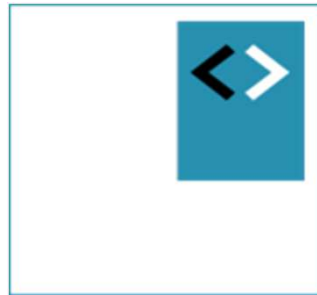


# React Fundamentals

## Module – handling user input



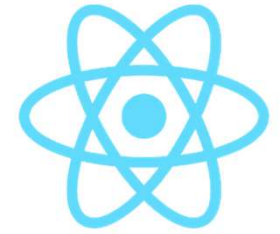
Peter Kassenaar –  
[info@kassenaar.com](mailto:info@kassenaar.com)



# Handling user input

Working with input from textboxes, radio buttons, and other form fields

# Types of form fields



- **User input** can come from a variety of elements:
  - `<input type="text">`
  - `<input type="checkbox">`
  - `<input type="radio">`
  - `<select> + <option>`
  - `<textarea>`
  - ...

# React – types of input components

## Controlled Components

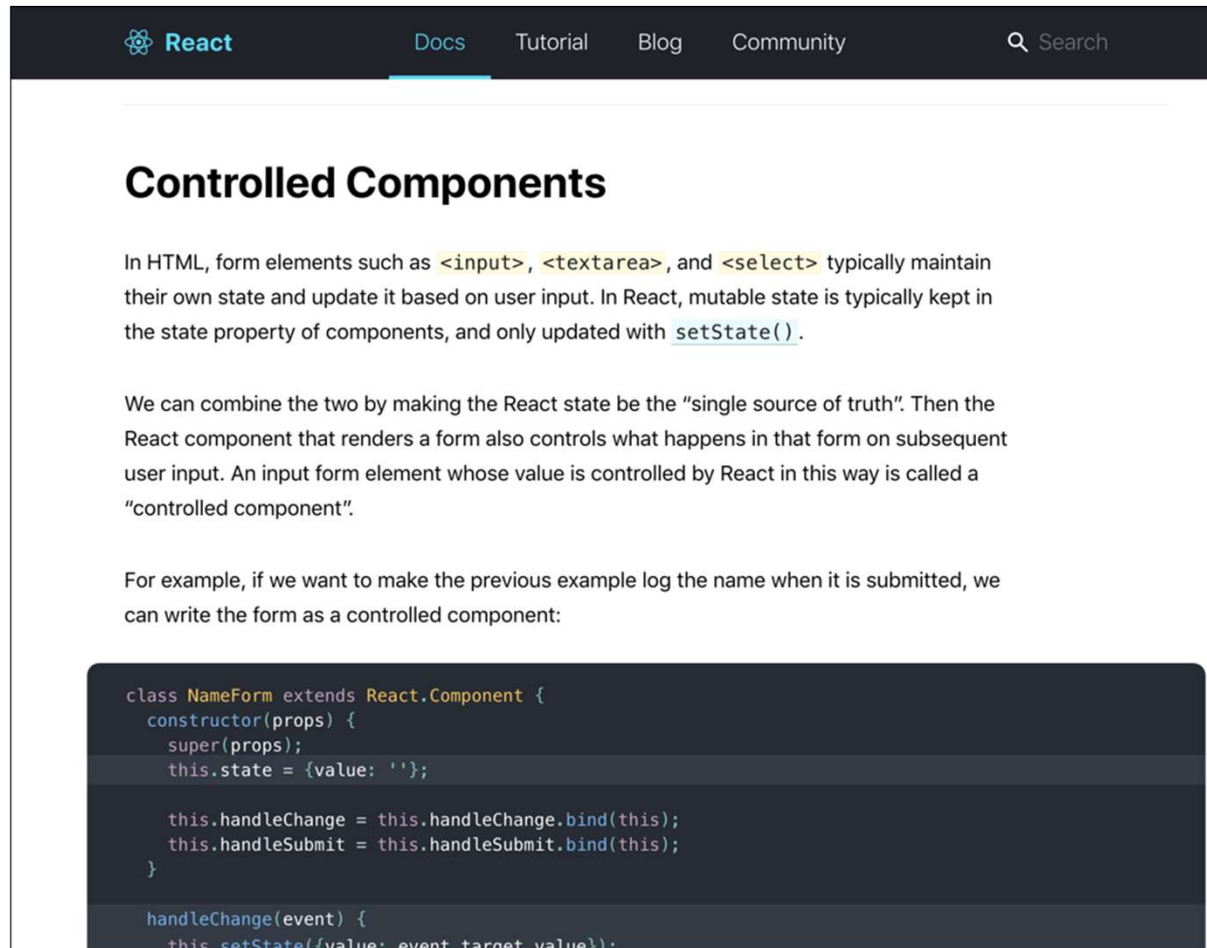
- React is the *single source of truth* for the component
- Component value is driven from the `state`
- Typically: more work, but *better control*
- Changes are *pushed* to the `state`

