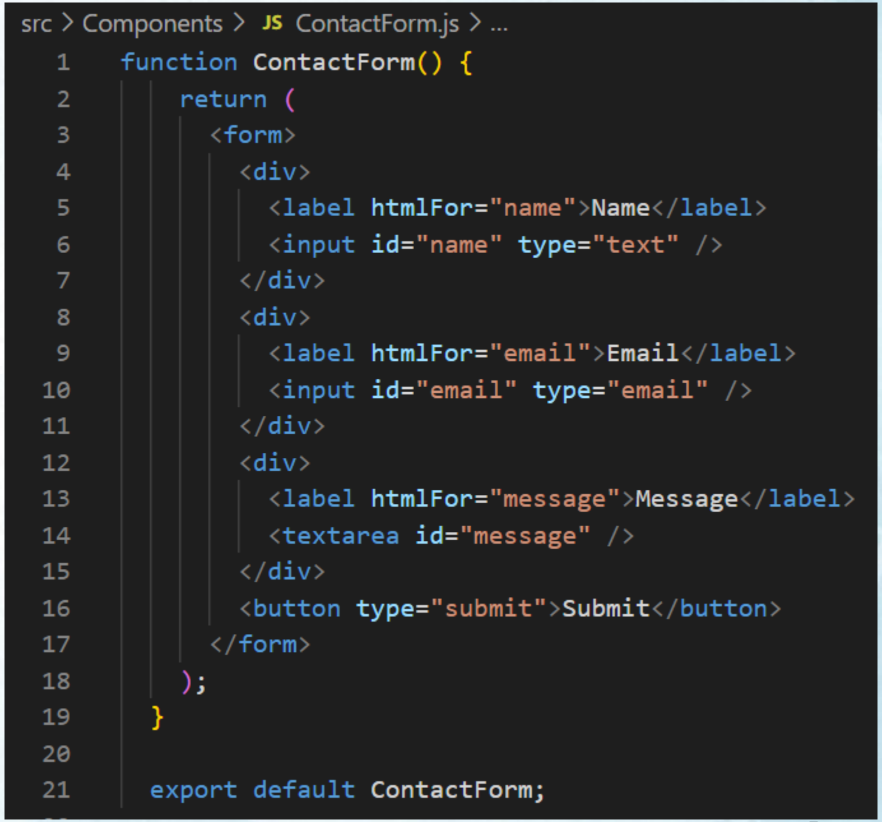
5.1.2 Create Forms

- Create a new folder inside src for adding components to it.



```
src
  Components
    ContactForm.js
```

- Create a function component "ContactForm" that renders the form with three basic input elements.

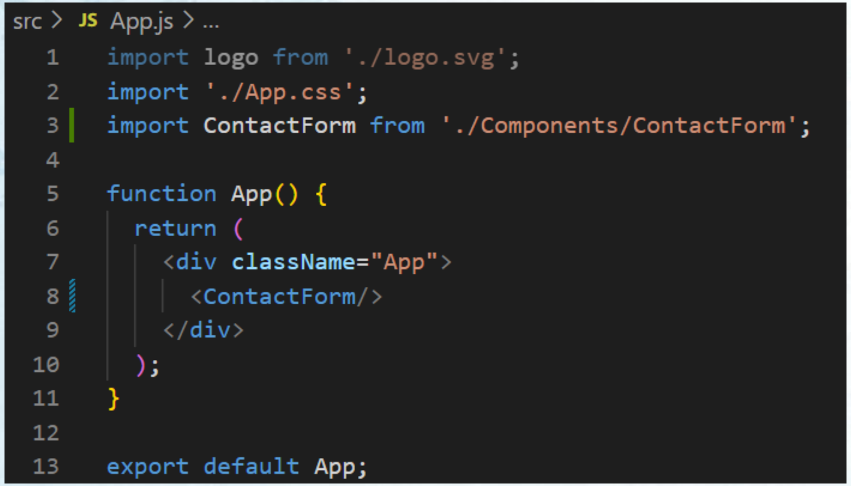


src > Components > ContactForm.js > ...

```jsx
function ContactForm() {
    return (
      <form>
        <div>
          <label htmlFor="name">Name</label>
          <input id="name" type="text" />
        </div>
        <div>
          <label htmlFor="email">Email</label>
          <input id="email" type="email" />
        </div>
        <div>
          <label htmlFor="message">Message</label>
          <textarea id="message" />
        </div>
        <button type="submit">Submit</button>
      </form>
    );
  }

  export default ContactForm;
```

5.1.2 Create Forms

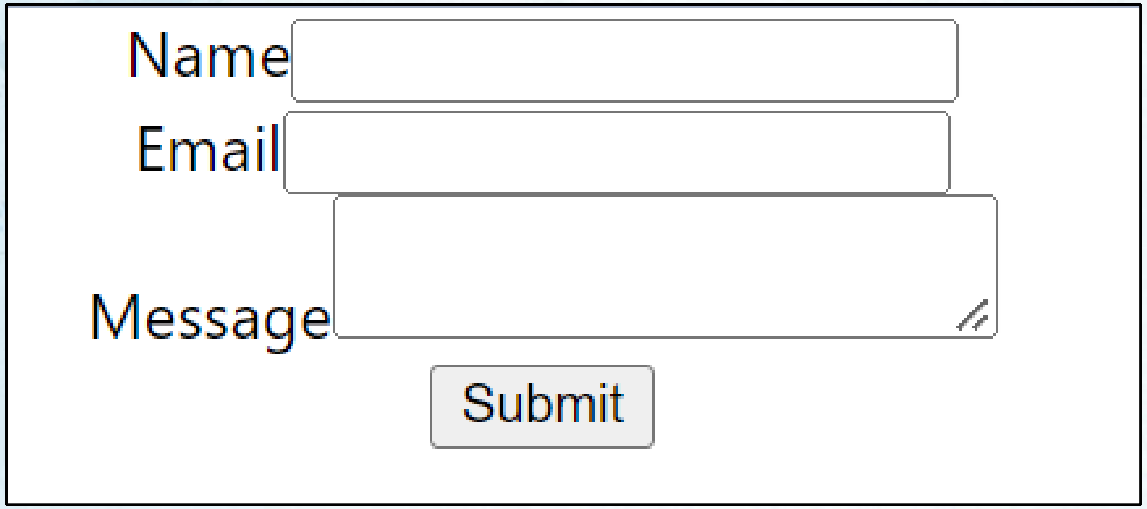
- Render the component “ContactForm” in the App component by replacing the default content inside it and make sure to import the “ContactForm”.


src > JS App.js > ...

```
import logo from './logo.svg';
import './App.css';
import ContactForm from './Components/ContactForm';

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

export default App;
```

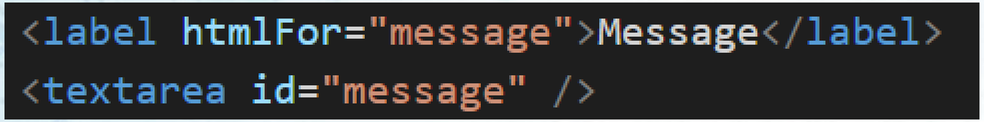
5.1.2 Create Forms

- View the output

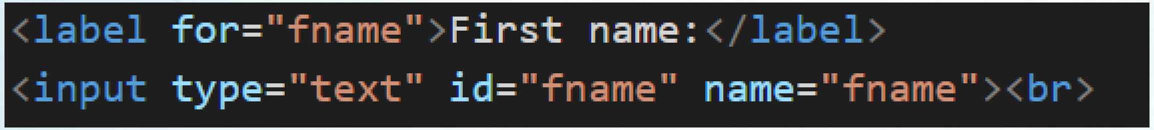


5.1.3 Forms VS HTML Forms

- React forms are probably similar to HTML forms.
- React forms also uses form tag, labels, and input elements as same as HTML forms.
- But in React each label has an “htmlfor” prop that matches the id on its corresponding input. Where in HTML the label attribute would be “for”.



```
<label htmlFor="message">Message</label>
<textarea id="message" />
```



```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname"><br>
```

5.2 Types of Form Inputs

5.2.1 Types of Form Inputs

- Inputs in React can be one of two types:
- Controlled Input.
- Uncontrolled Input.

