React Fundamentals

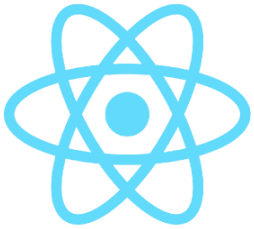
Module – Handling user input

Peter Kassenaar –
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

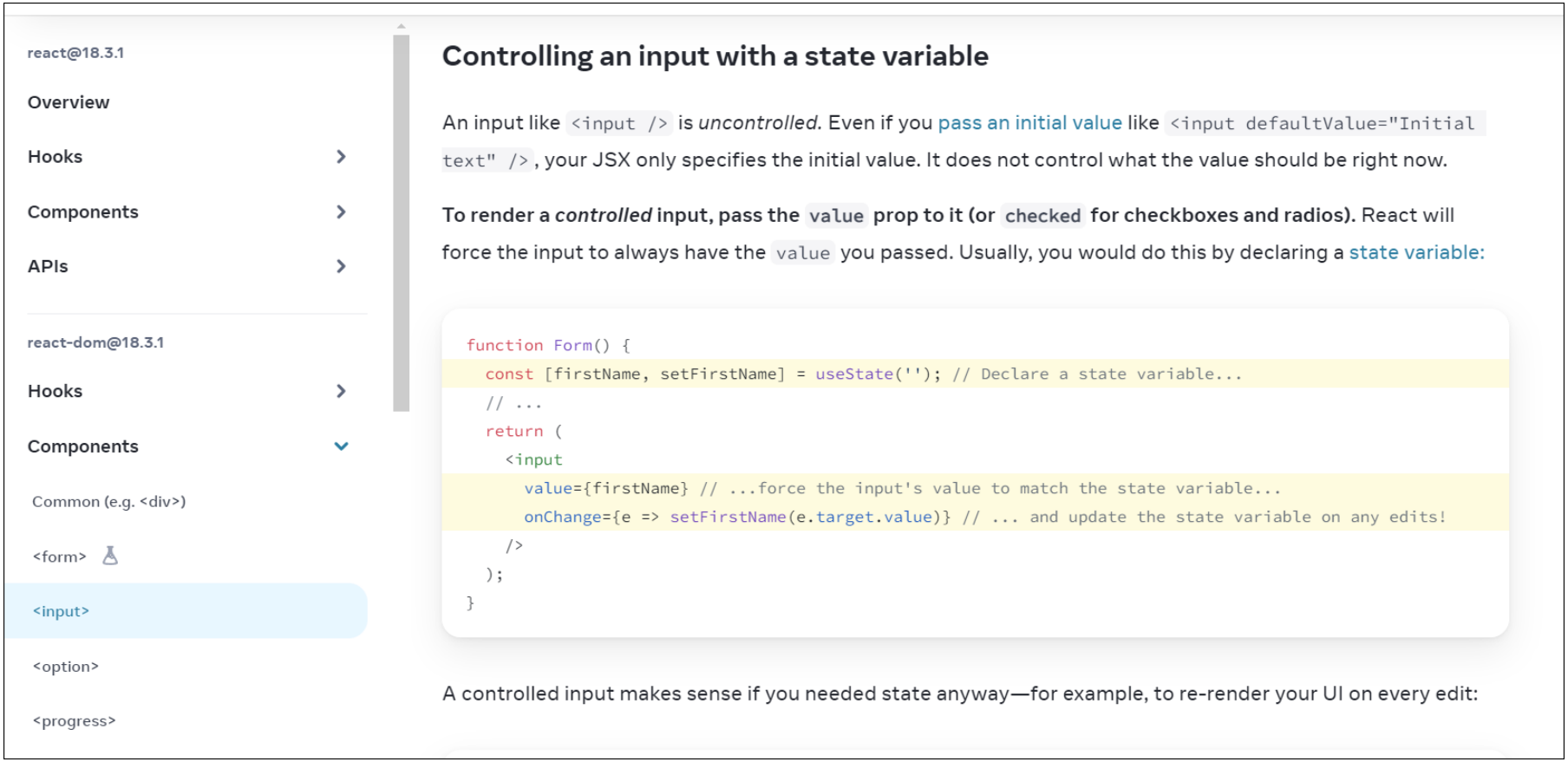
- React is the *single source of truth* for the component
- Component value is driven from the `state`
- Typically: more work, but better control
- Changes are *pushed* to the `state`

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`

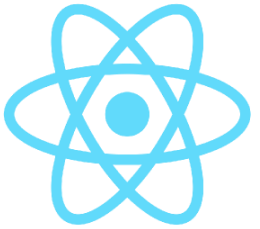
Controlled components

Preferred way of handling input controls



https://react.dev/reference/react-dom/components/input#controlling-an-input-with-a-state-variable

Let’s create a controlled component



Most used – creating a textinput box

Example: `../components/AddCountries`

```
// 0. We need state for this component
const [newCountry, setNewCountry] = useState('');
const [newCountries, setNewCountries] = useState([]);
```

