React & Forms

Handling and Validating User Input

React Form Events

- The events we looked at previously in React didn't have any data
  - An `onClick` event simply fires, that's that
- Form events, however, have data associated with them
- Not only does the user enter data we want, but they user also needs to see the data they've entered so far
- Because of that, we need to keep track of and re-render the form with that data
- We also want to keep track of the data entered so that we can use it when the user is ready to submit

React Forms - Input Requirements

- Inputs can be `<input>`, `<textarea>`, or `<select>` tags
- To properly have an input in react, we **must** provide two keys
  - `onChange` - A function that will fire when the input is changed
  - `value` - What the value of the input should be
- The `value` of the input should either come from a component's state (if it manages itself) or props (if `onClick` calls a prop function)
- If either of these isn't provided, the input may look like it's working, but a re-render could lose its values

React Forms - Basic <input>

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

    // This is called every time they type in the name input
    // ev.target is the input, so we can read its value
    _handleChangeName = (ev) => {
        this.setState({ name: ev.target.value });
    }

    // This gets re-rendered every time state changes, so every change to name
    render() {
        return (
            <div className="App">
                <span>Enter your name:</span>
                <input onChange={this._handleChangeName} value={this.state.name}/>
            </div>
        );
    }
}
```

React Forms - Multiple `<input>`s

- So that last example had one input, and one function to handle its change
- But it would be a pain to do that for *every single input*
- We can write a more generic handler that could handle any simple input
- All we need to do is set a dynamic key on the state object, instead of a hard coded key
- We'll also want to give each input a `name` attribute that has a matching key in the state object

React Forms - Multiple `<input>`s

```
class App extends React.Component {
    state = {
        firstName: '',
        lastName: '',
    };

    _handleChange = (ev) => {
        // You can make object keys dynamic by wrapping the key in []
        this.setState({ [ev.target.name]: ev.target.value });
    }

    render() {
        const { firstName, lastName } = this.state;
        return (
            <div className="App">
                <span>Enter your name:</span>
                <input name="firstName" onChange={this._handleChange} value={firstName}/>
                <input name="lastName" onChange={this._handleChange} value={lastName}/>
            </div>
        );
    }
}
```

React Forms - `<select>` Lists

- Select lists behave almost exactly the same as inputs
  - It still needs a `value` and `onClick` handler, and it would work in `this._handleChange`
- The only trick is that you must also provide an `<option>` tag as a child in the `<select>` for each of its options
  - Option tags must have a `value` attribute that can match select's `value`
  - They must also have something inside the tag, this is what's seen by users
- This is a great use case for `array.map`, to turn an array of options into a bunch of JSX `<option>` tags

React Forms - `<select>` Lists (Code)

```
class App extends React.Component {
    state = { favFood: '' };

    _handleChange = (ev) => { /* Same as before */ }

    render() {
        const { favFood } = this.state;
        const foods = ["Pizza", "Tacos", "Ice Cream"];

        return (
            <div className="App">
                <span>Enter your favorite food:</span>
                <select name="favFood" value={favFood} onChange={this._handleChange}>
                    {foods.map((food) => {
                        return <option key={food} value={food}>{food}</option>;
                    })}
                </select>
            </div>
        );
    }
}
```

`Array.map` in JSX

```
<select name="favFood" value={favFood} onChange={this._handleChange}>
    {foods.map((food) => {
        return <option key={food} value={food}>{food}</option>;
    })}
</select>
```

- The map function here converts an array like `["1", "2", "3"]` to `[<option>1</option>, <option>2</option>, <option>3</option>]`
- JSX is able to render arrays, each element gets placed in order
- `key` attribute is added for React to keep track of each one though (More on this later)

Input / Form Validation

- So far all of our inputs have been fine with whatever users enter into them
- But this is often not the case, we sometimes have strict fields
- The nice thing about checking every input from a user is we can either tell them immediately, or fix their problems for them
- We render the value that's in state, so we don't have to keep their input the same!

Input / Form Validation - Error Message

```
_handlePasswordChange = (ev) => {
    const password = ev.target.value;
    const error = password.length < 5 && "Password too short!";

    this.setState({ password, error });
}

render() {
    const { username, password, error } = this.state;
    return (
        <form onSubmit={this._handleSubmit}>
            <input name="username" onChange={this._handleChange} value={username}/>
            <input name="password" onChange={this._handlePasswordChange} value={password}/>
            {error && <div className="error">{error}</div>}
        </form>
    );
}
```

Input / Form Validation - Fix Input

- What the user types is in ev.target.value, but we can set anything to state
- Here we run the value through a phone number formatter before saving to state

```
// Force input to be lower case
_handleEmailChange = (ev) => {
    const email = ev.target.value.toLowerCase();
    this.setState({ email });
};

render() {
    return (
        <form onSubmit={this._handleSubmit}>
            <input
                name="email"
                onChange={this._handleEmailChange}
                value={this.state.phoneNum}
            />
        </form>
    )
}
```

Handling `<form>` Submission

- So you've made some form elements, and now you're ready to submit. How do we do it?
- Same as old JS, we'll have a listener on the `submit` event via `onSubmit` prop
- Once it submits, we don't need to grab all of the input values
- We already have them in state, from keeping track earlier!

Handling `<form>` Submission (code)

```
class App extends React.Component {
    state = {
        username: '',
        password: '',
    };

    // Submits state data to some submitLogin promise function
    _handleSubmit = (ev) => {
        ev.preventDefault(); // We still need to do this!
        submitLogin(this.state.username, this.state.password).then(() => {
            window.location = "/home";
        });
    }

    render() {
        return (
            <form onSubmit={this._handleSubmit}>
                {/* Username / PW inputs that are saved to state with _handleChange */}
            </form>
        );
    }
}
```

Simple Example: Mock Chat App

- While the previous example showed some `submitLogin` function from somewhere else, we'll look at an example that we can actually see every part
- Let's say we have a chat app with two major pieces to it:
  - The top part, which displays all of the messages in the chat
  - A bottom part, which is a form where a user can enter a message
- We'll want to listen to the input in the form to update as the user is typing
- But once the user submits, we'll want clear the inputs, and add their message to state so that it renders in the top part of the app

https://github.com/wbobeirne/nycda-react-chat

Your Turn: Improve our App

Make the following improvements to the chat app:

- Add an input to the form that is the user's name, attach their name to the message object
  - This name should also stay in state
  - It should not reset every time they send a message
- On form submission, validate that the user has entered some text before submission
  - If they try to, show an error message in the form (Don't just use `alert`)
  - You will need to add a new element and style it yourself
- Display the name of the user with every message
  - You can add a new element and style it yourself in the message map function

Additional Reading

- React Docs - Handling Events (In case you're still shaky on events)
- React Docs - Forms
- React Docs - Lifting State Up