## Uncontrolled Components

- Form data is handled by the DOM itself
- State lives in the HTML component
- Less work, but *less control*
- Changes need to be *pulled* from the component via `ref`

# Controlled components

## Preferred way of handling input controls



The screenshot shows the React.js documentation page for "Controlled Components". The page has a dark header with the React logo and navigation links: Docs, Tutorial, Blog, and Community. A search bar is also present. The main content area has the title "Controlled Components" and two paragraphs of text. The first paragraph explains that in HTML, form elements like `<input>`, `<textarea>`, and `<select>` maintain their own state, while in React, state is managed by components using `setState()`. The second paragraph describes how to combine these by making the React state the "single source of truth", where the component controls the form's behavior. The third paragraph provides an example of making a form a controlled component by logging the name on submission. Below the text is a code block showing a `NameForm` class that extends `React.Component`. The code includes a constructor that initializes state and binds event handlers, and a `handleChange` method that updates the state with the input value.

**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()`.

We can combine the two by making the React state be the "single source of truth". Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a "controlled component".

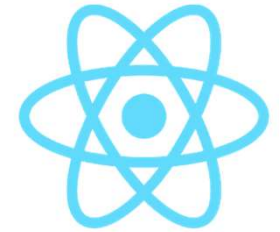
For example, if we want to make the previous example log the name when it is submitted, we can write the form as a controlled component:

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

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

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

<https://reactjs.org/docs/forms.html>



# Let's create a controlled component

## Most used – creating a textinput box

Example: ../components/AddCountries

```
// 0. We need state for this component
state = {
  newCountry: '',
  newCountries: []
};
```

```
{ /*1. We need a textbox*/}
<input type="text"
  placeholder="New country..."
  className="form-control-lg"
  value={this.state.newCountry}
  onChange={event => this.updateCountry(event)}
/>
```

Controlled components **need** a `value` property and an `onChange` event handler. They are driven from – and update- the `state` of the component

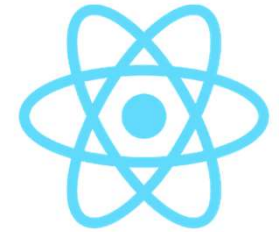
# Controlled component event handlers

```
// 1. updateCountry - updates the value of newCountry in the state  
// on every 'change' event (i.e. every keystroke)  
updateCountry(event) {  
  this.setState({  
    newCountry: event.target.value  
  })  
}  
  
// 2. addCountry - add a new country to the array of countries.  
// It uses the current value (i.e. the state) of the textbox.  
addCountry() {  
  this.setState({  
    newCountries: [...this.state.newCountries, this.state.newCountry]  
  })  
}
```




```
<button className="btn btn-success"  
  onClick={() => this.addCountry()}>  
  Add Country  
</button>
```

# Result



**New Countries**

Canada	Add Country
Finland	
Canada	

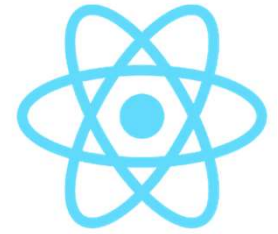
A large red arrow pointing downwards from the first input field to the second.

