ReactJS Events, Forms and Lists

by Vlad Costel Ungureanu
for “Learn Stuff” and “Pentalog Romania”

ReactJS
Events, Form and Lists

- ✓ Event Handling
- ✓ Lists
- ✓ Forms
- ✓ Error Handling

Event Handling

✓ In HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

✓ In ReactJS:

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

Event Handling

✓ Default Behavior can only be explicitly prevented

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }
}

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

Event Handling

- ✓ All events in ReactJS are implemented according to the W3C specifications and act the same as any other JS events
- ✓ Standard event handlers are not aware of the component context and as such cannot access the components “this” reference
- ✓ This is fixed by binding of handlers in the constructor function of a component or using arrow functions.
- ✓ Aside from the normal use of handler it is also common to pass additional parameters to a handler function:
  - ✓ <button onClick={(**e**) => this.deleteRow(id, **e**)}>Delete Row</button>
  - ✓ <button onClick={this.deleteRow.bind(**this, id**)}>Delete Row</button>

Event Handling

- ✓ All events in ReactJS are implemented according to the W3C specifications and act the same as any other JS events
- ✓ Standard event handlers are not aware of the component context and as such cannot access the components “this” reference
- ✓ This is fixed by binding of handlers in the constructor function of a component or using arrow functions.
- ✓ Aside from the normal use of handler it is also common to pass additional parameters to a handler function:
  - ✓ <button onClick={(**e**) => this.deleteRow(id, **e**)}>Delete Row</button>
  - ✓ <button onClick={this.deleteRow.bind(**this, id**)}>Delete Row</button>

Lists

✓ ReactJS use the "**map**" function to work with lists

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
console.log(doubled);
// 2 4 6 8 10
```

✓ So, in order to have a list rendered we can use

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Keys

- ✓ In order to avoid performance issues when updating list items, REACT requires the use of keys

```
// usually we have some sort of identifier as a property in each list item
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);

// using index when an identifier is not present in the list
const todoItems = todos.map((todo, index) =>
<li key={index}>
    {todo.text}
  </li>
);
```

Keys

✓ Keys only need to be unique among siblings

✓ Also, the “map” function can be embedded in JSX

```
function NumberList(props) {
  const numbers = props.numbers;
  return (
    <ul>
      {numbers.map((number) =>
        // we use a component for each list item
        <ListItem key={number.toString()}
                  value={number} />
      )}
    </ul>
  );
}
```

Forms

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    // binding
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }
}
```

Forms

```
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name:
        <input type="text" value={this.state.value} onChange={this.handleChange} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
 }
}
```

Forms

✓ ReactJS supports all HTML inputs: Input, Textarea, Select

✓Textarea behaves as an input n the sense that it uses a “value” attribute

```
<form onSubmit={this.handleSubmit}>
      <label>
        Pick your favorite flavor:
        <select value={this.state.value} onChange={this.handleChange}>
          <option value="grapefruit">Grapefruit</option>
          <option value="lime">Lime</option>
          <option value="coconut">Coconut</option>
          <option value="mango">Mango</option>
        </select>
      </label>
      <input type="submit" value="Submit" />
</form>
```

Forms

- ✓ While form creation is straight forward, we must make sure the used components in the form do no interact in unexpected ways
- ✓ Since there is no two-way binding by default, forms input changes need to be handled manually which may lead to more code
- ✓ Usually helper libraries are used to simply form components
- ✓ Another frequent practice is passing handler function bound to the form component context to the appropriate child component

Error Handling

- ✓ ReactJS introduces dedicated components that can catch errors from component trees, treat the errors and avoid crashing UI

- ✓ Any React component that defines a "***componentDidCatch***" method becomes an error boundary

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }
  componentDidCatch(error, info) {
    // Display fallback UI
    this.setState({ hasError: true });
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info);
  }
}
```

Error Handling

```
  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}

// Usage
<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```

Error Handling

- ✓ While there are no strict rules in using error boundaries, two approach seem sensible:
  - ✓Top level boundaries to prevent page crashes
  - ✓Component level boundaries for strict, distinct behavior for specific components

- ✓ Since ReactJS 16, any uncaught exception will cause the component tree to unmount

- ✓ Try/Catch usage is not indicated as it serve error management in imperative code, while React uses declarative code to describe what should be rendered.

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com