FORMS AND FORMS VALIDATION

Handling user input the right way

Controlled Components and Uncontrolled components

- React forms present a unique challenge because you can either allow the browser to handle most of the form elements and collect data through React change events, or you can use React to fully control the element by setting and updating the input value directly.
  - The first approach is called an uncontrolled component because React is not setting the value.
  - The second approach is called a controlled component because React is actively updating the input.

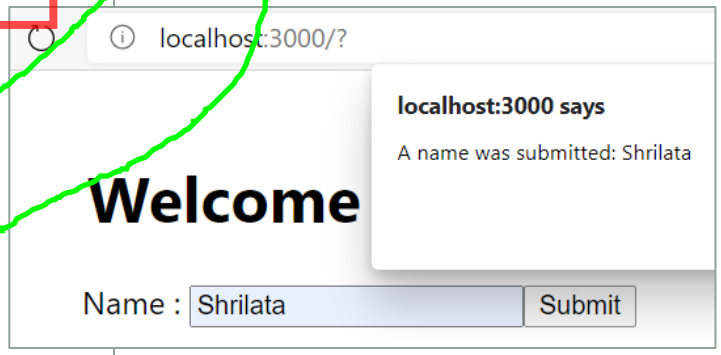
- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
  - When the data is handled by the components, all the data is stored in the component state.

Controlled Inputs in class components

- An input is said to be “controlled” when React is responsible for maintaining and setting its state.
  - The state is kept in sync with the input’s value, meaning that changing the input will update the state, and updating the state will change the input.

```
class ControlledInput extends React.Component {
   state = { name: '' };

    handleInput = (event) => {
      this.setState({name: event.target.value });
    }
    handleSubmit = (event) => {
        alert('A name was submitted: ' + this.state.name);
        event.preventDefault();
      }
    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          Name : <input value={this.state.name}
                        onChange={this.handleInput} />
          <input type="submit" value="Submit" />
        </form>
      );
    }
  }
```



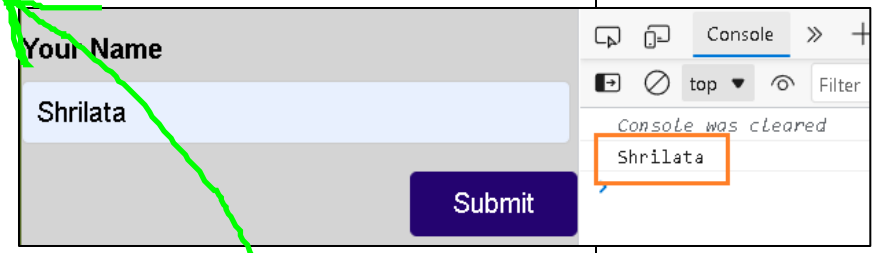
Controlled Inputs using React hooks (functional components)

```
import React, {useState} from 'react'

const SimpleInput = (props) => {
    const [inputName, setInputName] = useState('')

    const inputNameHandler = (event) => {
        setInputName(event.target.value)
    }

    const formSubmitHandler = event => {
        event.preventDefault();
        console.log(inputName)
    }
    return (
      <form onSubmit={formSubmitHandler}>
        <div className='form-control'>
          <label htmlFor='name'>Your Name</label>
          <input type='text' id='name' onChange={inputNameHandler}/>
        </div>
        <div className="form-actions">
          <button>Submit</button>
        </div>
      </form>
    );
  };
  export default SimpleInput;
```



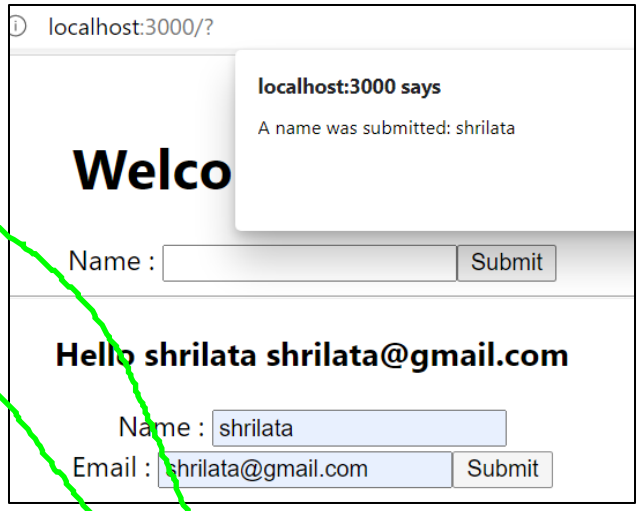
Controlled Inputs

- Controlled inputs open up the following possibilities:
  - **instant input validation**: we can give the user instant feedback without having to wait for them to submit the form (e.g. if their password is not complex enough)
  - **instant input formatting**: we can add proper separators to currency inputs, or grouping to phone numbers on the fly
  - **conditionally disable form submission**: we can enable the submit button after certain criteria are met (e.g. the user consented to the terms and conditions)
  - **dynamically generate new inputs**: we can add additional inputs to a form based on the user's previous input (e.g. adding details of additional people on a hotel booking)