Controlled:

- With a controlled input, YOU explicitly control the value that the input displays.
- You have to write code to respond to keypresses, store the current value somewhere, and pass that value back to the input to be displayed.
- It's a feedback loop with your code in the middle. It's more manual work to wire these up, but they offer the most control.

5.2.1 Types of Form Inputs

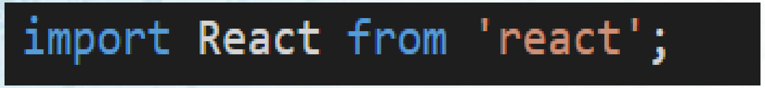
Uncontrolled:

- An uncontrolled input is the simpler of the two.
- It's the closest to a plain HTML input.React puts it on the page, and the browser keeps track of the rest.
- When you need to access the input's value, React provides a way to do that.
- Uncontrolled inputs require less code but make it harder to do certain things.

Let's look at these two styles in practice, applied to our contact form.

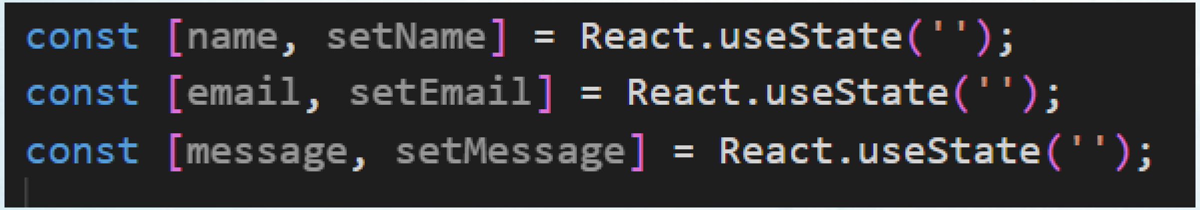
5.2.2 Controlled Inputs

- With a controlled input, you write the code to manage the value explicitly.
- You'll need to create a state to hold it, update that state when the value changes, and explicitly tell the input what value to display.
- Import React library.



```
import React from 'react';
```

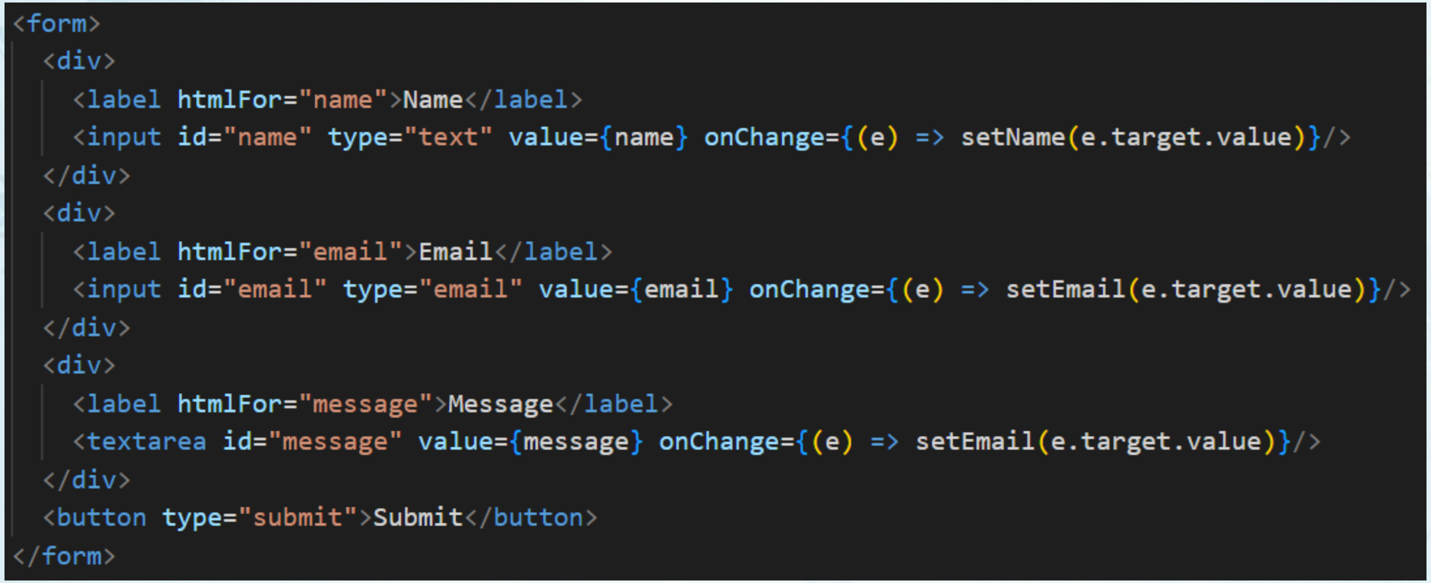
- Create a state to hold the inputs, and let the initial value be blank.



```
const [name, setName] = React.useState('');
const [email, setEmail] = React.useState('');
const [message, setMessage] = React.useState('');
```

5.2.2 Controlled Inputs

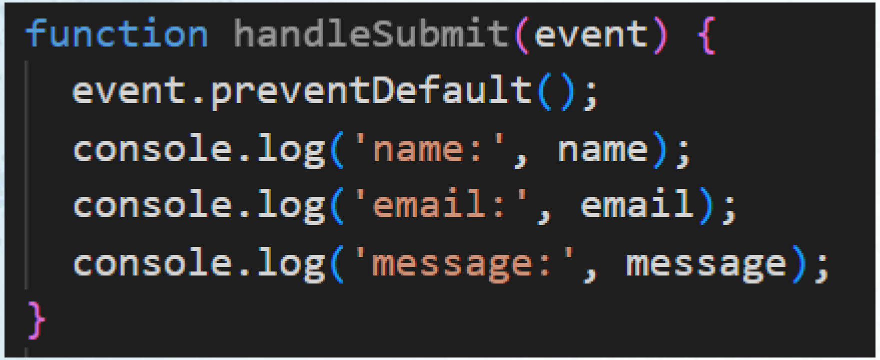
- Add the value attribute to the input elements and update the state on the change event of input values.



```
<form>
  <div>
    <label htmlFor="name">Name</label>
    <input id="name" type="text" value={name} onChange={(e) => setName(e.target.value)}/>
  </div>
  <div>
    <label htmlFor="email">Email</label>
    <input id="email" type="email" value={email} onChange={(e) => setEmail(e.target.value)}/>
  </div>
  <div>
    <label htmlFor="message">Message</label>
    <textarea id="message" value={message} onChange={(e) => setEmail(e.target.value)}/>
  </div>
  <button type="submit">Submit</button>
</form>
```

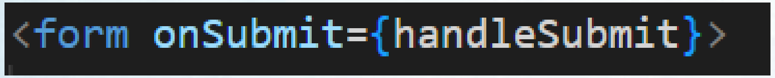
5.2.2 Controlled Inputs

- Create a function to handle the submit event, here we can just print the input value in the console on the submit of the form.



```
function handleSubmit(event) {
  event.preventDefault();
  console.log('name:', name);
  console.log('email:', email);
  console.log('message:', message);
}
```