Example [../400-textboxes](#)

Summary –  
Controlled components  
have a `value` property  
and an `onChange` event  
handler



# Multiple textboxes?

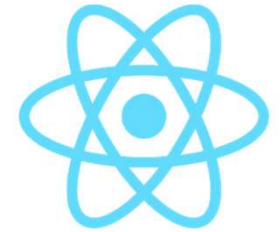


1. Add new `state` properties
2. Add a `name` attribute to the textbox. Set it to the property in the state
3. Handle/retrieve the name in the `onChange` handler

```
state = {  
  name: '',  
  population: '',  
  newCountries: []  
};
```

```
<input type="text"  
  name="name"  
  value={this.state.name}  
  onChange={ (event => this.updateCountry(event)) }  
/>  
<input type="text"  
  name="population"  
  value={this.state.population}  
  onChange={ (event => this.updateCountry(event)) }  
/>
```

```
updateCountry(event) {  
  // Create constants...  
  const target = event.target;  
  const value = target.value;  
  const name = target.name;  
  this.setState({  
    [name]: value  
  })  
}
```

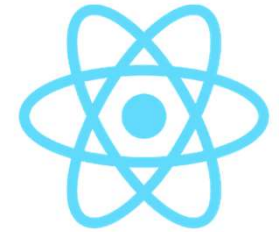


# Adding the new object

1. Create a new object in the `onClick`-handler of the button,
2. add it to the `state`
3. Reset the `state`

```
addCountry() {  
  // 1. create a new country, based on the state  
  const newCountry={  
    name: this.state.name,  
    population: this.state.population  
  };  
  // 2. set the state - and reset the individual fields  
  this.setState({  
    name: '',  
    population: '',  
    newCountries: [...this.state.newCountries, newCountry]  
  })  
}
```

# Result



## New Countries

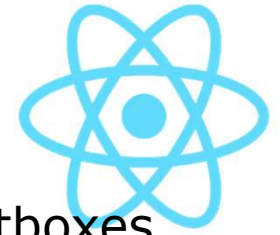
Add Country

Canada- (pop. 20000000)

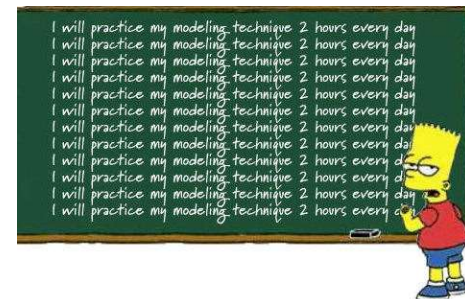
Finland- (pop. 4000000)

Example `../410-multiple-textboxes`

# Workshop



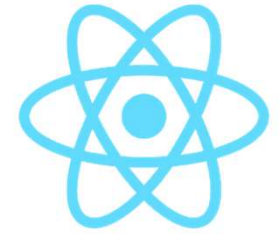
- Own application: create a component containing multiple textboxes. Add them to a new array in the state.
- OR: add textboxes to the VacationPicker in example `../410-multiple-textboxes`
- Add a new country to the list of countries, creating textboxes for every property
- This is only *in-memory* for now. You can push it to a backend later.
- Optional: create a **Delete** button for every new item, to remove the item from the array (CRUD)





# Creating checkboxes

Selecting one or more items in a list of items



# Uncontrolled checkbox

- We have now created an *uncontrolled component*, b/c its state (checked/not checked) is controlled by the DOM.
- To create a *controlled component*, you'd modify the state (add a `visited` property) and set its value in the UI (`<input type="checkbox" checked={country.visited} />`)
- Then also create an `onChange` handler

# Checkboxes

Update state

```
// Local state holding an array of visited countries.  
state = {  
  visitedCountries: []  
};
```

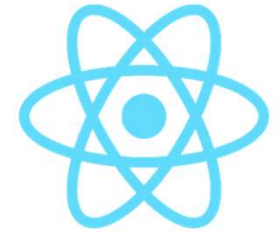
Add to UI

```
/*Checkbox, indicating if you visited a country*/  
/*This is an *uncontrolled component* as it doesn't have a checked attribute*/  
/*an onChange handler and a value/checked property*/  
<input type="checkbox"  
  onClick={() => this.checkCountry(country)}
```