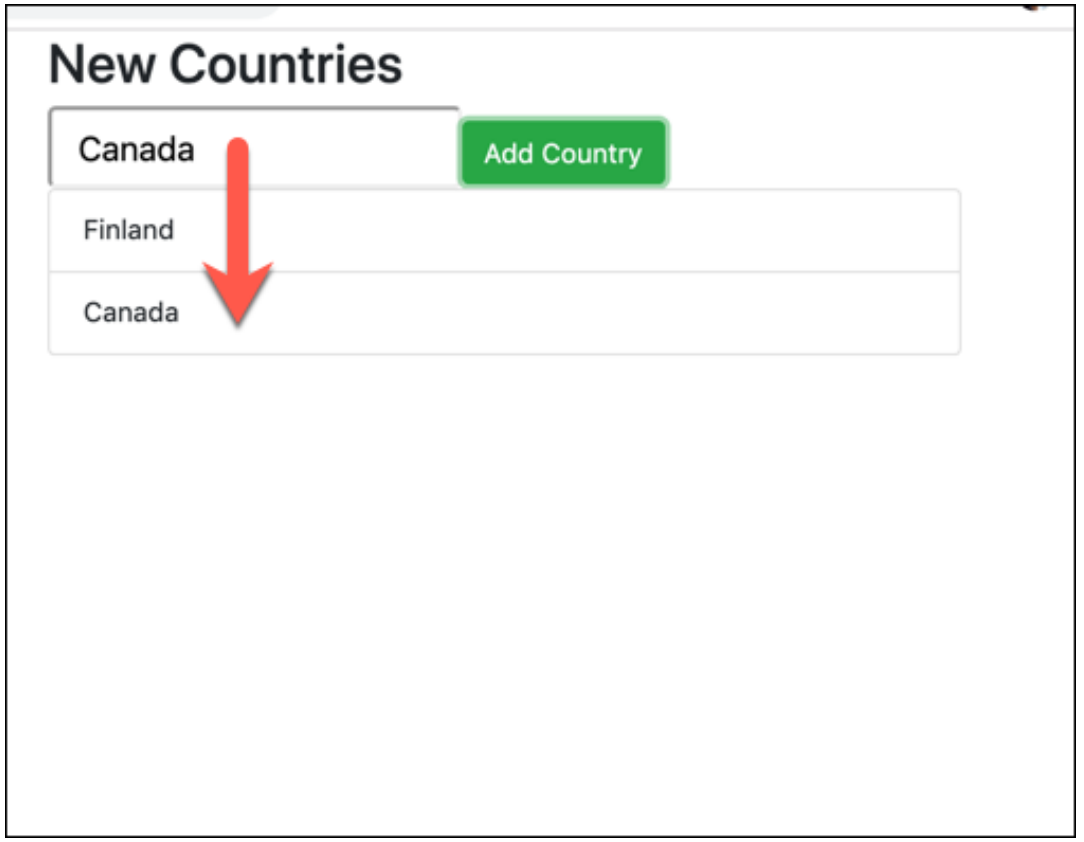
```
{/*1. We need a textbox*/}
<input type="text"
       placeholder="New country..."
       className="form-control-lg"
       value={newCountry}
       onChange={event => updateCountry(event))}
/>
```

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

```
// 1. updateCountry - updates the value of newCountry in the state
// on every 'change' event (i.e. every keystroke)
const updateCountry = (event) => {
    setNewCountry(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.
const addCountry = () => {
    setNewCountries([...newCountries, newCountry]);
}
```

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

Result

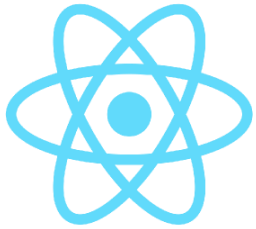


Example `../400-textboxes`

Summary –

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

Multiple textboxes?



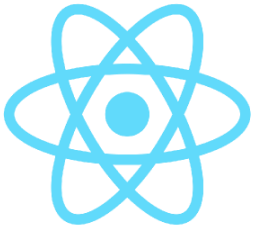
1. Add new `state` properties
2. Add a `name` attribute to the textbox. Set it to the property in the state
3. Handle/retrieve the name in the `onChange` handler

```
// We need state for this component
const emptyCountry = {name: '', population: ''}
const [newCountry, setNewCountry] = useState(emptyCountry)
const [newCountries, setNewCountries] = useState([])
```

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

```
// UpdateCountry - updates the value of newCountry in the state
// on every 'change' event (i.e. every keystroke).
const updateCountry = (event) => {
    // Create constants, find out which control the
    // value comes from, by retrieving the 'name' attribute.
    // Then set the correct state property
    const name = event.target.name;
    const value = event.target.value;
    setNewCountry({
        ...newCountry, // spread in the existing, current country
        [name]: value // update the given property
    })
}
```

