React Fundamentals

Working with data

Peter Kassenaar –
info@kassenaar.com

Working with data

Importing local and external data in your app

Warning in advance!

- We're going to use *local* state here
  - E.g. state inside the component
  - So: statefull components!
  - For learning purposes

- In a real life environment w/ testing:
  - Work as much as possible with Stateless components
  - State in Top Level Components/containers, Context or Redux store
  - Those are more easily tested
  - Test snapshots of the components, with mocked state

Read the warning on the

previous slide again

...

Then continue 😀

Building an app: VacationPicker

- Requirements:
- Loading external data
  - Comes from `.json/.js` file
  - Later the data will be fetched from a 'real' API
- Looping over- and displaying data
- Selecting data – showing details
- Binding images
- Conditionally render pieces of the UI

Creating the data file

```
// ./data/countryData.js - holding an array of country/capital data.
// This of course will come from a real db in the future.
const data = {
   countries: [
      {
         id: 1,
         name: 'USA',
         capital: 'Washington',
         cost: 1250,
         details: 'United States are among the most visited country in the world.',
         img: 'washington.jpg'
      },
      {
         id: 2,
         name: 'Netherlands',
         capital: 'Amsterdam',
         cost: 795,
         details: 'The capital of the Netherlands, Amsterdam, is over 1000 years old.',
         img: 'amsterdam.jpg'
      },
      { …
}
```

Our example: a simple JavaScript object, holding an array with countries and some (fake) data, don't forget to `export` it!

New component(s)

Create a new component, `VacationPicker.js` with the default content

```
// VacationPicker.js
import React from 'react';

const VacationPicker = () => {

    return (
        <div>
            <h1>List of Countries…</h1>
        </div>
    );
}

export default VacationPicker
```

Updating <App />

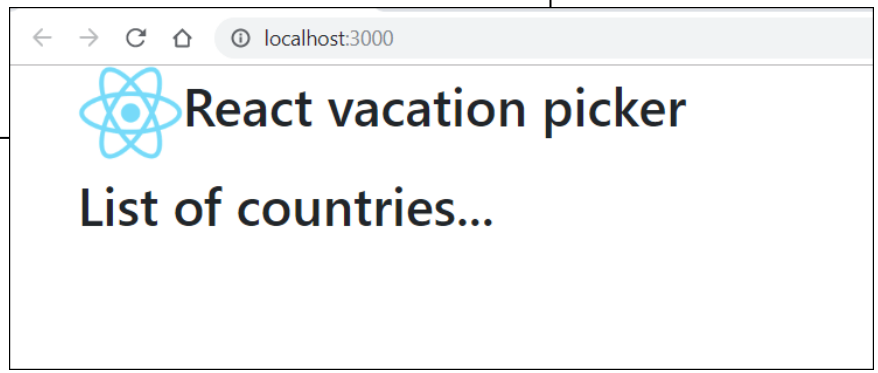
```
// App.js
import React from 'react';

// Child components
import VacationPicker from './VacationPicker/VacationPicker'

// Our parent component - it later holds the state for the child components
const App = () => {

  // Render UI
  return (
      <div className="container">
        <h1>React Vacation Picker</h1>
        <VacationPicker/>
      </div>
  )
}

export default App;
```

Adding the Logo

- Static images must be imported before they can be bound in JSX

```
import logo from '../img/logo-react-small.png'
```

```
…
<h1>
    <img src={logo} alt="react logo" width={80}/>
    React vacation picker
</h1>
<VacationPicker/>
```



Importing the data

- Use default `import` statement for the `countryData.js` file
- Data is made available as a `prop` on the component