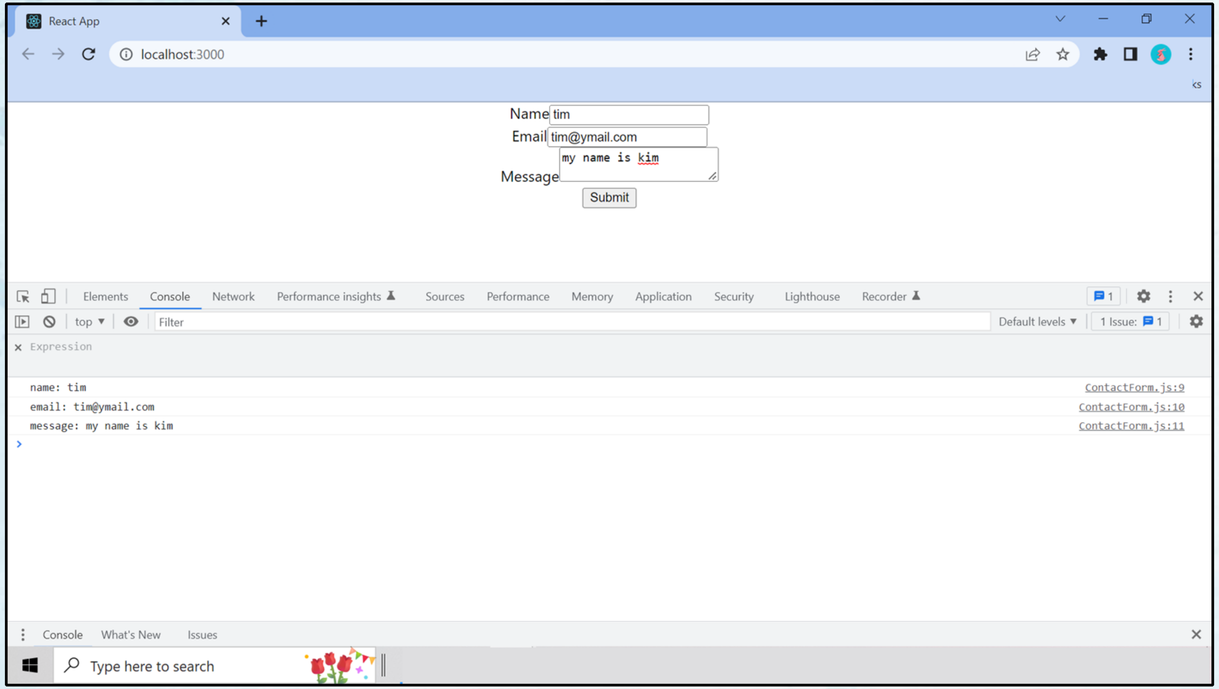
- Invoke the function on the submit event of the form.



```
<form onSubmit={handleSubmit}>
```

5.2.2 Controlled Inputs

- Validate the output.



Name tim
Email tim@ymail.com
Message my name is kim

```
name: tim                                    ContactForm.js:9
email: tim@ymail.com                         ContactForm.js:10
message: my name is kim                      ContactForm.js:11
```

5.2.3 Uncontrolled Inputs

- The uncontrolled input is similar to the traditional HTML form inputs.
- The DOM itself handles the form data.
- Here, the HTML elements maintain their own state that will be updated when the input value changes.
- If you do nothing beyond dropping an <input> in your render function, that input will be uncontrolled.
- No need to manually track it.
- But if we're not actively tracking the value… how can we tell what the value is?
- Here's where "refs" come in.

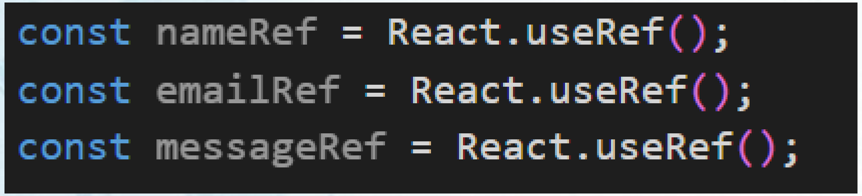
5.3 Ref

5.3.1 What is Ref?

- React takes your JSX and constructs the actual DOM, which the browser displays.
- Refs tie these two representations together, letting your React component get access to the DOM nodes that represent it.
- A ref holds a reference to a DOM node.
- The JSX is merely a description of the page to be created.
- So, to get the value from an uncontrolled input, you need a reference to it, which we get by assigning a ref prop. Then you can read out the value when the form is submitted.
- Uncontrolled inputs are the best choice when you only need to do something with the value at a specific time, such as when the form is submitted.

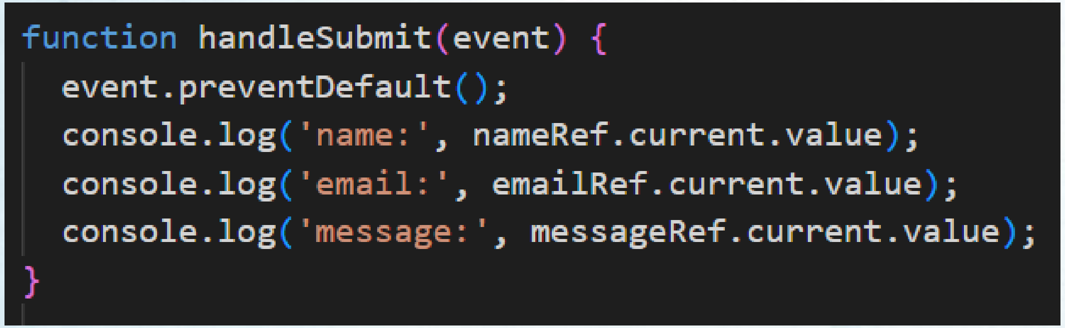
5.3.2 Add Refs to Our Contact form Inputs

- Create 3 refs with the useRef hook.



```
const nameRef = React.useRef();
const emailRef = React.useRef();
const messageRef = React.useRef();
```

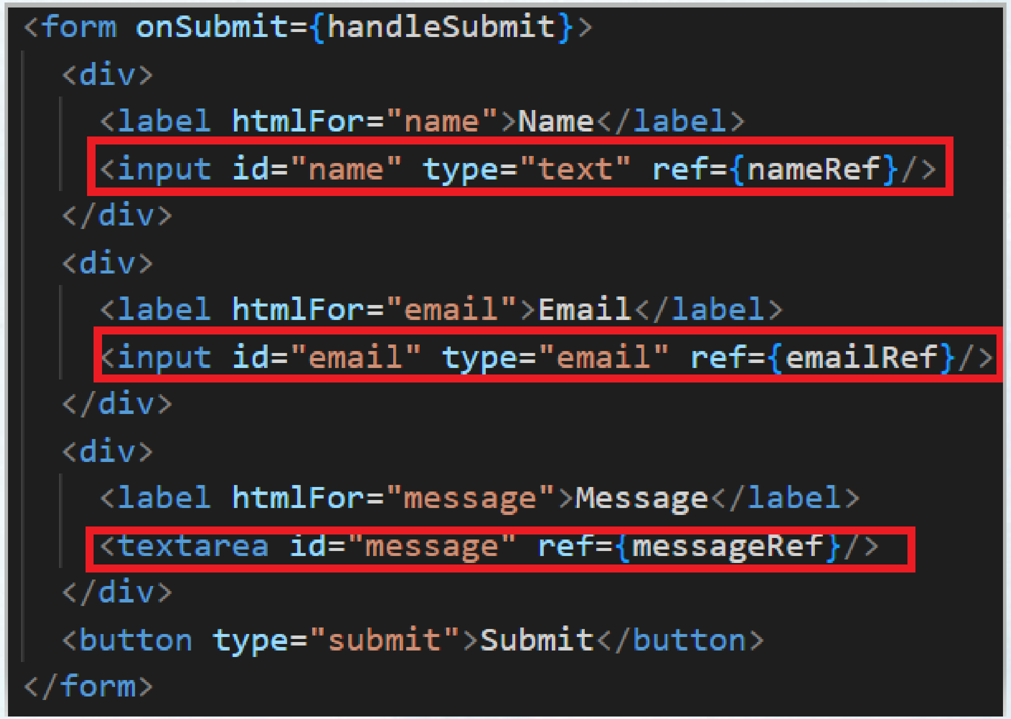
- Handle form submission, and print out the values.



```
function handleSubmit(event) {
  event.preventDefault();
  console.log('name:', nameRef.current.value);
  console.log('email:', emailRef.current.value);
  console.log('message:', messageRef.current.value);
}
```