Handling Multiple Form Inputs

```
class ControlledLoginForm extends React.Component {
   state = {
    username: '',
    email: ''
  };
    handleInput = (event) => {
      let name = event.target.name;
      let val = event.target.value;
      this.setState({[name]: val});
      console.log(this.state)
    }
    handleSubmit = (event) => {
        alert('A name was submitted: '
              + this.state.username);
        event.preventDefault();
      }
    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <h3>Hello {this.state.username} {this.state.email}</h3>
          Name : <input  name="username" onChange={this.handleInput} /><br />
          Email : <input  name="email" onChange={this.handleInput} />
          <input type="submit" value="Submit" />
        </form>
      ); }
  }
```

- {username: 'shrilata'} //where username is event.target.name which is field name ie username
- {email: 'Shrilata@gmail.com'} //where email is event.target.name which is field name ie email



One more example

Name:

Shrilata

Observation:

Good fabric

Desired color:

Green

T-shirt Size:Small ○ Medium ○ Large ◉ XL ○ XXL ○ 3XL ○

Submit

```
MultipleInputFields.js:16
▶ {name: 'Shrilata', onservation: '', colo
  r: '', size: '', observation: 'Good fa'}
MultipleInputFields.js:16
▶ {name: 'Shrilata', onservation: '', colo
  r: '', size: '', observation: 'Good fab'}
MultipleInputFields.js:16
▶ {name: 'Shrilata', onservation: '', colo
  r: '', size: '', observation: 'Good fab
  r'}
MultipleInputFields.js:16
▶ {name: 'Shrilata', onservation: '', colo
  r: '', size: '', observation: 'Good fabr
  i'}
MultipleInputFields.js:16
▶ {name: 'Shrilata', onservation: '', colo
  r: '', size: '', observation: 'Good fabri
  c'}
MultipleInputFields.js:16
▶ {name: 'Shrilata', onservation: '', colo
  r: 'green', size: '', observation: 'Good
  fabric'}
from submit        MultipleInputFields.js:21
▶ {name: 'Shrilata', onservation: '', colo
  r: 'green', size: 'LARGE', observation:
  'Good fabric'}
>
```

```
class MultipleInputFields extends Component {

    state = {
        name:'',
        onservation:'',
        color:'',
        size:'',
    }
    handleChanges = (event) => {
         let name = event.target.name;
         let val = event.target.value;
         this.setState({[name]: val});
         console.log(this.state)
    };
    submitFormHandler = (event) => {
        event.preventDefault();
        console.log("from submit ", this.state);
    };
    render(){
      const colors = ['Blue', 'Red', 'Green', 'Yellow'];
      const sizes = ['Small', 'Medium', 'Large', 'XL', 'XXL', '3XL'];
```

```
return (
    <form onSubmit={this.submitFormHandler}>
        <div className='form-control'>
          <label>Name:</label>
          <input name="name" type="text" value={this.state.name} onChange={this.handleChanges} />
          </div>
          <div className='form-control'>
            <label>Observation:</label>
            <textarea name="observation" value={this.state.observation} onChange={this.handleChanges} />
           </div>
           <div className='form-control'>
          <label>Desired color:</label>
          <select name="color" value={this.state.color} onChange={this.handleChanges}>
            {colors.map((color, i) => <option key={i} value={color.toLowerCase()}>{color}</option>)}
          </select>
          </div>
          <div >
          <label>T-shirt Size:</label>
          {sizes.map((size, i) =>
            <label key={i}> {size}
              <input
                name="size"  value={size.toUpperCase()}  checked={this.state.size === size.toUpperCase()}
                onChange={this.handleChanges}    type="radio" />
            </label>
          )}
        </div>
        <div className="form-actions">
        <button type="submit">Submit</button>
        </div>
    </form>
  )}
}
export default MultipleInputFields;
```

Controlled components : Summary

- A controlled component is bound to a value, and its adjustments will be handled in code by using event-based callbacks.
  - Here, the input form variable is handled by the react itself rather than the DOM.
  - In this case, the mutable state is maintained in the state property and modified using setState().
- Controlled components have functions which regulate the data that occurs at each on Change event.
  - This data is subsequently saved in the setState() method and updated. It helps components manage the elements and data of the form easier.
-
- You can use the controlled component when you create:
  - Forms validation so that when you type, you always have to know the input value to verify whether it is true or not!
  - Disable submission icon, except for valid data in all fields
  - If you have a format such as the input for a credit card

Validation

```
class ControlledInputValidation1 extends React.Component {
   state = { age: '' };

    handleInput = (event) => {
      let nam = event.target.name;
      let val = event.target.value;
      if (nam === "age") {
        if (!Number(val))
          alert("Age must be a number");
      }
      this.setState({[nam]: val});
    }
    handleSubmit = (event) => {
        alert('A age was submitted: ' + this.state.age);
        event.preventDefault();
      }
    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          Age : <input name="age" value={this.state.age} onChange={this.handleInput}
/>
          <input type="submit" value="Submit" />
        </form>
      );
    }
  }
```

Age : 111 Submit

Uncontrolled Inputs

- “uncontrolled” form inputs: React doesn’t track the input’s state.
  - HTML input elements naturally keep track of their own state as part of the DOM, and so when the form is submitted we have to read the values from the DOM elements themselves.