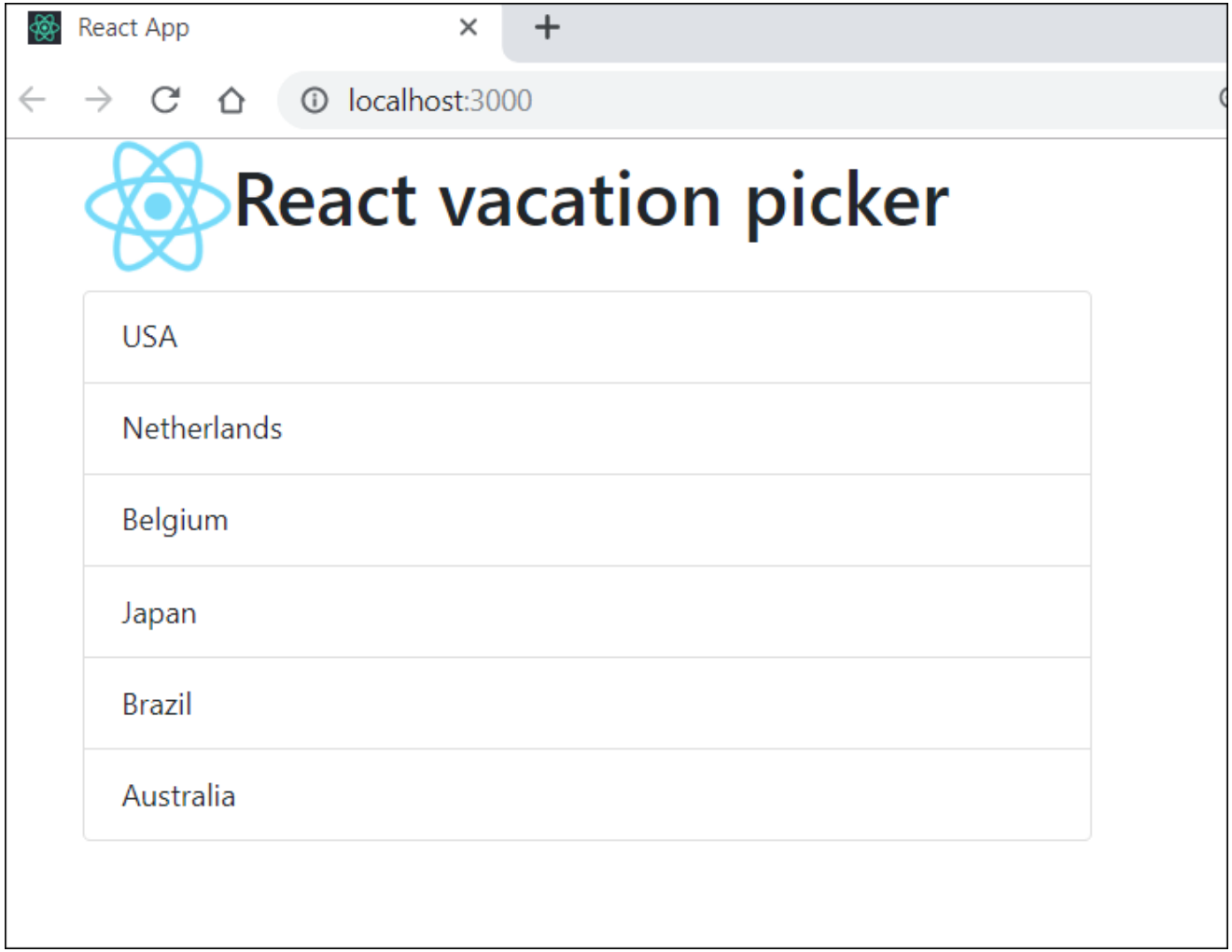
```
// App.js
import countryData from '../data/CountryData’;

const App = () => {
    const [countries] = React.useState(countryData.countries);

    return (
        <div className="container">
            …
            <VacationPicker countries={countries}/>
        </div>
    )
}
```

Binding to data

- Use a default JavaScript `.map()` function to render the data.
- Remember, we are looping over a JavaScript array!
- In React, you can put any JavaScript expression inside curly braces in the JSX
  - Like `{props.countries.map(…)}`
- Inside the expression your render additional UI

```
// VacationPicker.js
const VacationPicker = ({countries}) => {
    return (
        <div>
            <ul className="list-group">
                {countries.map(country =>
                    <li className="list-group-item"
                        key={country.id}
                        id={country.id}
                        title={country.details}
                    >
                        {country.name}
                    </li>
                )}
            </ul>
        </div>
    );
}
```