5.3.2 Add Refs to Our Contact form Inputs

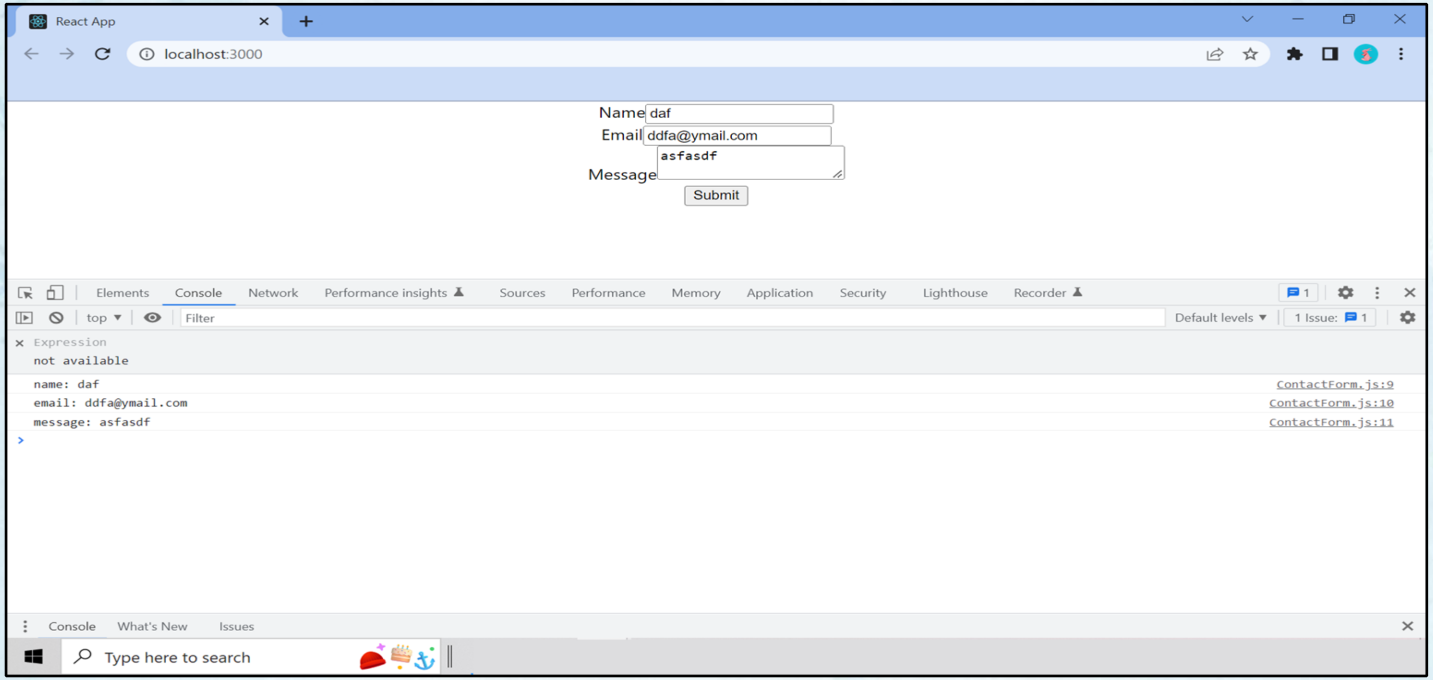
- Bound the refs to the inputs with the ref prop.



```
<form onSubmit={handleSubmit}>
  <div>
    <label htmlFor="name">Name</label>
    <input id="name" type="text" ref={nameRef}/>
  </div>
  <div>
    <label htmlFor="email">Email</label>
    <input id="email" type="email" ref={emailRef}/>
  </div>
  <div>
    <label htmlFor="message">Message</label>
    <textarea id="message" ref={messageRef}/>
  </div>
  <button type="submit">Submit</button>
</form>
```

5.3.2 Add Refs to Our Contact form Inputs

- Validate the output, the values should be printed on the submission of the form.



```
Expression
not available
name: daf                                   ContactForm.js:9
email: ddfa@ymail.com                       ContactForm.js:10
message: asfasdf                            ContactForm.js:11
```

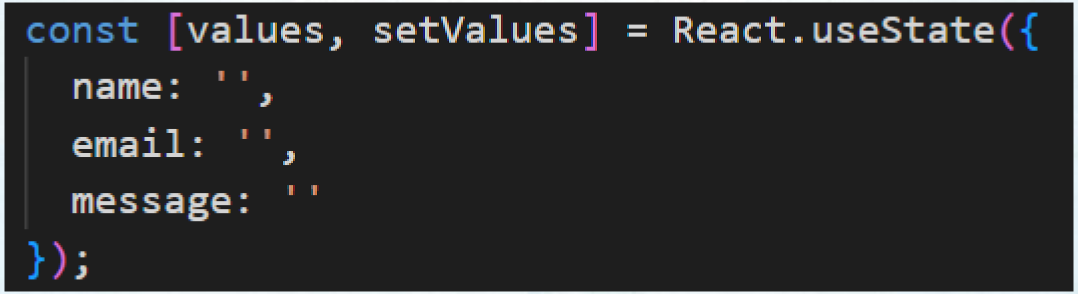
5.3.3 When and Why to Use Controlled Inputs

- Controlled inputs instantly validates the form on every keypress
- Controlled inputs are useful if you want to keep the Submit button disabled until everything is valid.
- Controlled inputs handle formatted input, like a credit card number field, or preventing certain characters from being typed.
- Controlled inputs Keep multiple inputs in sync with each other when they’re based on the same data.

5.4 Handle Multiple Inputs

5.4 Handle Multiple Inputs

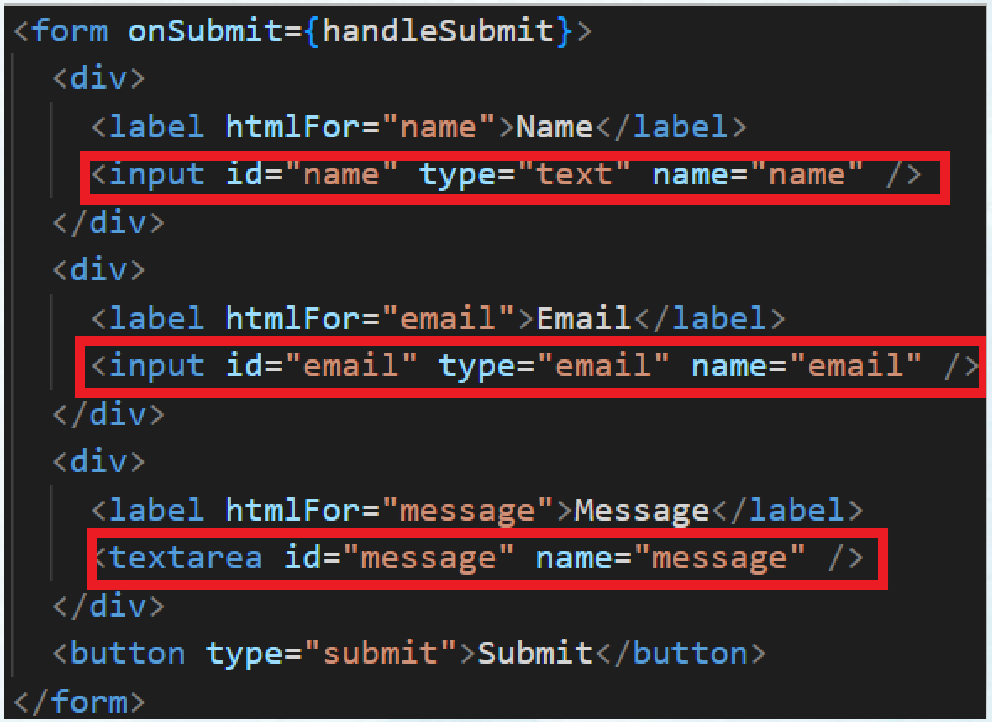
- Our contact form has three inputs: name, email and message. We have the following for every input:
  - A state to hold it.
  - A change handler function.
- Let's work around part of this problem by combining the inputs into one state object.
- Replace the ref with useState to handle multiple inputs as an object.



```
const [values, setValues] = React.useState({
  name: '',
  email: '',
  message: ''
});
```

