Working with Datasets

DEVELOPING APPLICATIONS USING REACTJS

Objectives

- To be able to add to a dataset from within a web page
- To be able to read in a dataset to display on a page
- To be able to request and receive data from a RESTful service

Adding to a Dataset

- Many applications require users to fill in a form
- The form data needs to be collected and then processed somewhere
- React can retrieve form data and then push it into an existing state
  - This can then be outputted as JSON to wherever it is needed
- We have already seen how to do this in the previous chapter

3

Data Sources

- React is for producing front end components
- Need to be able to convert data from data sources into state and/or props
  - E.g. Displaying a list of items in a table from a HTTP search response
- Can convert data source into or create an array
  - Array can then be stored in state
  - `render()` function can then use JavaScript map function to return state array as modified array of Components which are then displayed on the screen
- map() works in the following way
  1. For every item in the array, takes item itself and its index
  2. Creates a new anonymous function that receives item and index and returns a value of the modified data

4

Using a dataset - map() example

- Imagine we have an array called people that stores a person object holding an ID, their first name and their country of origin
- This can be stored in the component's state using the following code:

```
…
    this.state = { people };
    // If key is same name as value, only key name is required
…
```

- State can then be used in the render function to produce JSX code for each person in the array through the `map()` function

```
…
  {this.state.people.map((person, index) => (
    <p>Hello, {person.name}, you're from{person.country}</p>
  ))};
…
```

5

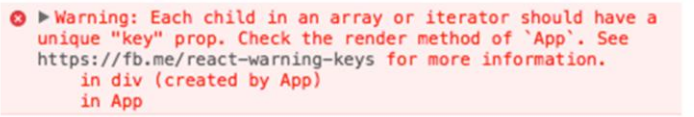
Remember: => (called fat-arrow notation) is ES6's way of declaring a return from an anonymous function.

Multiple HTML elements could be used here as long as they have a wrapping element, eg.

```
…
    {this.state.people.map((person, index) => (
        <div>
            <p>Hello, {person.name}!</p>
            <p>You're from {person.country}.</p>
            <hr />
        <div>
    ))};
…
```

Array/Iterators and Keys

- Running the code in this way would produce a warning on the console



- React likes each every top-level item printed by a loop to have a KEY attribute that identifies it uniquely
  - Called RECONCILIATION
  - Helps React identified which items have been modified
  - Best practice to select a string that uniquely identifies a list item amongst its siblings (no need to be globally unique)
    - Most often ID from the dataset
    - Can be index of array as a last resort
  - Keys should be kept on the array element produced rather than the root element

6

For more information on RECONCILIATION see: https://facebook.github.io/react/docs/reconciliation.html

Arrays/Iterators and Keys

- Dealing with the warning is important for more advanced apps
- The key attribute should be added to the wrapping element of the return of the anonymous function

```
…
   {this.state.people.map((person, index) => (
     <p key={person.id.toString()}>Hello, {person.name}, you're
       from{person.country}</p>
   ))};
…
```

7

If multiple HTML elements used, the key should be included in the wrapping element, eg.

```
…
    {this.state.people.map((person, index) => (
        <div key={person.id.toString()}>
            <p>Hello, {person.name}!</p>
            <p>You're from {person.country}.</p>
            <hr />
        <div>
    ))};
…
```

Sub-Components

- Data can be passed from state of a parent component to a child using props
  - Using a sub-component - change code in parent **`render()`** function

```
…
   {this.state.people.map((person, index) => (
     <Person key={person.id} name={person.name}
       country= {person.country} />
   ))};
…
```

  - Create a Person Component as a class

```
export default class Person extends React.Component {
    render() {
         return (
                  <p>Hello, {this.props.name}! You're from
    {this.props.country}</p>
         );
   }
}
```

8

Note that the key attribute stays within the parent as this is the wrapping component

Functional Components

- Person could have been written as a functional component:

```
const Person = (props) => (
  <p>Hello, {props.name}! You're from {props.country}</p>
);

export default Person;
```

  - The functional component can exist in a file by itself or as part of another
  - If it is to be used in class in another file, the export statement is needed
  - This would be the recommended way to achieve this

9

RESTful APIs

- Default data source for most Web and mobile applications
- REST stands for Representational State Transfer
  - Lightweight
  - Maintainable
  - Scalable
- Not dependent on any protocol
  - Most use HTTP or HTTPS as the underlying protocol
- Can be made to provide data in JSON (desirable for JS applications) or XML

10

React and RESTful APIs

- Makes use of Fetch API and Promises
  - fetch() call returns a Promise
    - Promises can resolve or reject
      - Resolve allows for data to be returned and then passed to state (if desired)
- Make initial requests for data in **componentDidMount** lifecycle method
  - Called after component renders so placeholder can be used until data is returned from the source

```
componentDidMount() {
    fetch(`http://someRESTfulAPIURL.com`)
       .then(response => response.json())
       .then(people => this.setState({
           people
    }));
}
```

11

A Promise is a placeholder for asynchronous data that will be available immediately, some time in the future or not at all. Each .then passes the result of the previous action as the argument for the next.

For more information on Using Fetch see https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

For more information on Promises see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

React and RESTful APIs

- ES2017 standard introduced async and await – returns a Promise still
- Allows removal of the chaining of then from promises
- Code on previous slide could have been rewritten as:

```
componentDidMount() {
        this.getData();
}

async getData() {
        let response = await fetch(`http://someRESTfulAPIURL.com`);
        let people = await response.json();
        this.setState({
                people
        });
}
```

12

For more information on async/await see: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Objectives

- To be able to add to a dataset from within a web page
- To be able to read in a dataset to display on a page
- To be able to request and receive data from a RESTful service

Exercise time!

- EG08 – Working With Datasets

14