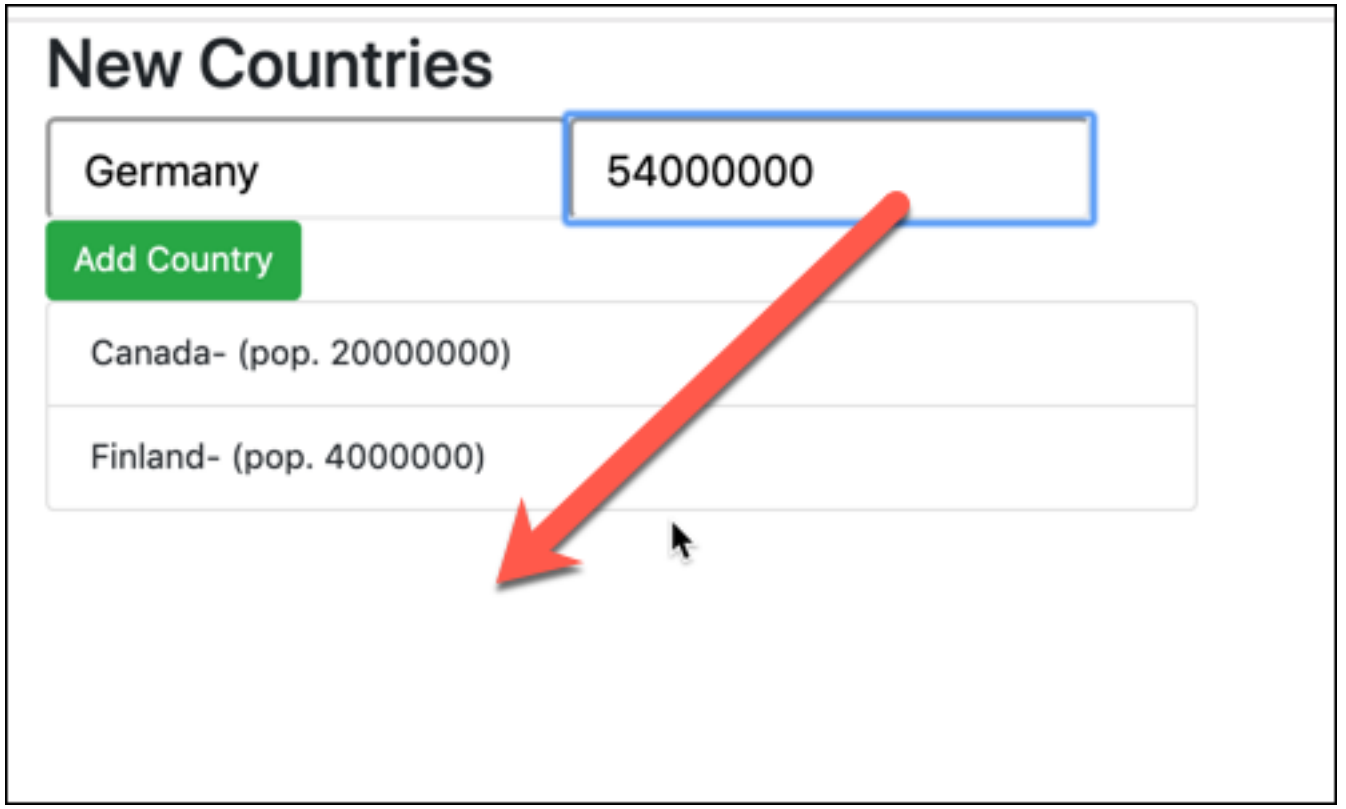
Adding the new object



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

```
// AddCountry - add a new country to the array of countries.
// It uses the current value (i.e. the state) of the textbox.
const addCountry = () => {
    // Set the state - and reset the individual fields w/ an empty country
    setNewCountries([...newCountries, newCountry])
    setNewCountry(emptyCountry)
}
```