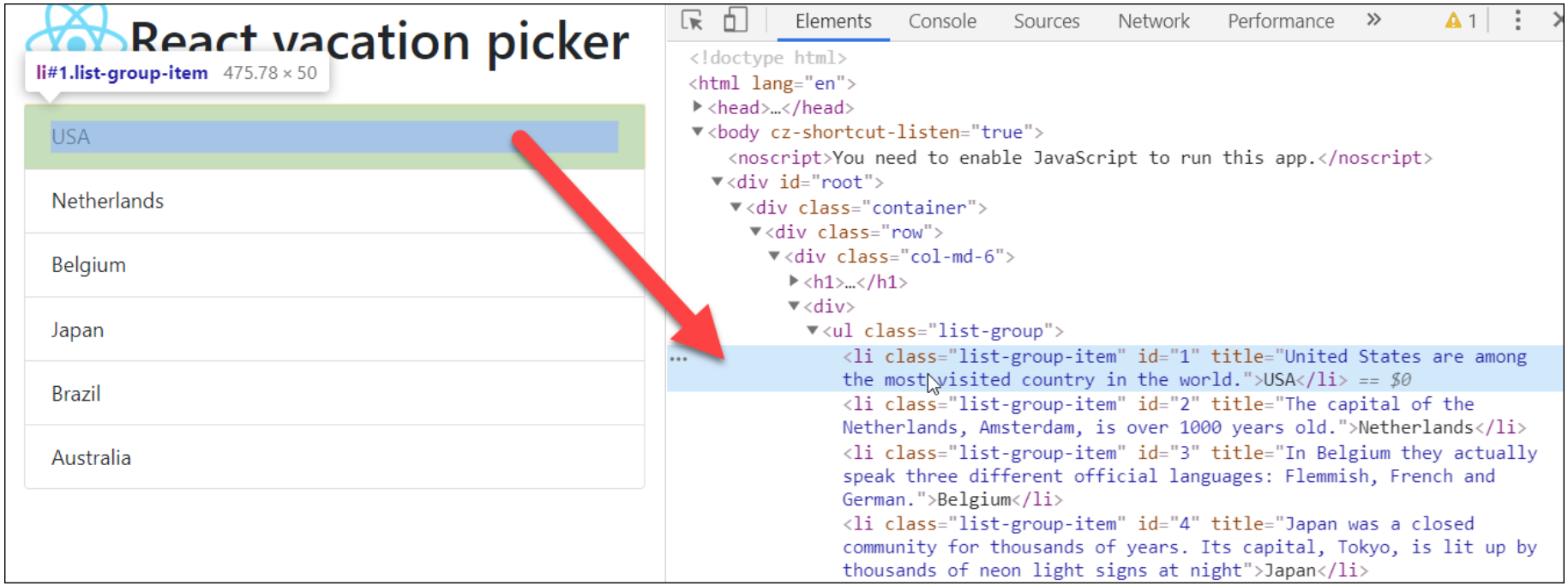
Result


React App
localhost:3000
React vacation picker
USA
Netherlands
Belgium
Japan
Brazil
Australia

Binding to attributes

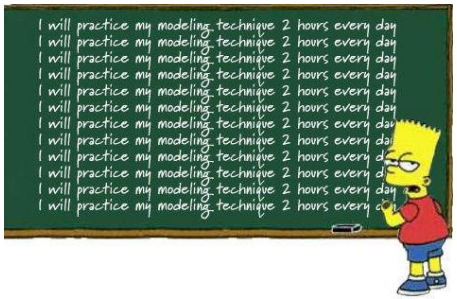
- Note that we are binding data to attributes here
  - `key` – to create a unique React-key for each list item
  - `id` – to create an HTML-`id` for each list item
  - `title` – to create a small popup if one hovers over the item
- In React, you use *single curly braces* to assign attribute values
  - `id={country.id}` - Good
  - `id="country.id"` - Bad
  - `id="{country.id}"` - Bad