Handle logic

```
checkCountry(country) {  
  // The country can be only once in the array  
  if (this.state.visitedCountries.indexOf(country) > -1) {  
    const newVisitedCountries = this.state.visitedCountries.filter(i => i !== country)  
    this.setState({  
      visitedCountries: newVisitedCountries  
    })  
  } else {  
    // it's not there yet, so add it.  
    this.setState({  
      visitedCountries: [...this.state.visitedCountries, country]  
    });  
  }  
}
```





# Result

 **React vacation picker**

☒ USA

☒ Netherlands

☒ Belgium

☐ Japan

☐ Brazil

☐ Australia

**I visited:**

Belgium

Netherlands

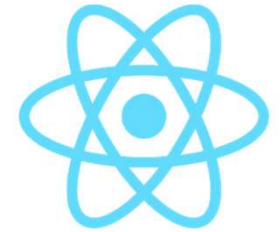
USA

`../examples/420-checkboxes`



# Creating radio buttons

Selecting one item in a list of items

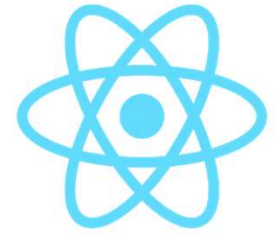


# Radio buttons

- Only one (1) button at a time can be selected.
  - Set the `name` property of the radio buttons to the same value.

```
<ul className="list-group">
  {this.props.countries.map(country =>
    <li>
      ...
      <label>
        <input type="radio"
          name="countries"
          onClick={() => this.checkCountry(country)}
        />   
        {country.name}
      </label>
    </li>
  )}
</ul>
```

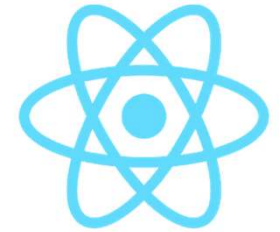
## Updating checkCountry()




- The `checkCountry()` function is now simpler:

```
checkCountry(country) {  
  // The country now comes from a radio button, so only one  
  // at a time can be selected. Easy peasy.  
  this.setState({  
    visitedCountries: [country]  
  });  
}
```

# Result



 **React vacation  
picker**

☐ USA

☐ Netherlands

☒ Belgium

☐ Japan

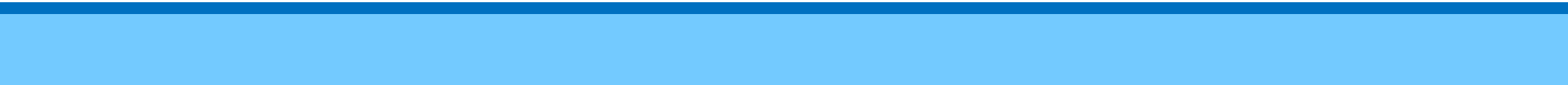
☐ Brazil

☐ Australia

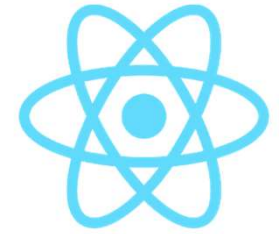
In Belgium they actually speak three different official languages: Flemish, French and German.

A large red arrow pointing downwards from the 'Belgium' radio button to the 'My favorite:' section.**My favorite:**

Belgium

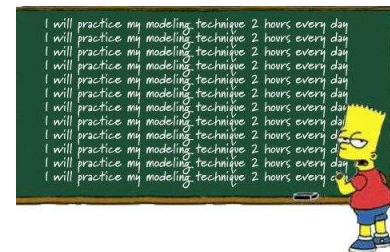


# Workshop

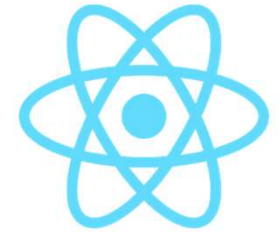


- Create a Todo-component with a list of Todo's
  - Place a checkbox before each Todo item
- If a Todo is marked as completed, add it to an array of completed Todo's.
  - Show this array in the UI
  - Show the completed Todo with a `strikethrough` class in the UI
- Add a new component, showing the same list.
  - Now place a radio button before each item
  - You should be able to mark an item as a 'favorite' Todo
- Example :