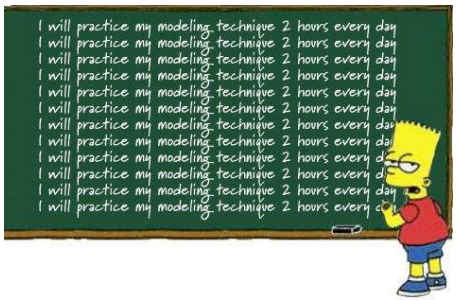
Result



Example `../410-multiple-textboxes`

Workshop

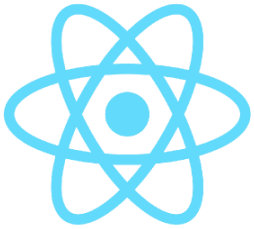
- Own application: create a component containing multiple textboxes. Add them to a new array in the state.
- OR: add textboxes to the VacationPicker in example `../410-multiple-textboxes`
- Add a new country to the list of countries, creating textboxes for every property
- This is only *in-memory* for 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 for VacationPicker, holding an array with visited countries.
const [visitedCountries, setVisitedCountries] = React.useState([]);
```

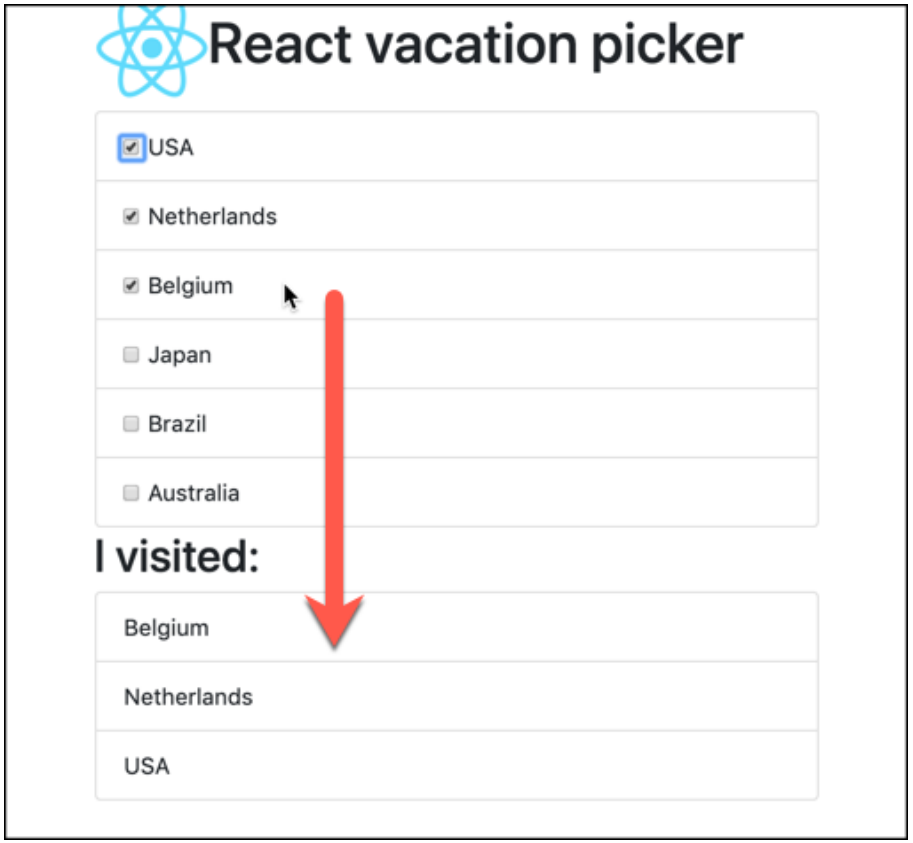
Add to UI

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

Handle logic

```
const checkCountry = (country) => {
    // The country can be only once  in the array of visited countries.
    // if it is already there, it means the box has changed from checked to unchecked and
    // the item should be removed from the array
    if (visitedCountries.indexOf(country) > -1) {
        const newVisitedCountries = visitedCountries.filter(i => i !== country)
        setVisitedCountries(newVisitedCountries);
    } else {
        // it's not there yet, so add it.
        setVisitedCountries([...visitedCountries, country])
    }
}
```

Result

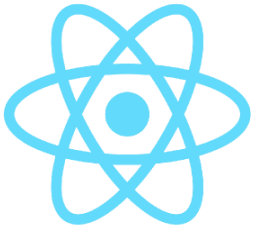


`../examples/420-checkboxes`

Creating radio buttons

Selecting one item in a list of items

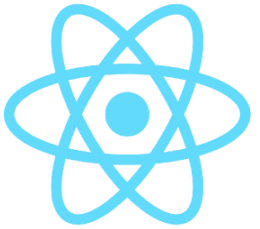
Radio buttons



- 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>
```