Result in the DOM



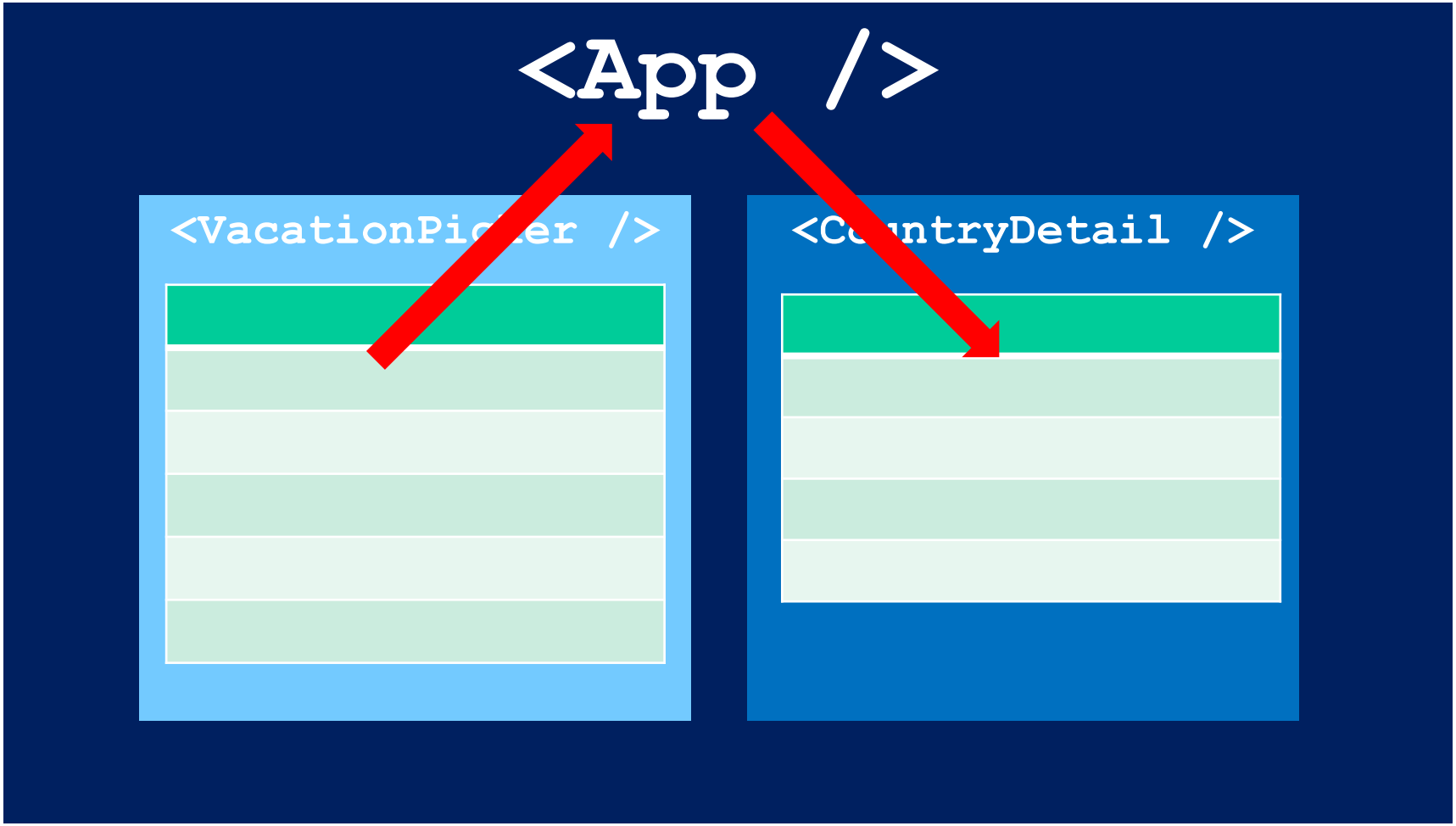
Workshop

- Add a data file to your application.
  - You can use `countryData.js` as an example, or build your own data file
  - Use for instance `CustomerData.js` as a blueprint and add your own data
- Import the data file to your application
- Show its contents in a new component, or inside `VacationPicker`, using `.map()`
- Create a `key` binding for your data
- Example: `../200-data-list`



Selecting a specific country

Passing details to a specific component


<App />
<VacationPicker />
<CountryDetail />

Selecting a specific country

1. Create a `<CountryDetail />` component
2. Define new state property on `<App />`
   - we call it `currentCountry`
3. Pass state to `<CountryDetail />`
4. Write method to set new `currentCountry` on click
5. Update `<VacationPicker />` to transfer the clicked country up to `<App />`

1. Creating CountryDetail

Create a component as usual.

```
// CountryDetail.js - show details of a specific country
import React from 'react'

// A pure presentational component
const CountryDetail = ({country}) => {
    return (
        <div>
            <h2>{country.name}</h2>
            <ul className="list-group">
                <li className="list-group-item">
                    id: {country.id}
                </li>
                …
             </ul>
        </div>
    )
}

export default CountryDetail
```