5.4 Handle Multiple Inputs

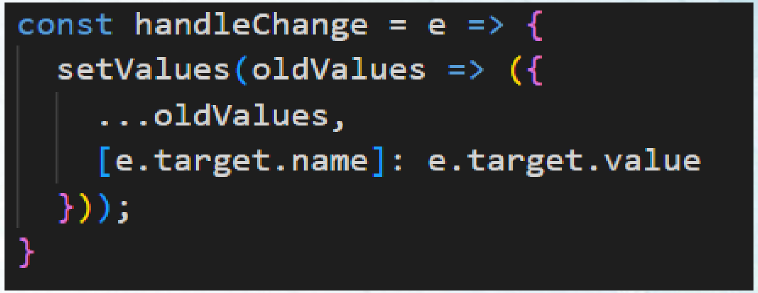
- Remove the "ref" attributes in the form input elements and add name attribute.



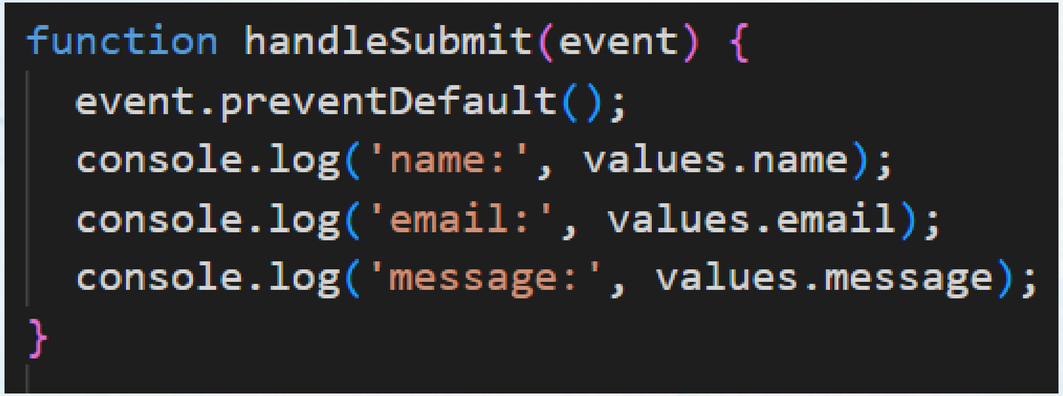
```
<form onSubmit={handleSubmit}>
  <div>
    <label htmlFor="name">Name</label>
    <input id="name" type="text" name="name" />
  </div>
  <div>
    <label htmlFor="email">Email</label>
    <input id="email" type="email" name="email" />
  </div>
  <div>
    <label htmlFor="message">Message</label>
    <textarea id="message" name="message" />
  </div>
  <button type="submit">Submit</button>
</form>
```

5.4 Handle Multiple Inputs

- Create a handle change event to update values on the change event.
- Make changes in the *handleSubmit* function.



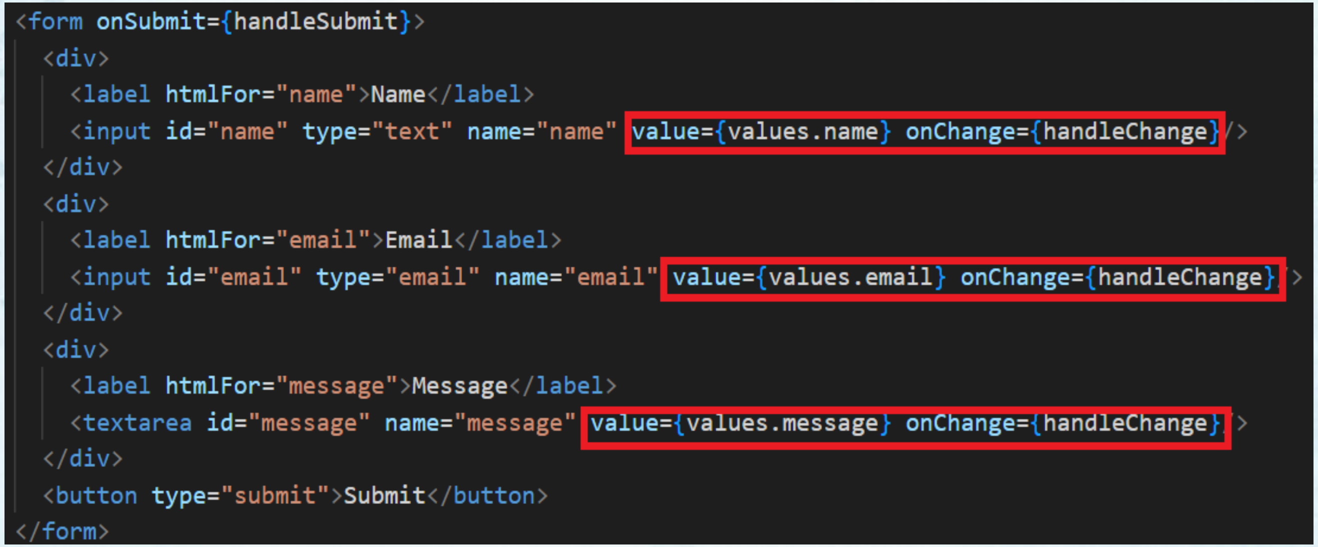
```javascript
const handleChange = e => {
  setValues(oldValues => ({
    ...oldValues,
    [e.target.name]: e.target.value
  }));
}
```



```javascript
function handleSubmit(event) {
  event.preventDefault();
  console.log('name:', values.name);
  console.log('email:', values.email);
  console.log('message:', values.message);
}
```

5.4 Handle Multiple Inputs

- Add value and *onchange* attribute to all the input elements in the form.



```
<form onSubmit={handleSubmit}>
  <div>
    <label htmlFor="name">Name</label>
    <input id="name" type="text" name="name" value={values.name} onChange={handleChange}/>
  </div>
  <div>
    <label htmlFor="email">Email</label>
    <input id="email" type="email" name="email" value={values.email} onChange={handleChange}/>
  </div>
  <div>
    <label htmlFor="message">Message</label>
    <textarea id="message" name="message" value={values.message} onChange={handleChange}/>
  </div>
  <button type="submit">Submit</button>
</form>
```

5.5 Controlled VS Uncontrolled Inputs

5.5 Controlled VS Uncontrolled Inputs

Controlled Inputs Re-render on Every Keypress

- Every time you press a key, React calls the function in the *onChange* prop, which sets the state. Setting the state causes the component and its children to re-render.
- This is mostly fine. Renders are fast. For small-to-medium forms you probably won't even notice. And it's not that rendering a piddly little *input* is slow… but it can be a problem in aggregate.
- As the number of inputs grows – or if your form has child components that are expensive to render – keypresses might start to feel perceptibly laggy. This threshold is even lower on mobile devices.
- It can become a problem of death-by-a-thousand-cuts.

