Form

CS568 – Web Application Development I

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- Controlled Components
- Form

Controlled Components

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**().

An input form element whose **value** is controlled by React in this way is called a "**controlled component**".

Read more: React forms and <input>

Controlled Components

In React, the source of truth is state. Meaning, if you want to read the value for the field, the value is stored in the state. What user entered is always synced up in the state.

You can still get the input value directly from the real DOM element. But don’t do that since it will slow down your program.

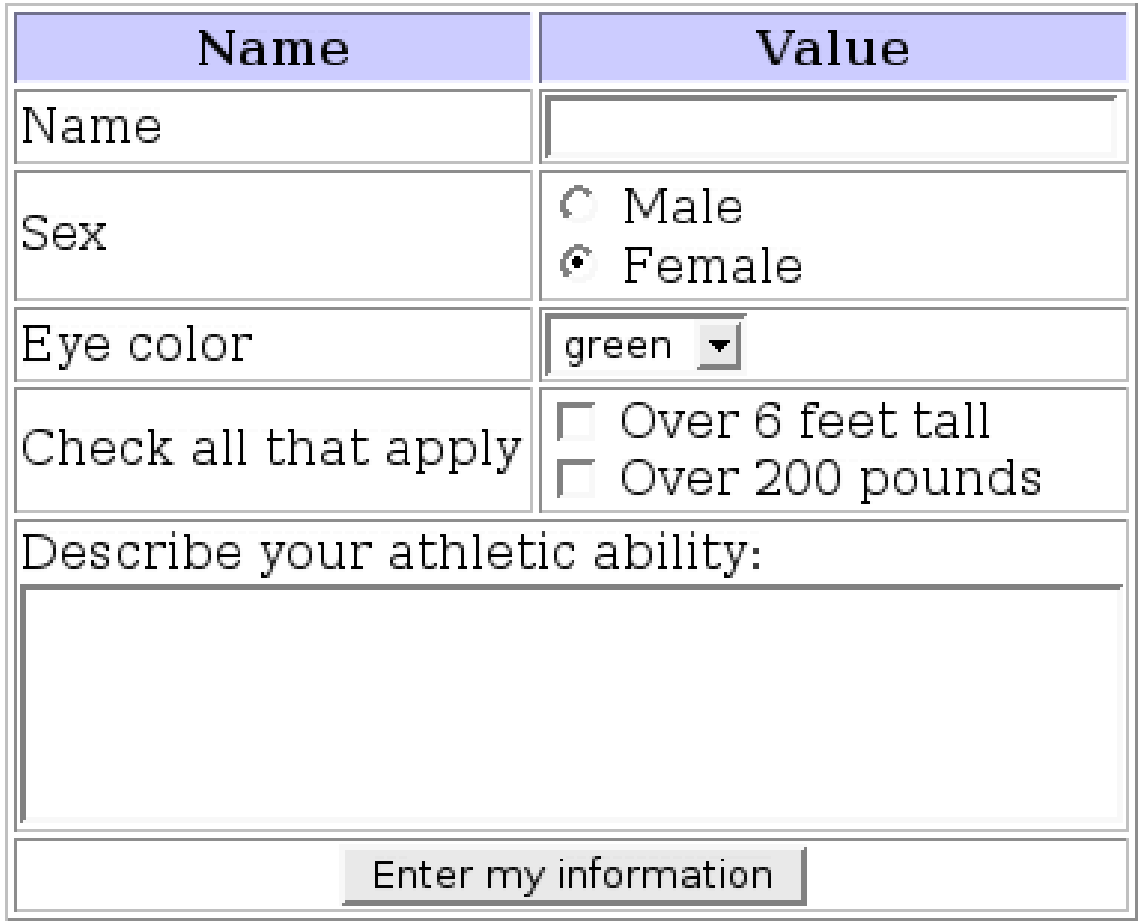
To read the input value from the state, you have to bind the input with the state. For that you set two properties of the component

1. **Value:** Set the value of this component
2. **OnChange:** Set the callback function to listen any change in this component

Components that read the input value from state is known as Controlled Components.

Web form

A webform or HTML form on a web page allows a user to enter data that is sent to a server for processing. Forms can resemble paper or database forms because web users fill out the forms using checkboxes, radio buttons, or text fields.