Render all items from the passed in `country`

2. Define state for current country

Design decision: the first country in the list is by default the selected country

```
// Our parent component - it holds the state for the child components
const App = () => {

    const [countries] = React.useState(countryData.countries);
    const [currentCountry, setCurrentCountry] = React.useState(countryData.countries[0]);
    …
}
```

3. Pass the state to `<CountryDetail />`

Render the component in the UI of `App.js` like normal

```
<div className="container">
   …
     <div className="col">
        <CountryDetail country={currentCountry}/>
     </div>
</div>
```

4. Write method to update the state

```
const selectCountry = country => {
    const newIndex = countries.indexOf(country);
    setCurrentCountry(countries[newIndex]);
}
```

`selectCountry()` will be called if a specific country in the list is clicked. So we need to pass this function as a prop

Calling `selectCountry()`

```
<VacationPicker
    select={country => selectCountry(country)}
    countries={countries}/>
```

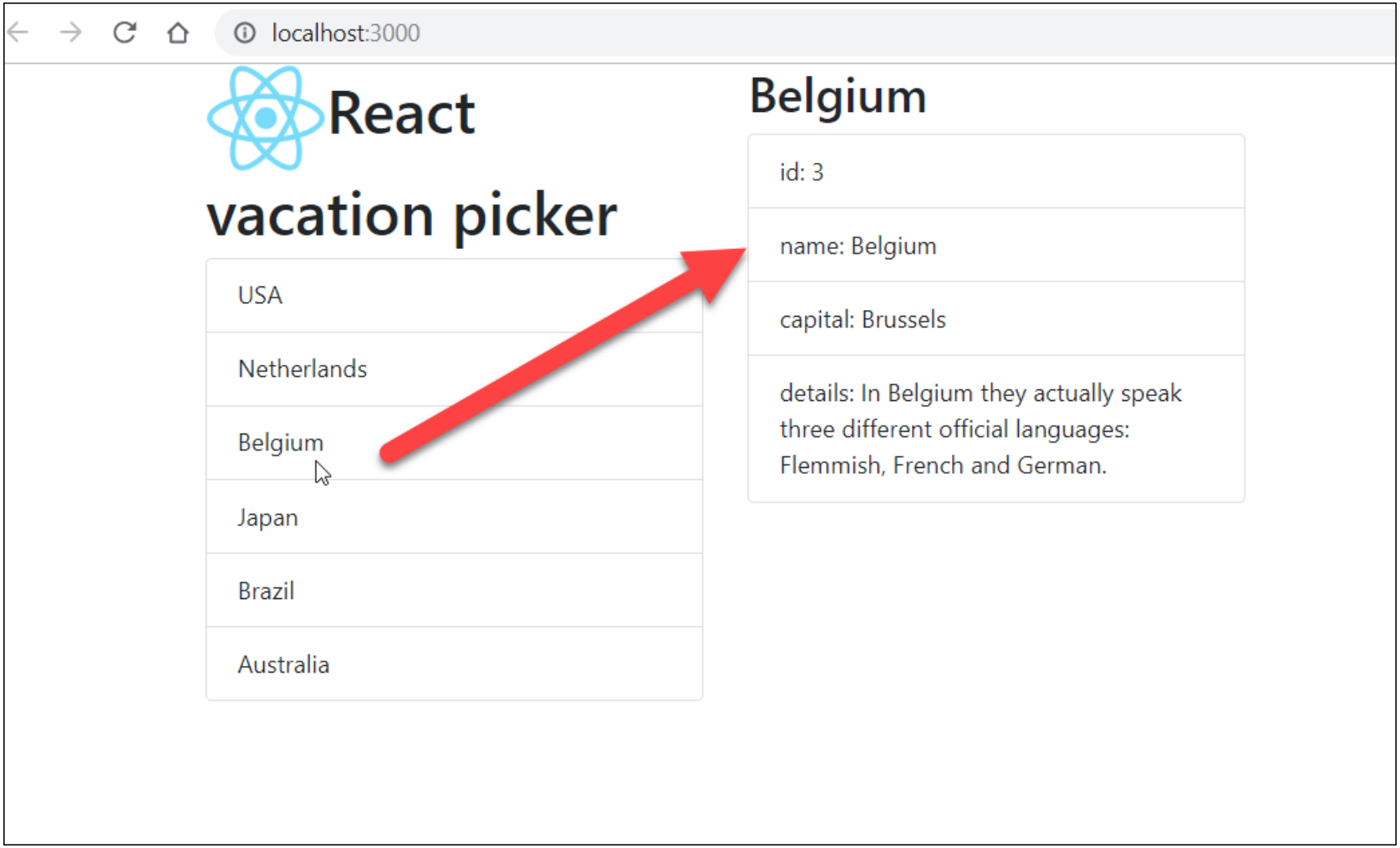
Note: the `country` that is passed to `selectCountry()` is coming from the `<VacationPicker />`.