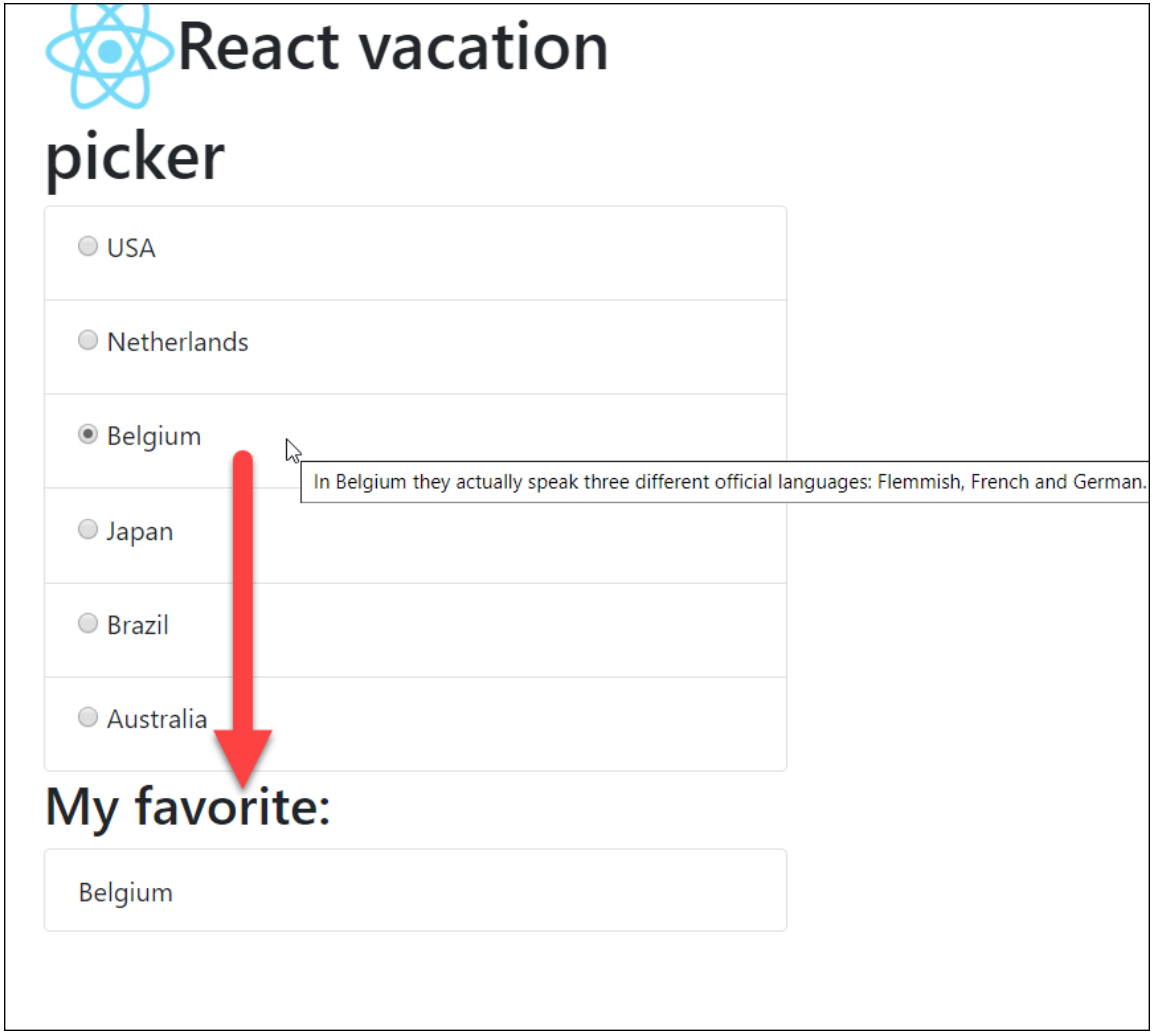
Updating `checkCountry()`



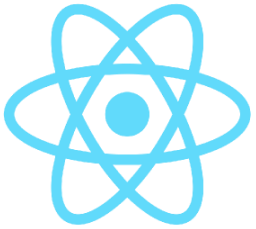
- The `checkCountry()` function is now simpler:

```
const checkCountry = country => {
    // The country now comes from a radio button, so only one
    // at a time can be selected. Easy peasy.
    setVisitedCoutries([country])
}
```

Result

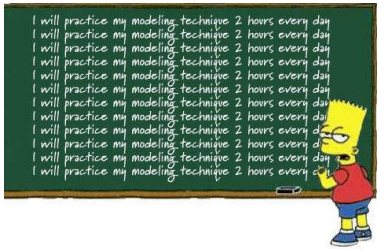

React vacation
picker
USA
Netherlands
Belgium
In Belgium they actually speak three different official languages: Flemmish, French and German.
Japan
Brazil
Australia
My favorite:
Belgium

Workshop

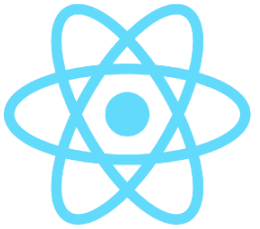


- Create a Todo-component with a list of Todo's
  - Place a checkbox before each Todo item
- If a Todo is marked as completed, add it to an array of completed Todo's.
  - Show this array in the UI
  - Show the completed Todo with a `strikethrough` class in the UI
- Add a new component, showing the same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