Inputs

- Text
- TextArea
- Select
- Radio
- CheckBox
- File

Text

```
const [value, setValue] = useState("");
  const change = (e) => {
    setValue(e.target.value)
  }
  return (
    <div className="App">
      <input type="text" value={value} onChange={change} />
    </div>
  );
```

TextArea

```
const [value, setValue] = useState("");
  const change = (e) => {
    setValue(e.target.value)
  }
  return (
    <div className="App">
      <textarea value={value} onChange={change} rows={10}
   coloumns={20}></textarea>
    </div>
  );
```

Select: Single option

```
const [value, setValue] = useState("");
  const change = (e) => {
    console.log(e.target.value)
    setValue(e.target.value)
  }
  return (
    <div className="App">
      <select value={value} onChange={change}>
        <option value="car">Car</option>
        <option value="bike">Bike</option>
      </select>
    </div>
  );
```

Select: multiple options

```
const [value, setValue] = useState([]);
  const change = (e) => {
    const options = Array.from(e.target.selectedOptions, option => option.value);
    setValue(options);
  }
  return (
    <div className="App">
      <select value={value} onChange={change} multiple={true}>
        <option value="car">Car</option>
        <option value="bike">Bike</option>
        <option value="yacht">Yacht</option>
      </select>
    </div>
  );
```

Radio button

Radio button uses the same input tag but is different from other standard inputs. User selects a single value from a group of radio button inputs for one field.

```
<label>
<input type="radio" value="male" name="gender" onChange={handleChange}/> Male
</label>
<label>
<input type="radio" value="female" name="gender" onChange={handleChange} /> Female
</label>
```

Checkbox

- Users can select multiple values of a limited choices

```
<label>
<input type="checkbox" value={state.option1} name="option1"
onChange={handleChange}/> Option1
</label>
<label>
<input type="checkbox" value={state.option2} name="option2"
onChange={handleChange}/> Option2
</label>
```

General handleChange for multiple input

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

File - onChange

```
const handleChange = e => {
console.log(e.target.files);
};
return (
<input type='file' onChange={handleChange} multiple/>
)
```

Storing an object in state

Let’s say that you have a form for the Student object. In that case, the Student object can have N number of attributes. You shouldn’t create N number of states for each attribute. Instead, just have ONE JSON state.

```
const [student, setstudent] = useState(
{"firstname":"","lastname":"","email":"","date":"","phone":"","colo
r":"#000000","education":"","courseSatisfaction":"","hobbies":[],"a
bout":"","luckynumber":""} )
```

Keyboard event: KeyDown/KeyDown

KeyDown and KeyDown are just like onChange event. It is handy when you want to allow users to use **only keyboard** to enter data.

KeyDown is triggered when user presses the keyboard button.

KeyUp is triggered when user releases the keyboard button.

```
const onKeyUp = (e) =>{
console.log(e.key,  e.keyCode);
}
return (
<input type='text' onKeyUp={onKeyUp}/>
);
```

Form Validation

Main reasons why we insist on validating forms:

- We want to get the right data in the right format
- We want to protect us from hackers. Hackers can mess up your app entering a malicious data in the form. For more details: React XSS Guide

We can validate form data on server side or client side. And we should validate on both side.

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.

Form Validation

You can implement validation:

- Manually yourself: onChange, onClick, onSubmit
- Regex: Define the validation function using Regex
- Third-party libraries: formik, useForm
- Built-in HTML5 validation: required, type, min, max, pattern...

Client-side validation enhances user experience, but server-side validation is crucial for data integrity and security against potential internet threats.

Form: Best practice

- Always put all inputs in the form
- Handle the onSubmit event of form by stopping the default action.

```
const handleSubmit = e => {
e.preventDefault();
//your code is here
};
```

Form: Best practice

```
const handleChange = e => {
const {name, value} = e.target;
setState({...state, [name]: value});
}
return (
<form onSubmit={handleSubmit}>
<input type="text" value={state.name} name="email" onChange={handleChange}/>
<input type="password" value={state.name} name="password" onChange={handleChange}/>
<input type="submit" />
<input type="reset" />
</form>
);
```

Summary

- Controlled Components
- Form