So we *need* to define this as a parameter for the incoming function call

5. Update `<VacationPicker />`

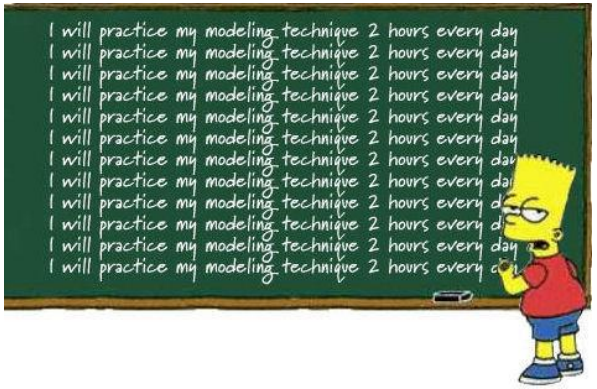
```
const VacationPicker = ({countries, select}) => {
    return (
        <div>
            <ul className="list-group">
                {countries.map(country =>
                    <li className="list-group-item"
                        key={country.id}
                        id={country.id}
                        title={country.details}
                        onClick={() => select(country)}>
                        {country.name}
                    </li>
                )}
            </ul>
        </div>
    );
}
```

Result


localhost:3000
React
vacation picker
USA
Netherlands
Belgium
Japan
Brazil
Australia
Belgium
id: 3
name: Belgium
capital: Brussels
details: In Belgium they actually speak three different official languages: Flemmish, French and German.

Workshop

- Create a detail view for your own app.
  - If an element/data is clicked, show details in the UI
  - Remember to pass data to the detail component
- Ready made example: `../210-data-detail`
- You can also work from this example and create a *new* detail component and pass for instance only `country.name` as an exercise



Rendering images

Showing dynamic images in the UI

Static images

Like the logo: import first, then use as a variable reference

```
import logo from '../img/logo-react.png'
import background from '../img/background.jpg'
```

```
<img src={logo} alt="react logo" width={80}/>
<img src={background} alt="background" />
```

Dynamically binding to images

- React can not simply interpolate the name of an external resource and re-use it for binding
  - For instance, the `src` attribute of an image.
- So this is invalid:

```
<img
    className="img-fluid"
    src={'../../img/countries/' + country.img} alt={country.name}/>
```

Invalid!

WebPack to the rescue

- Because Webpack builds JavaScript strings of everything, it needs to be able to determine the location of the requested file at compile time.
- Use `require()` that returns a string with the correct location:

```
<img
    className="img-fluid"
    src={require('../../img/countries/' + country.img)} />
```

Correct!

Using ES6 string interpolation

- You can also use ES6 string interpolation, like so:

