Framework Training

REACT

Exercise NEWYORK

August 2019

- This exercise practices **managing state** in the containing component and passing state down as a prop to a presentation component.

Installation

- Install and run the starter version of the project.

```
npm install
npm start
```

Review the existing project.

- Look at a snapshot of the finished exercise at **src/help.jpg.**
- The user can select from 0-4 places to visit in New York.
- Once a place has been selected, its name appears as a **green ticket** in the basket below the line.
- The selected place then becomes **visually faded**.
- Once selected, a place **cannot be selected again**.
- Each place will be a **stateless** component.
- The **visible appearance** of the place and **its behaviour** (whether it is selectable) will be passed in as a prop from the containing component.
- The starter version of the project uses **composition**. The NewYork component render method maps over an array of objects creating instances of a stateless Place component.

The public folder at runtime

- At runtime, the transpiled version of your code will run from **public/index.html**.
- It will load images from the **public/ny** folder and will read JSON data from the **public/json** folder.

Read JSON data using Fetch

- Replace the hard coded array of objects in the **constructor** with a Fetch call to read **public/json/newyork.json**.

```
// Simplified state. Hard coded data removed.
this.state = { newyork:[],tour:[] };
```

- Remember to do this work in the **componentDidMount** LifeCycle method.

```
componentDidMount() {

fetch( this.props.path )
    .then( response => response.json())
    .then( data => this.setState( { newyork:data }))
    .catch(e => console.log(e))
```

}

Buy tickets

- Create a **buy ticket** method that logs the name of the place to the console.

```
buyTicket = name => console.log(name);
```

- Pass this method down as a **prop** to the Place component.
- *Adding an onClick attribute directly onto the Place instance will not work.*

```
    <Place  key={photo} name={name}
```

photo={photo} buyTicket={this.buyTicket}>

- Use **destructuring** on the Place component function signature.
- **Call** the method when the user clicks on this Place.

```
let Place = ({name,photo,buyTicket}) =>
<article onClick={buyTicket}>
```

- Clicking on a place executes buyTicket but passes the **event** object to the method.
- We could work out the name with an expression like **e.currentTarget.textContent**.
- But we already know the **name** of the place. Pass this as a parameter to buyTicket.

```
 onClick={buyTicket( name )}
```

- This does **not work** as intended. The **buyTicket(name)** is executed immediately.
- Change the onClick expression to wrap the function call in an **anonymous inline ES6 arrow function**.

```
onClick={ () => buyTicket(name) }
```

- We want to add the selected place name to this.state.basket.
- But we **should not directly change state**. React updates state async in batches.

```
// Do not do this.
this.state.basket.push( name )
```

- Instead we should make a copy of the basket and add the name to it. Then pass this copy to **this.setState**.

```
// Use spread operator to make copy and add to it
let copy = [...this.state.basket,name]
// Pass the copy to setState.
this.setState( { basket:copy } );
```

- The code to map over and display this.state.basket already exists.

```
basket.map((name) =>
<section className="basket" key={name}>{ name }</section>
```

Stop duplicates in the basket.

- The **buyTicket(name)** clickHandler in the Place component should not run if that place-name is already in the basket.
- We want to avoid introducing **state** in both the Place and NewYork components.
- We can pass a **boolean prop** down to Place to decide if each place is selectable.

```
<Place selectable={true}>
```

- Add **destructuring** in the function signature.
- Change the onClick expression to read this boolean prop.

```
let Place = ({name,photo,buyTicket,selectable}) =>
<article onClick={ selectable ? null : () => buyTicket(name) }>
```

- Create a boolean method **basketContains** which returns true if the basket already contains a specific place name.
- The Javascript **includes** method returns true if it finds a specific value in an array.

```
let fruit = ["apples","pears"];
fruit.includes("apples"); // returns true
```

- Implement includes in basketContains:

```
basketContains = name => this.state.basket.includes(name) ;
```

- Change the prop to call this method.

```
selectable={this.basketContains(name)}
```

Make selected places look faded

- Each place should look **faded** once it has been selected.
- We want to **conditionally apply this CSS class**.

```
.selected{
    cursor: default;
    opacity: 0.4;
}
```

- Create an expression which **conditionally applies** the selected CSS class based on the **selectable prop**.

```
<article className={selectable ? "selected" : null} ....>
```

Remove tickets from the basket

- If the user clicks on a **green ticket**, it should be removed from the basket.
- The corresponding place should then become **available/selectable** again.
- Add a **click handler** to the basket.
- Code the remove method so that it **filters** in only basket items that do not match this name.
- Note, filter is a **pure** function. It returns a new array.

```
    onClick={ () => this.remove(name)}

remove = name => {
let basket = this.state.basket.filter( n => n !== name )
this.setState( { basket:basket })
}
```

Empty basket

- Implementing the empty button is straightforward because all **state is managed in one place**, the NewYork component.

```
<p className="button" onClick={this.empty}>Empty</p>

empty = () => this.setState( { basket:[] })
```