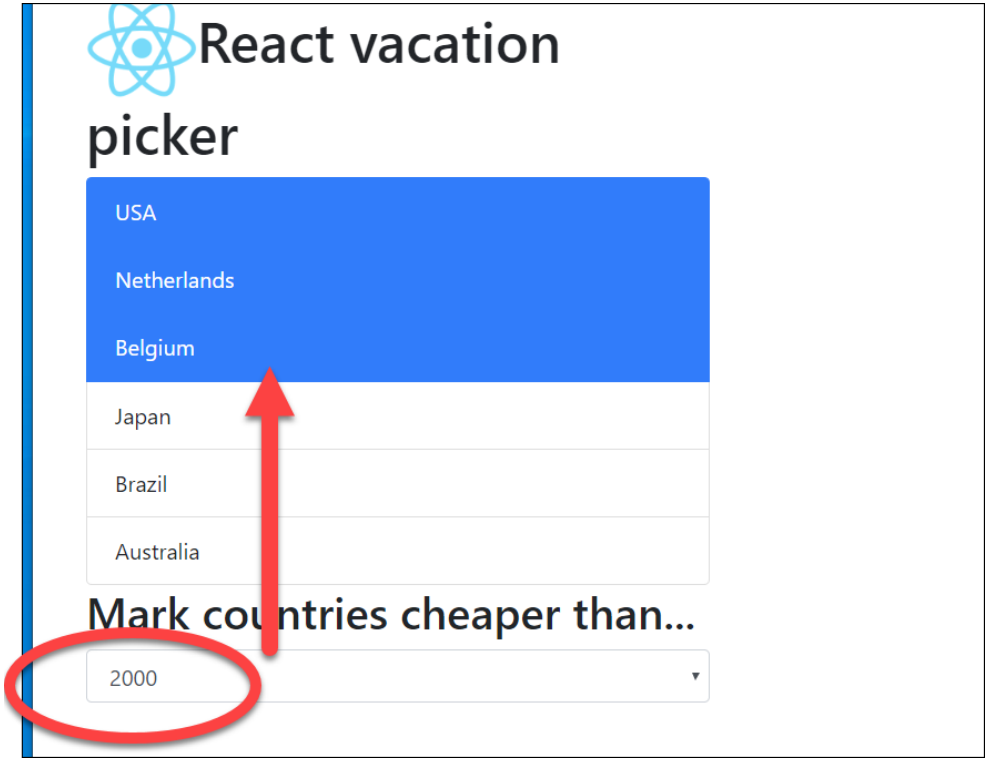
  `../examples/430-radio-buttons`



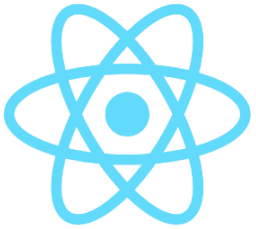
Select/dropdown lists



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

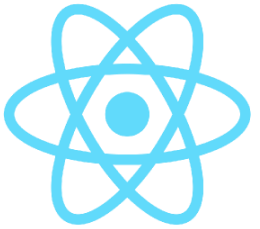


Using `<select>` and `<option>`



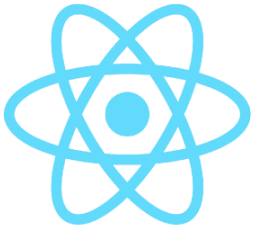
- In a selection list, we also use the `value` and `onChange` handler to read/set it's values
- This is NOT standard HTML. It is created for/by React
- The selected option is set as the `value` for the complete list.
- Selecting a new option, is handled by `onChange`.

1. Creating the selection list



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

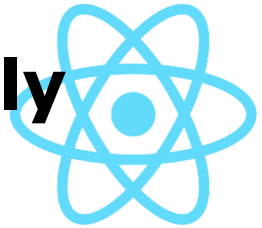
2. Updating the `state`



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.
const [price, setPrice] = useState(1000);
const prices = [1000, 2000, 3000, 4000, 5000];

const updatePrice = (event) => {
    console.log('selected value:', event.target.value);
    setPrice(event.target.value)
}
```

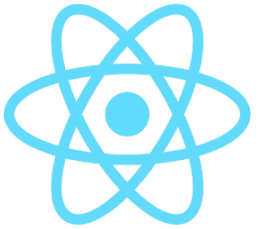
3. Adding/removing classes conditionally



```
<ul className="list-group">
    {props.countries.map(country =>
        <li
            // conditionally add class name.
            // We're using the Bootstrap class .active here,
            className={'list-group-item ' + (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

Usage with React.js

This package is the official replacement for `classSet`, which was originally shipped in the React.js Addons bundle.

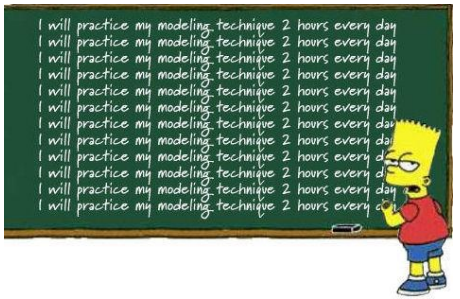
One of its primary use cases is to make dynamic and conditional `className` props simpler to work with (especially more so than conditional string manipulation). So where you may have the following code to generate a `className` prop for a `<button>` in React:

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

export default function Button (props) {
  const [isPressed, setIsPressed] = useState(false);
  const [isHovered, setIsHovered] = useState(false);

  let btnClass = 'btn';
  if (isPressed) btnClass += ' btn-pressed';
  else if (isHovered) btnClass += ' btn-over';

  return (
    <button
      className={btnClass}
      onMouseDown={() => setIsPressed(true)}
```

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 :

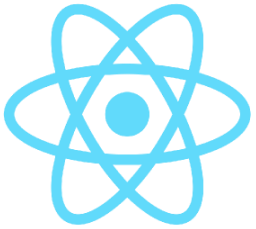
  `../examples/440-selection-lists`



Submitting forms

Sending your forms to a backend

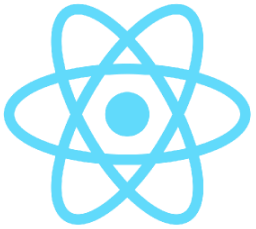
Submitting a form



- Use the `<form>` tag with an `onSubmit()` handler
- Write your custom handler
- You can use props as usual, or use inline event handler

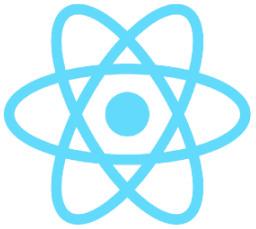
```
<form onSubmit={submitCountry}>
    …
    <input type="submit" className="btn btn-success"
           value="Submit new country"/>
</form>
```

Submitting a country