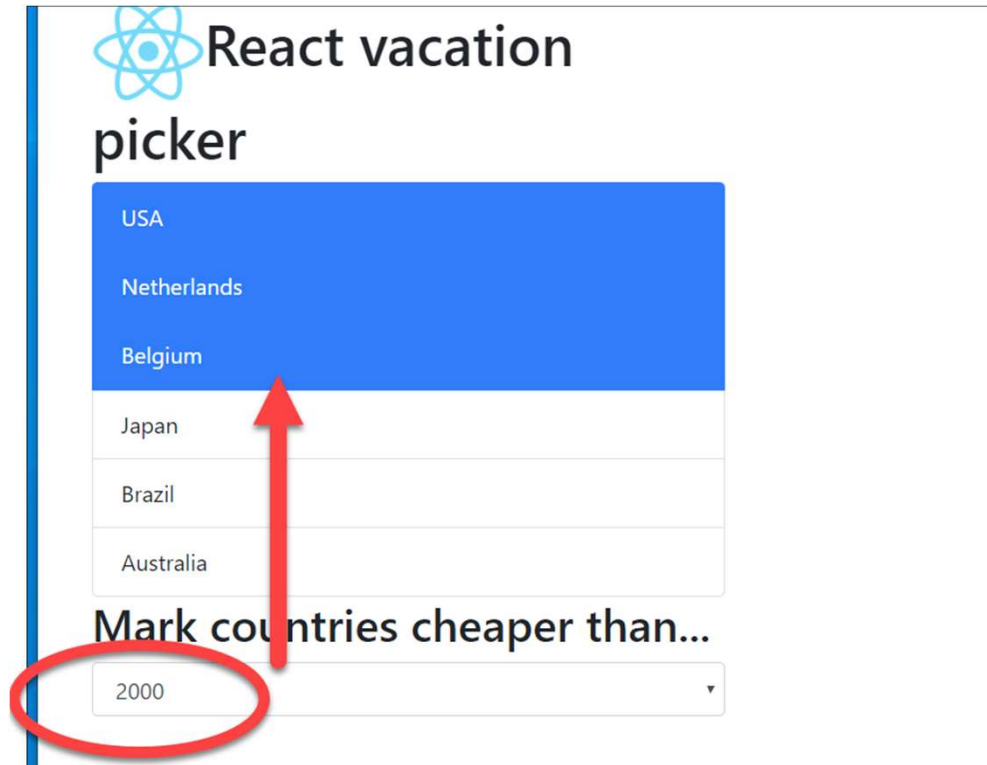
[../examples/430-radio-buttons](http://examples/430-radio-buttons)



# Select/dropdown lists



- We want to conditionally mark countries (i.e. add/remove a CSS-class) based on a value in a selection list

A screenshot of a web form titled "React vacation picker". The form contains a list of countries: USA, Netherlands, Belgium, Japan, Brazil, and Australia. The "USA" option is highlighted with a blue background. A red arrow points from the "2000" value in a dropdown menu to the "Belgium" option in the list. The dropdown menu is labeled "Mark countries cheaper than..." and has "2000" selected. The entire form is enclosed in a light gray border.

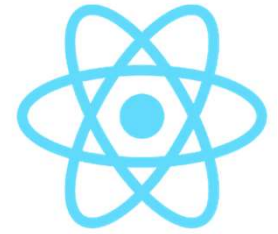
React vacation  
picker

USA
Netherlands
Belgium
Japan
Brazil
Australia

Mark countries cheaper than...