5.5 Controlled VS Uncontrolled Inputs

Uncontrolled Inputs Don’t Re-render

- A big point in favor of using uncontrolled inputs is that the browser takes care of the whole thing.
- You don’t need to update state, which means you don’t need to re-render. Every keypress bypasses React and goes straight to the browser.
- For Example: Typing the letter 'a' into a form with 300 inputs will re-render exactly zero times, which means React can pretty much sit back and do nothing. Doing nothing is very performant.

5.6 Form Labels

5.6 Form Labels

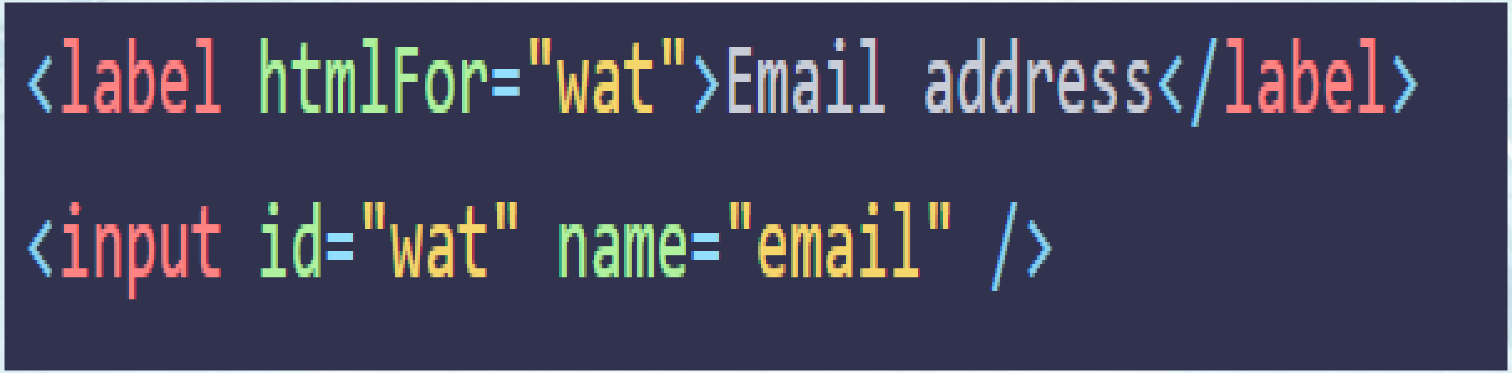
Accessible Form Labels

- Every input may have a label but not mandatory.
- Label-less inputs make trouble for screen readers, which makes trouble for humans… and placeholder text unfortunately doesn't cut it.
- The two ways to do labels are:
  - Label Next to Input. (2 sibling Element)
  - Input Inside Label.

5.6 Form Labels

<u>Label Next to Input:</u>

- Give the input an id and the label an htmlFor that matches, and put the elements side-by-side. Order doesn't matter, as long as the identifiers match up.

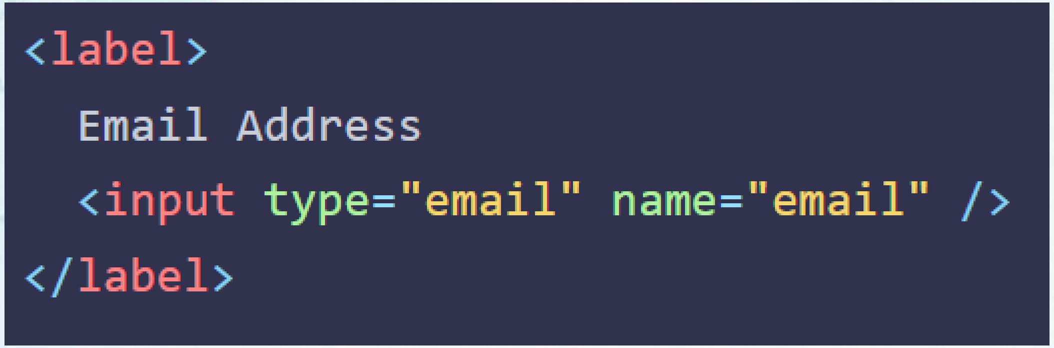


```
<label htmlFor="wat">Email address</label>

<input id="wat" name="email" />
```

5.6 Form Labels

Input Inside Label

- If you wrap the *input* in a *label*, you don't need the *id* and the *htmlFor*. You'll want a way to refer to the input though, so give it an *id* or a *name*.



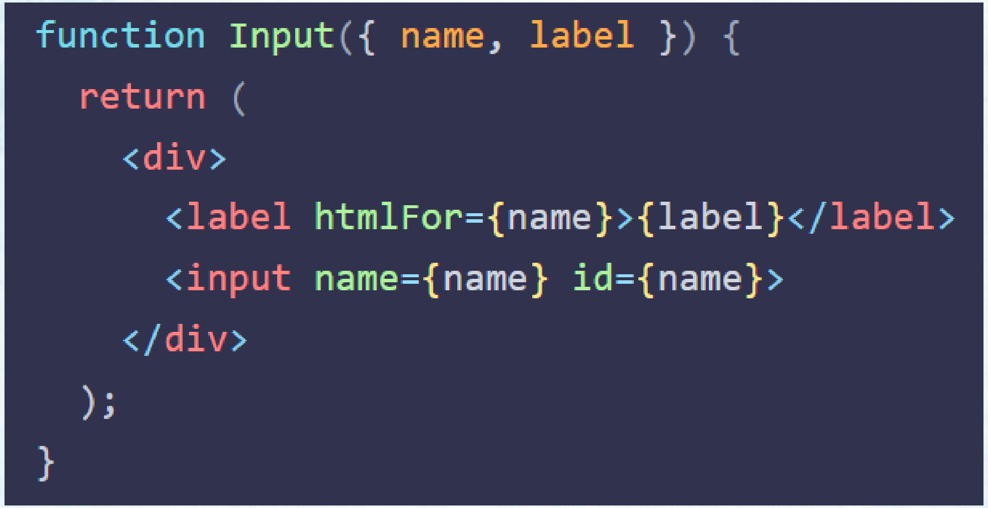
```
<label>
  Email Address
  <input type="email" name="email" />
</label>
```

- If you need more control over the style of the text, you can wrap it in a *span*.

5.6 Form Labels

Reduce Form Boilerplate with Small Components

- You can easily move the label and input element to a component.



```
function Input({ name, label }) {
  return (
    <div>
      <label htmlFor={name}>{label}</label>
      <input name={name} id={name}>
    </div>
  );
}
```

5.6 Form Labels

Reduce Form Boilerplate with Small Components

- Now every input is simple again.

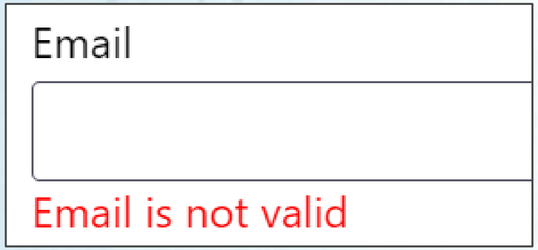
```
<Input name="email" label="Email Address"/>
```

- And if you’re using uncontrolled inputs, you can still use the trick of reading the values off the form, no refs or state required.