```
// 2. submitCountry - create a new country and send it to the parent.
// It uses the current value (i.e. the state) of the form input.
const submitCountry = (event) => {
    // 0. prevent sending the form to the server
    event.preventDefault();
    // 1. create a new country, based on the state
    const newCountry = {
        id: Math.random(), // IRL not sufficient!
        name: name,
        capital: capital,
        cost: cost,
        details: details,
    };
    // 2. send the country to the parent
    submit(newCountry);
};
```

Handling a form submission

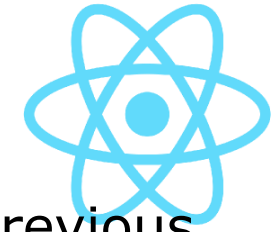


In our case: adding a new country to the state (still just in-memory only!)

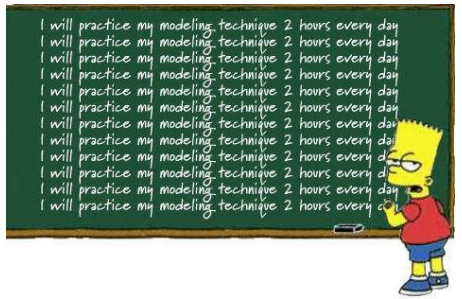
```
<AddCountries submit={(country) => addCountry(country)}/>
```