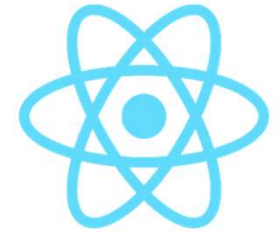
2000

## Using `<select>` and `<option>`



- In a selection list, we also use the `value` and `onChange` handler to read/set it's values
- This is NOT standard HTML. It is created for/by React
- The selected option is set as the `value` for the complete list.
- Selecting a new option, is handled by `onChange`.





# 1. Creating the selection list

```
<div className="form-group">
  <select
    value={this.state.price}
    onChange={(e) => this.updatePrice(e)}
    className="form-control"
    name="select">
    {
      this.state.prices.map(price =>
        <option
          key={price}
          value={price}>
            {price}
          </option>
        )
    }
  </select>
</div>
```



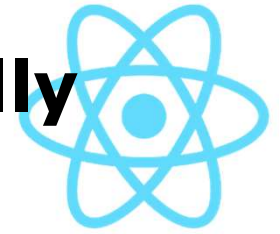
## 2. Updating the state



We pass in the event and read the selected value from `event.target.value`

```
// The state now also holds prices that are shown in the selection list.  
// By default the currently selected price is 1000.  
state = {  
  visitedCountries: [],  
  prices: [1000, 2000, 3000, 4000, 5000],  
  price: 1000  
};  
  
updatePrice(event) {  
  console.log('selected value:', event.target.value);  
  this.setState({  
    price: event.target.value  
  })  
}
```

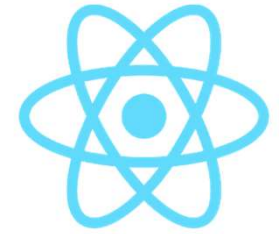
### 3. Adding/removing classes conditionally



```
<ul className="list-group">
  {this.props.countries.map(country =>
    <li
      // conditionally add class name.
      // We're using the Bootstrap class .active here,
      className={'list-group-item ' + (this.state.price > country.cost ? 'active' : '')}
      key={country.id}
      id={country.id}
      title={country.details}>
        {country.name}
    </li>
  )}
</ul>
```

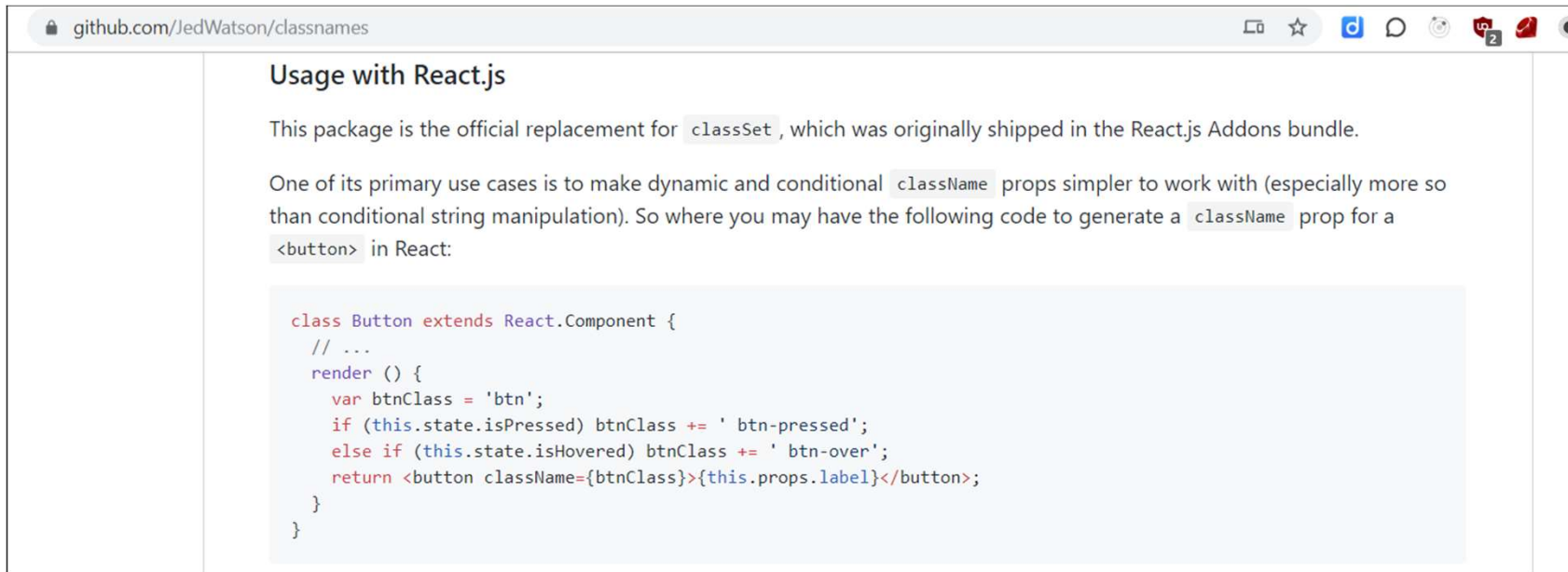
We are performing a simple inline comparison here. Of course you can also create a function for this, or handle conditional classes in a more elegant way.