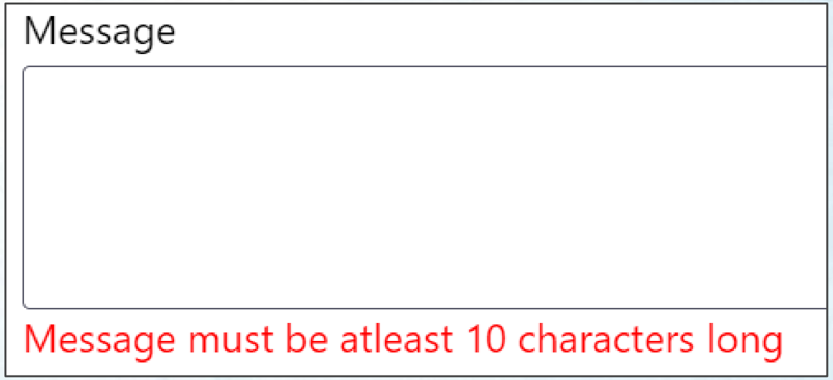
5.7 React Form Validation

5.7 React Form Validation

- Form validation in React allows an error message to be displayed if the user has not correctly filled out the form with the expected type of input.
- There are several ways to validate forms in React, creating a validator function with validation rules is one among them.
- Let's add validation to email and message inputs.


Email

Email is not valid

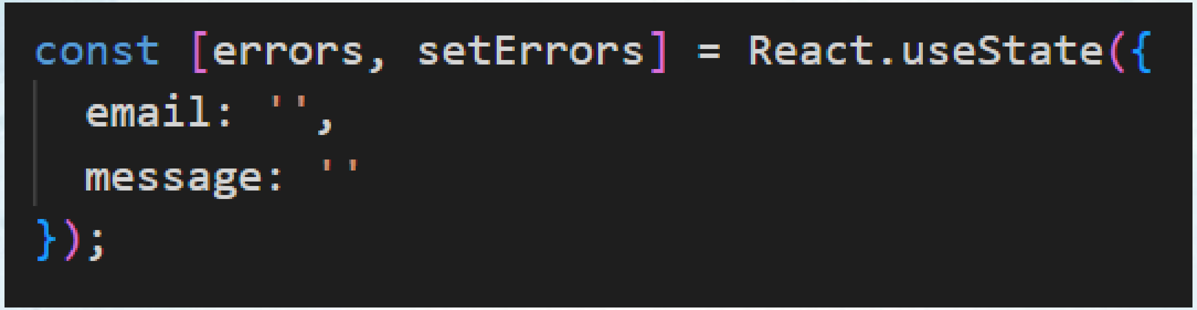

Message

Message must be atleast 10 characters long

5.7 React Form Validation

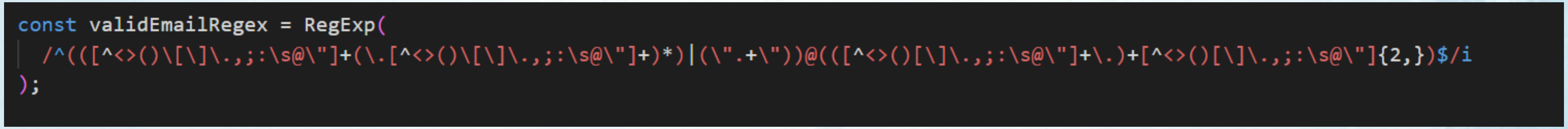
Creating Form validation

- Create a state that handles multiple errors.



```
const [errors, setErrors] = React.useState({
  email: '',
  message: ''
});
```

- Create a Regular expression to validate email.

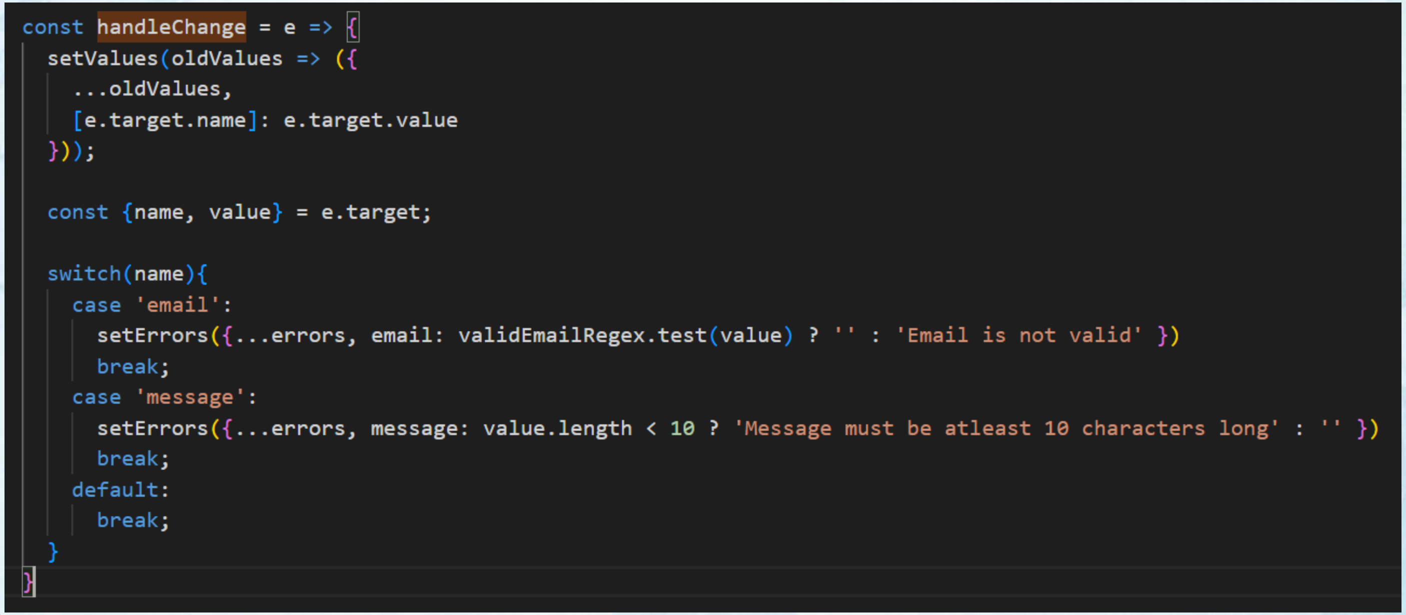


```
const validEmailRegex = RegExp(
  /^(([^<>()\[\]\.,;:\s@\"]+(\.[^<>()\[\]\.,;:\s@\"]+)*)|(\".+\"))@(([^<>()[\]\.,;:\s@\"]+\.)+[^<>()[\]\.,;:\s@\"]{2,})$/i
);
```

5.7 React Form Validation

Creating Form validation

- Inside the handle change function, add a switch case to validate the email and message input fields and display an error message if the condition fails.

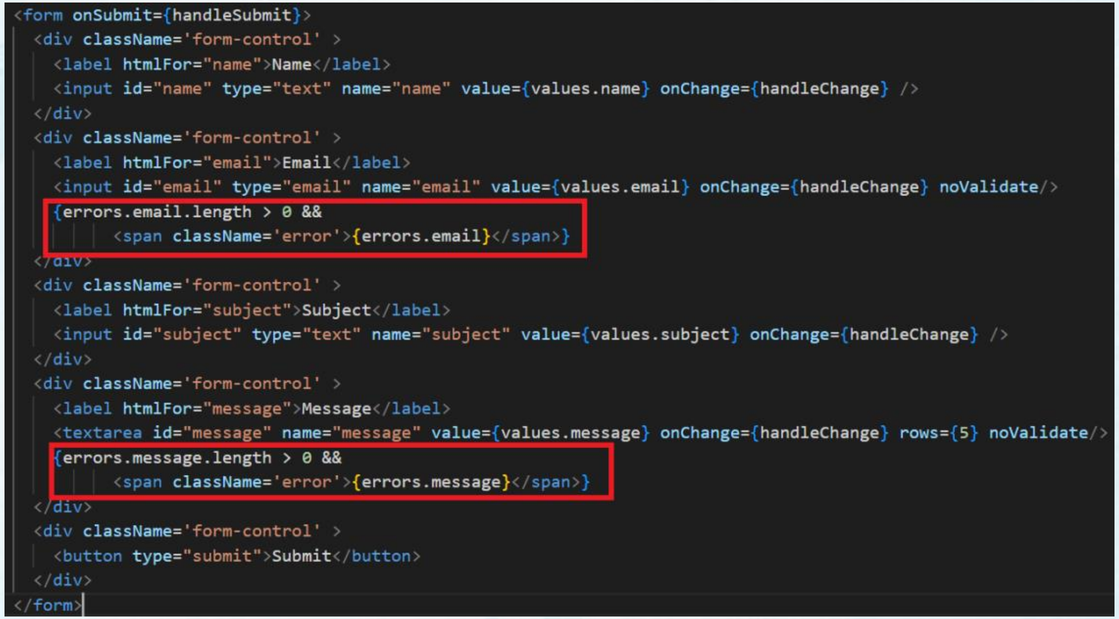


```
const handleChange = e => {
  setValues(oldValues => ({
    ...oldValues,
    [e.target.name]: e.target.value
  }));

  const {name, value} = e.target;

  switch(name){
    case 'email':
      setErrors({...errors, email: validEmailRegex.test(value) ? '' : 'Email is not valid' })
      break;
    case 'message':
      setErrors({...errors, message: value.length < 10 ? 'Message must be atleast 10 characters long' : '' })
      break;
    default:
      break;
  }
}
```