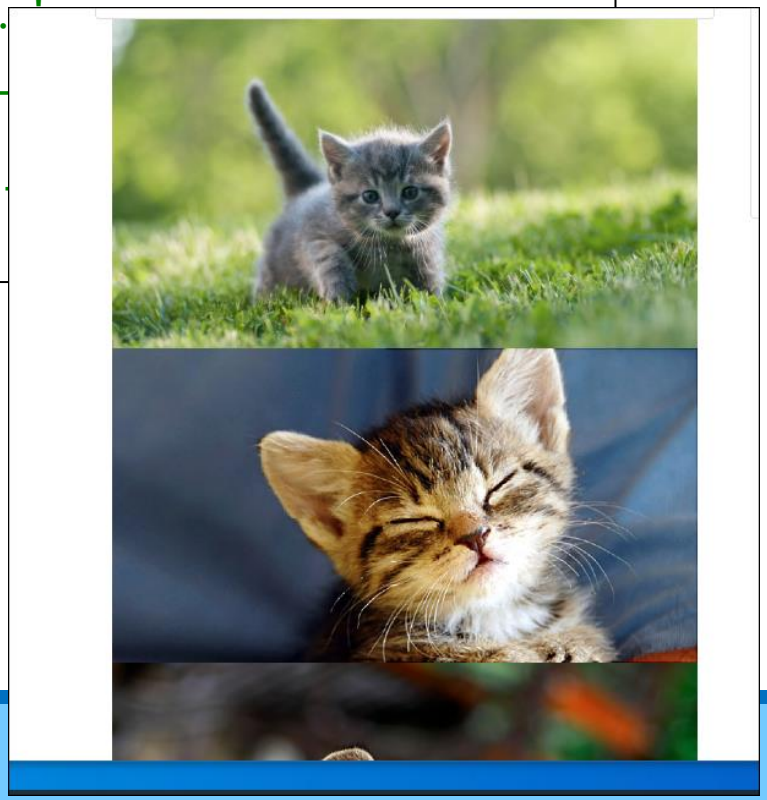
```
<img
    className="img-fluid"
    src={require(`../../img/countries/${country.img}` )} />
```

Correct!

Full URL's

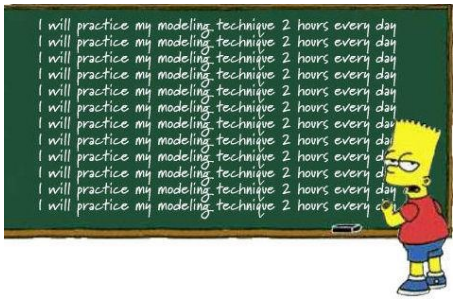
When the image is a fully qualified URL, directly binding inside `<img>` is correct

```
cats = [
    'https://www.vets4pets.com/siteassets/.
    'https://img.webmd.com/caring_for_your_
    'https://images.unsplash.com/…',
    'https://imgix.bustle.com/uploads/safe-
];
```



Workshop

- Own application: add images to your data and render them dynamically in the app
- Ready made example `../220-image-binding`
- Optional: create a new component, holding an array of static images (fully qualified URL)
  - Render them in a loop to the UI
  - (like in the previous slide)



Conditional rendering

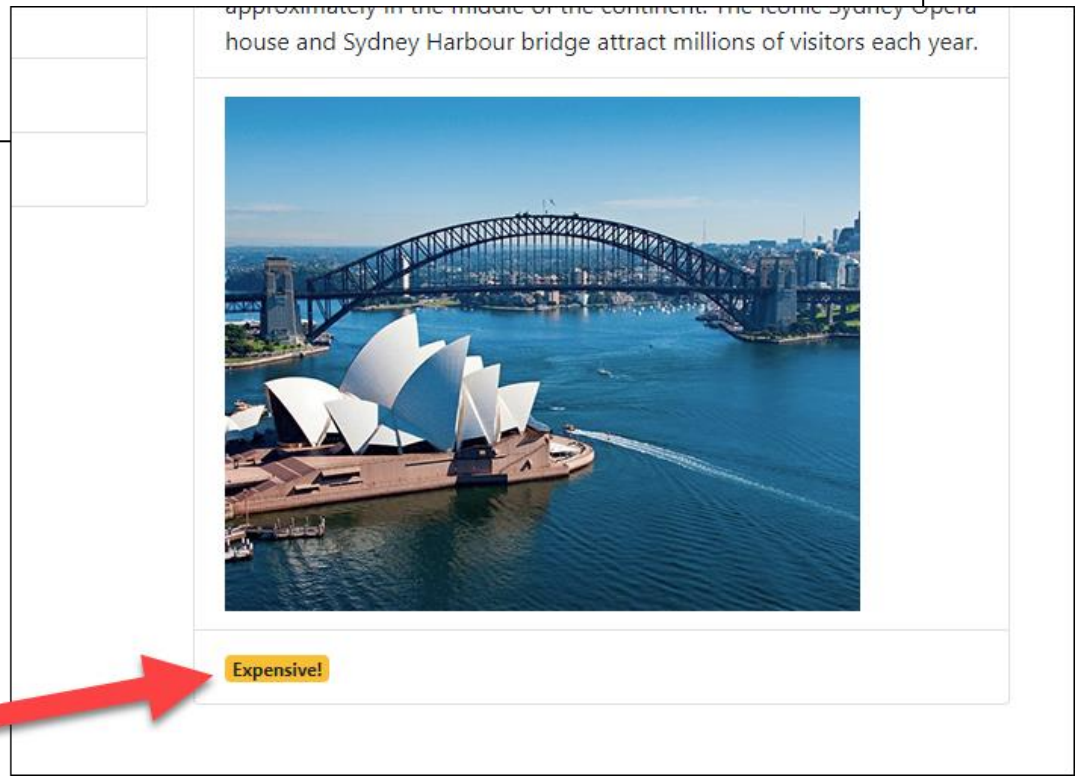
Only show some UI if a certain condition is met