# Using the `classnames` npm package



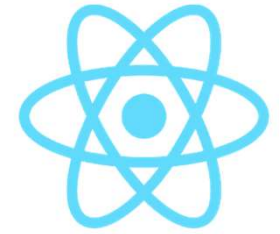
If you want to do more complex class operations, you might want to use the `classnames` npm package

<https://github.com/JedWatson/classnames>

A screenshot of a web browser showing the GitHub repository page for 'JedWatson/classnames'. The browser's address bar shows 'github.com/JedWatson/classnames'. The page content includes a section titled 'Usage with React.js' which explains that the package is the official replacement for 'classSet' and provides a code example for using it in a React component.

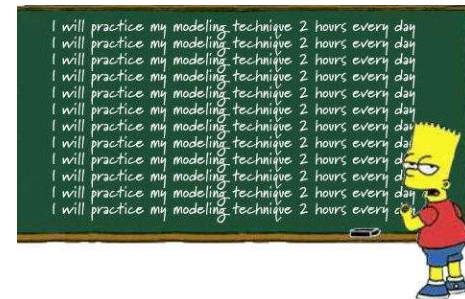
```
class Button extends React.Component {  
  // ...  
  render () {  
    var btnClass = 'btn';  
    if (this.state.isPressed) btnClass += ' btn-pressed';  
    else if (this.state.isHovered) btnClass += ' btn-over';  
    return <button className={btnClass}>{this.props.label}</button>;  
  }  
}
```

# Workshop



- Continue with your Todo-component from the previous workshop
- If a Todo is marked as completed, updated its class, so it is shown as strike-through in the UI
  - Tip: use the CSS-property `text-decoration: line-through;`
- Optional: install `classnames` package via npm and explore its possibilities.
- Example :

[../examples/440-selection-lists](https://examples/440-selection-lists)

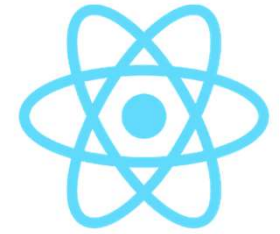




# Submitting forms

Sending your forms to a backend

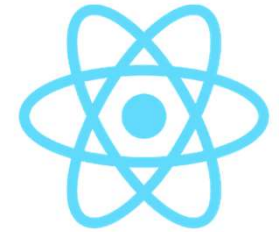
# Submitting a form



- Use the `<form>` tag with an `onSubmit()` handler
- Write your custom handler
  - don't forget to pass in the event and call `.preventDefault()`
- You can use props as usual

```
<form onSubmit={(e) => this.submitCountry(e)}>  
  ...  
  <input type="submit" className="btn btn-success"  
    value="Submit new country"/>  
</form>
```

# Submitting a country

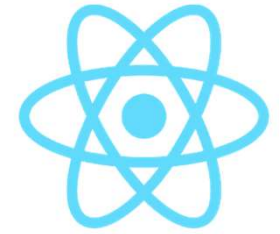


```
// 2. addCountry - create a new country and send it to the parent.  
// It uses the current value (i.e. the state) of the form input.  
submitCountry(event) {  
  // 0. prevent sending the form to the server  
  event.preventDefault();  
  
  // 1. create a new country, based on the state  
  const newCountry = {  
    name: this.state.name,  
    capital: this.state.capital,  
    cost: this.state.cost,  
    details: this.state.details  
  };  
  
  // 2. send the country to the parent  
  this.props.submit(newCountry)  
}
```