5.7 React Form Validation

Creating Form validation

- Render the error message below the input fields and add "*noValidate*" attribute to the respective input elements.



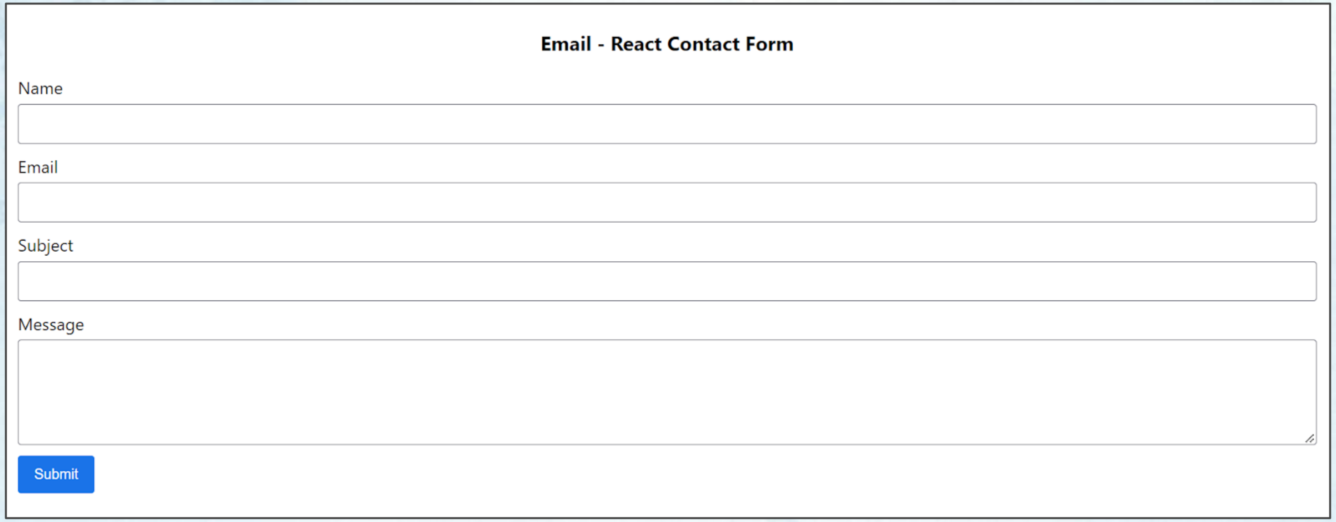
```
<form onSubmit={handleSubmit}>
  <div className='form-control' >
    <label htmlFor="name">Name</label>
    <input id="name" type="text" name="name" value={values.name} onChange={handleChange} />
  </div>
  <div className='form-control' >
    <label htmlFor="email">Email</label>
    <input id="email" type="email" name="email" value={values.email} onChange={handleChange} noValidate/>
    {errors.email.length > 0 &&
          <span className='error'>{errors.email}</span>}
  </div>
  <div className='form-control' >
    <label htmlFor="subject">Subject</label>
    <input id="subject" type="text" name="subject" value={values.subject} onChange={handleChange} />
  </div>
  <div className='form-control' >
    <label htmlFor="message">Message</label>
    <textarea id="message" name="message" value={values.message} onChange={handleChange} rows={5} noValidate/>
    {errors.message.length > 0 &&
          <span className='error'>{errors.message}</span>}
  </div>
  <div className='form-control' >
    <button type="submit">Submit</button>
  </div>
</form>
```

5.8 Assignment

5.8 Assignment

Outline:

Take your react contact form to the next level, make the form attractive by adding CSS and/or bootstrap and add extra fields and control them using multiple inputs.



Email - React Contact Form

Name

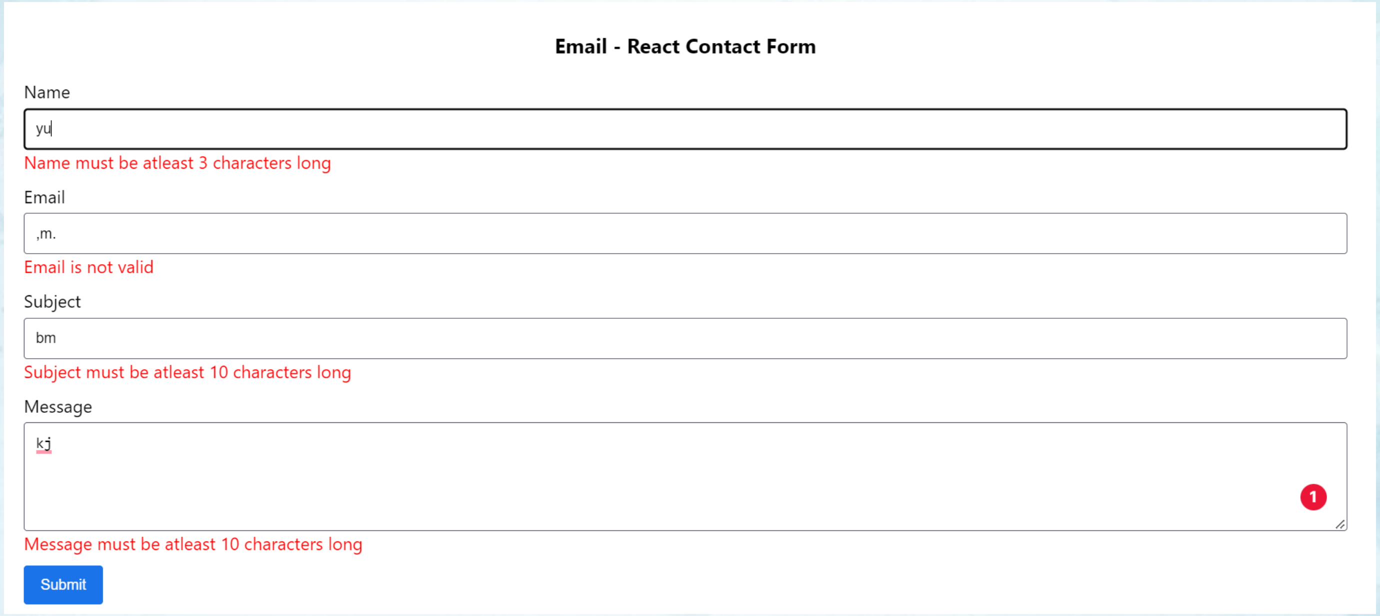
Email

Subject

Message

5.8 Assignment

Make sure to validate Name and Subject



Email - React Contact Form

Name

yu

Name must be atleast 3 characters long

Email

,m.

Email is not valid

Subject

bm

Subject must be atleast 10 characters long

Message

kj

Message must be atleast 10 characters long