- “uncontrolled” form inputs: React doesn’t track the input’s state.
  - If the DOM handles the data, then the form is uncontrolled, and if the state of the form component manages the data, then the form is said to be controlled
  - Uncontrolled components are inputs that do not have a value property. In opposite to controlled components, it is the application's responsibility to keep the component state and the input value in sync.
  - In order to do this, React allows us to create a “**ref**” (reference) to associate with an element, giving access to the underlying DOM node

Uncontrolled Inputs

- In **uncontrolled** components form data is being handled by DOM itself.
- For example here we can reference form values by name
- This is quick and dirty way of handling forms. It is mostly useful for simple forms or when you are *just learning React*.

- HTML input elements keep track of their own state
  - When the form is submitted we typically read the values from the DOM elements ourselves

```
class ProfileForm extends Component {
  handleSubmit = (event) => {
    event.preventDefault();

    const firstName = event.target.firstName.value;
    const lastName = event.target.lastName.value;

    // Here we do something with form data
    console.log(firstName, lastName)
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input name="firstName" type="text" />
        </label>
        <label>
          Surname:
          <input name="lastName" type="text" />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

First Name:

Last Name:

Submit

Uncontrolled Inputs

- "**ref** is used to receive the form value from DOM.
  - To enable this, React allows us to create a "ref" (reference) to associate with an element, giving access to the underlying DOM node.
  - Refs provide a way to access DOM nodes or React elements created in the render method.
  - Refs are created using React.createRef() and attached to React elements via the ref attribute.
  - Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component.

```
class MyComponent extends React.Component {
   constructor(props) {
      super(props);
      this.myRef = React.createRef();
  }
  render() {
      return <div ref={this.myRef} />;
  }
}
```

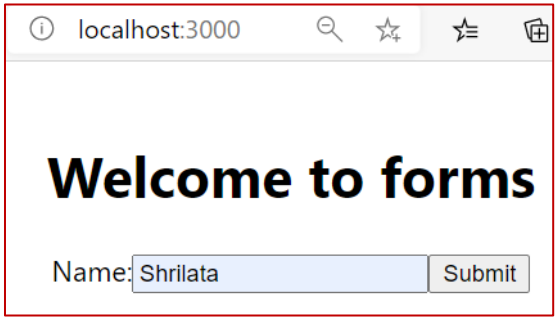
Uncontrolled Inputs

```
import React, {Component} from 'react';
class SimpleForm extends Component {
    constructor(props) {
        super(props);
        // create a ref to store the DOM element
        this.nameEl = React.createRef();
    }

    handleSubmit = (e) => {
      e.preventDefault();
      alert(this.nameEl.current.value);
    }

    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>Name:
            <input type="text" ref={this.nameEl} />
          </label>
          <input type="submit" name="Submit" />
        </form>
      )
    }
  }
export default SimpleForm;
```

- You initialize a new ref in the constructor by calling React.createRef, assigning it to an instance property so it's available for the lifetime of the component.
- In order to associate the ref with an input, it's passed to the element as the special ref attribute.
- Once this is done, the input's underlying DOM node can be accessed via this.nameEl.current.



Another example : Login form

```
class LoginForm extends Component {
   constructor(props) {
      super(props);
      this.nameEl = React.createRef();
      this.passwordEl = React.createRef();
      this.rememberMeEl = React.createRef();
   }
   handleSubmit = (e) => {
      e.preventDefault();
      const data = {
         username: this.nameEl.current.value,
         password: this.passwordEl.current.value,
         rememberMe: this.rememberMeEl.current.checked,
      }
      console.log(data)
   }
   render(){
      return (
       <form onSubmit={this.handleSubmit}>
         <fieldset><legend>Login Form</legend>
         <input type="text" placeholder="username" ref={this.nameEl} /><br></br>
         <input type="password" placeholder="password" ref={this.passwordEl} /><br></br>
         <label><input type="checkbox" ref={this.rememberMeEl} />Remember me
         </label><br></br>
         <button type="submit" className="myButton">Login</button>
         </fieldset>
       </form>
      );
}}
```

Login Form

username

password

Remember me

Login

```
▶ {username: 'aaa', password: 'bbb', rememberMe: true}
```