# Handling a form submission

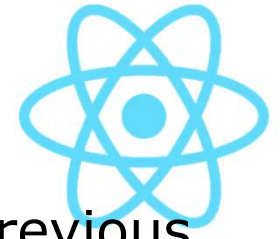


In our case: adding a new country to the state (still just *in-memory* only!)

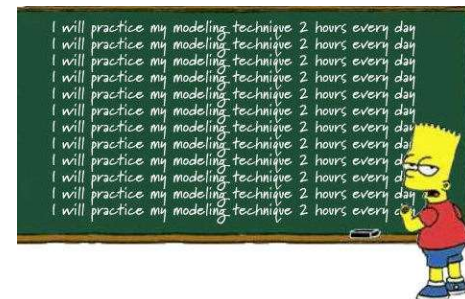
```
<AddCountries submit={country => this.addCountry(country)}/>
```

```
addCountry(country) {  
  this.setState({  
    countries: [...this.state.countries, country]  
  })  
}
```

# Workshop




- Continue using your own app, or your TODO-app from previous workshops:
  - Add some form fields to add data.
  - Create a 'real' form by using the `<form>` tag and submit the form using `onSubmit()`.
- Ready made example `../450-form-submit`



# More info



 **React**

Docs Tutorial Blog Community

🔍 Search

v16.11.0 🌐 Languages GitHub

## Forms

HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state. For example, this form in plain HTML accepts a single name:

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

This form has the default HTML form behavior of browsing to a new page when the user submits the form. If you want this behavior in React, it just works. But in most cases, it's convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form. The standard way to achieve this is with a technique called "controlled components".

INSTALLATION ▾

MAIN CONCEPTS ▲

1. Hello World

2. Introducing JSX

3. Rendering Elements

4. Components and Props

5. State and Lifecycle

6. Handling Events

7. Conditional Rendering

8. Lists and Keys

**9. Forms**

10. Lifting State Up

11. Composition vs Inheritance

12. Thinking In React

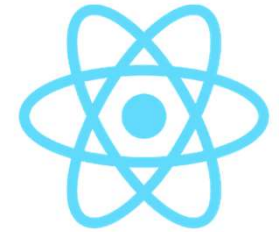
ADVANCED GUIDES ▾

API REFERENCE ▾


HOOKS ▾

<https://reactjs.org/docs/forms.html>

# Blog article on forms




**Medium** Javascript

Q 🔔 Upgrade 

---

## The complete guide to Forms in React

a letter about react forms to me in the future

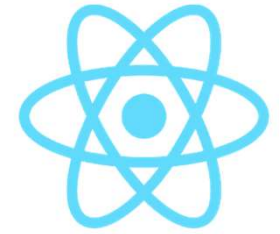
 Agoi Abel [Follow](#)  
Sep 7, 2018 · 6 min read

*Forms are very useful in any web application. Unlike angular and angularjs, that gives form validation out of the box. You have to handle forms yourself in React. This brought about many complications like how to get form values, how do I manage form state, how do I validate my form on the fly and show validation messages. There are different methods and libraries out there to help with this but if you are like me that hates dependent on too many libraries, welcome on board, We are going to bootstrap our own form from the ground up.*

There are two types of form input in `react`. We have the `uncontrolled input` and the `controlled input`. The `uncontrolled input` are like traditional HTML form inputs, they remember what you typed. We will use `ref` to get the form values.

<https://medium.com/@agoiabeladeyemi/the-complete-guide-to-forms-in-react-d2ba93f32825>

# Checkpoint



- You know about handling **form elements** in your app
  - Text fields
  - Textareas
  - Checkboxes
  - Radio buttons
  - Selection lists
- You know how to set classnames **conditionally**, based on some form value
- You can **submit** forms to a parent component (or database)