Adding conditions to the UI

- We want to show a badge if a destination is `expensive`
  - Decision: "a destination is expensive if it costs more than 4000"
- Remember: you can just write JavaScript expressions between `{ … }`
- We're using a ternary statement `? … :…` here.
- React does *not* have a construct like `v-if`, or `*ngIf`, like Vue and Angular

Writing a conditional statement

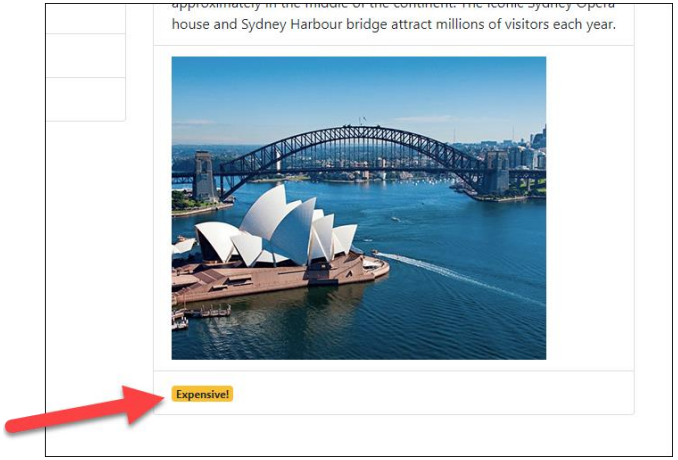
```
{country.cost > 4000 ?
    <li className="list-group-item">
        <span className="badge bg-warning">Expensive!</span>
    </li>
    : ''
}
```



Or: use the Logical AND operator `&&`

```
{/* Using the logical AND operator to render items conditionally . */}
{country.cost > 4000  &&
    <li className="list-group-item">
        <span className="badge badge-warning">Expensive!</span>
    </li>
}
```

The AND operator returns `true` if both operands return `true`. Pieces of UI are *always* truthy, so if the first expression is `true`, the UI is rendered.



Results are visually the same.

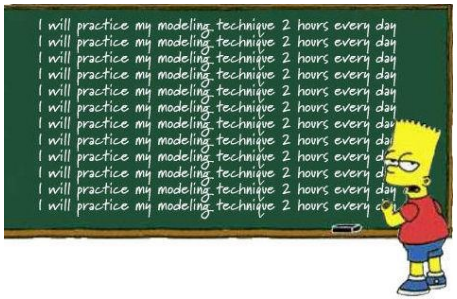
Use whatever floats your boat.

Workshop

1. Create an ‘on sale’ label, if a destination is cheaper than 1000
2. Show the `expensive` and `on sale` badges directly in the list
3. Create a button that shows/hides the country details

- Example `../230-conditional-rendering`

1. Optional: create an `OnSale` and/or `Expensive` *component*
2. Optional: create a `favorite` property on the data model.
   - Inside the detail view, users can mark the item as favorite.
   - The status should show in the list/overview
   - Possible solution: `../workshops/30-binding-favorite`
   - But first try it yourself!



Checkpoint

- You can import static data in your application
- You know how to loop over the data using `.map()` and render it in the UI
- You can select data from the master view and pass it to a detail component
- You can render dynamic images using `require(…)`
- You know how to render pieces of the UI conditionally