```
const addCountry = (country) => {
    setCountries([...countries, country]);
    // TODO: POST  the new country to a database
}
```

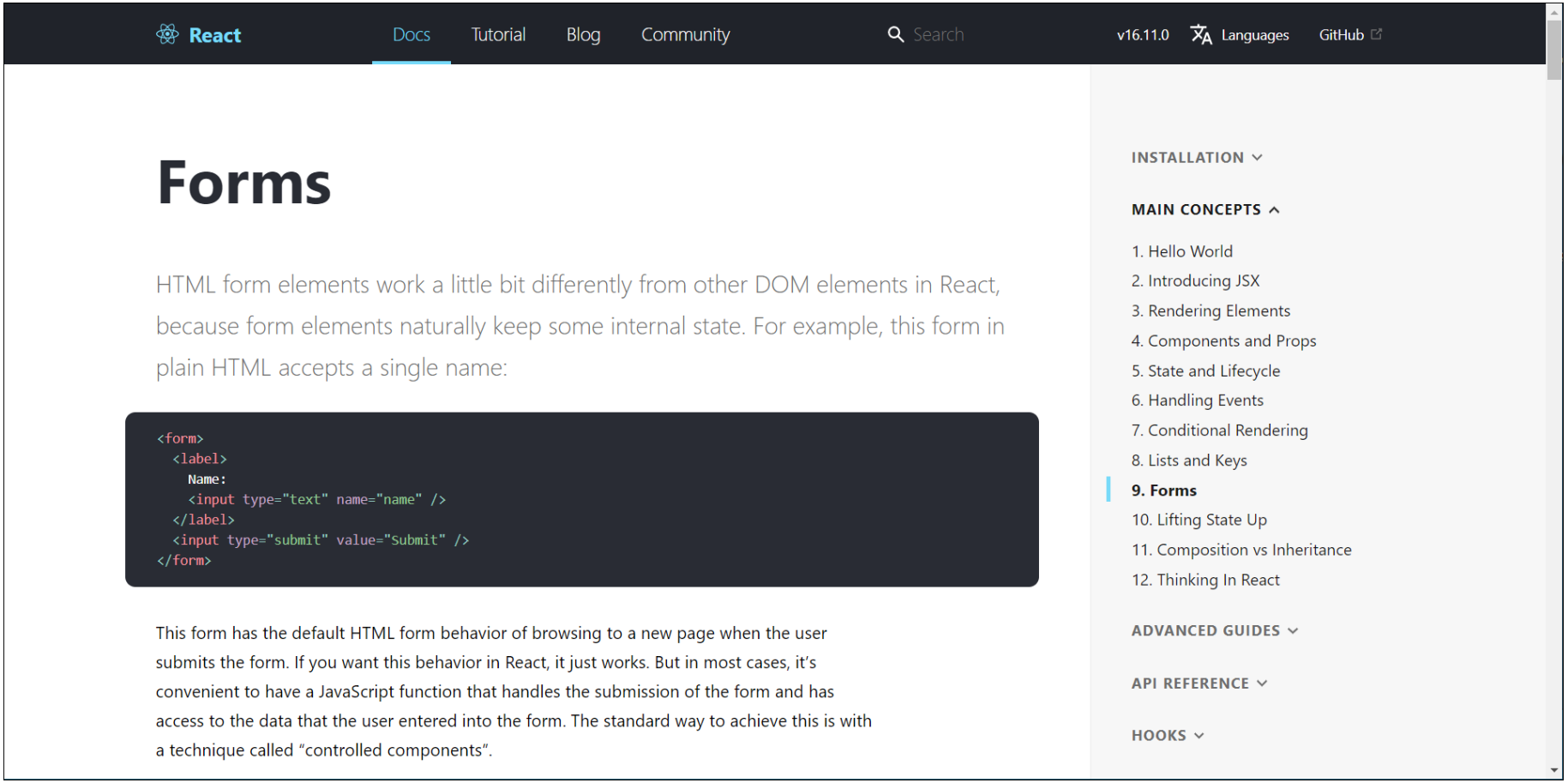
Workshop



- Continu 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`

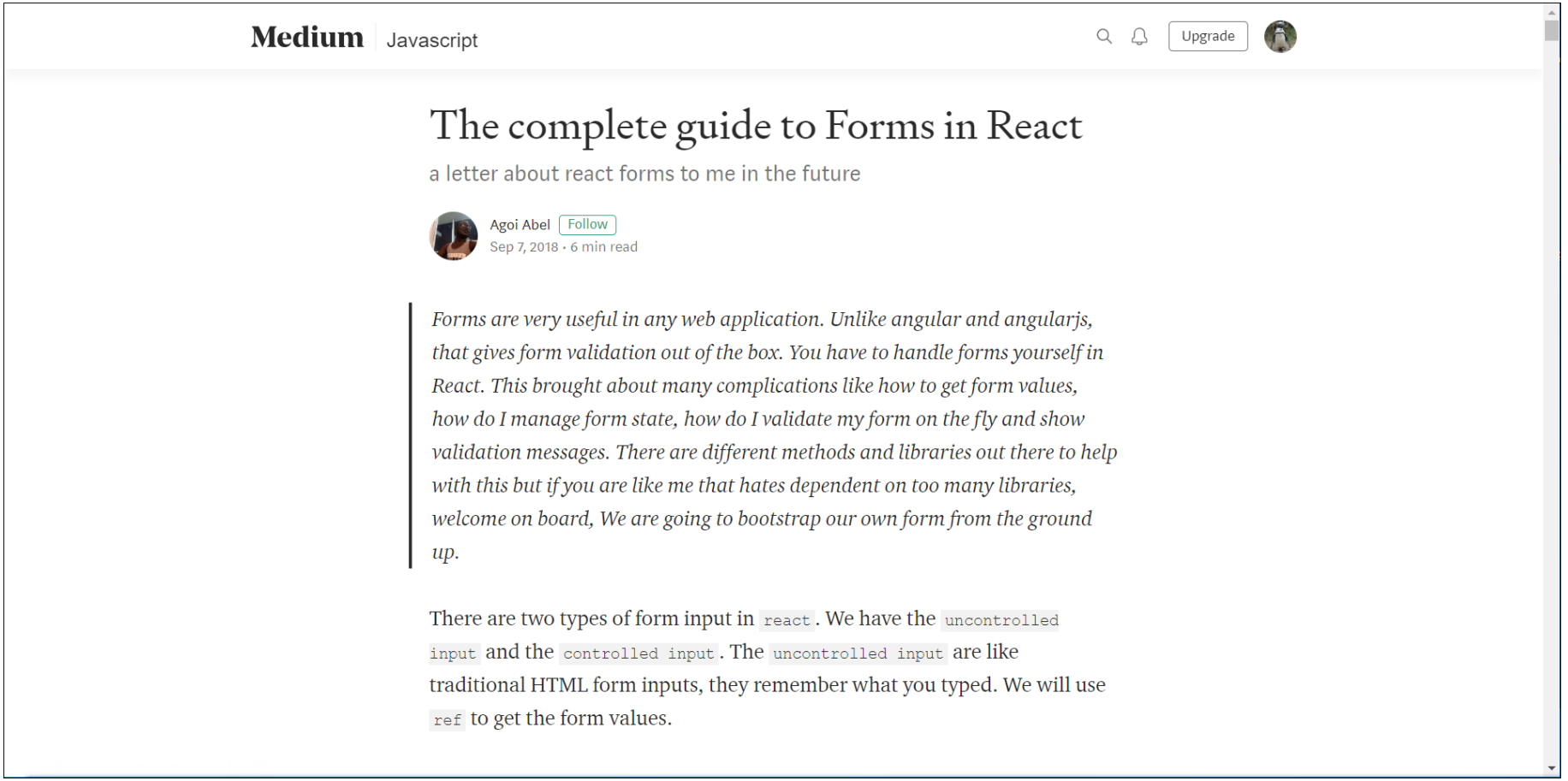


More info



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

Blog article on forms



Medium | Javascript

Upgrade

The complete guide to Forms in React

a letter about react forms to me in the future

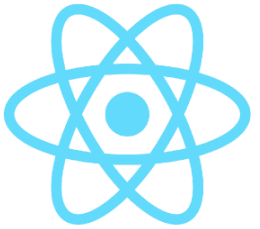
Agoi Abel Follow
Sep 7, 2018 · 6 min read

> *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)