Week 3 Workshop

React Course

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes
Check-In	10 minutes
Review	60 minutes
Task 1	40 minutes
BREAK	15 minutes
Tasks 2 & 3	90 minutes
Check-Out	15 minutes

Check-In

- How was this week? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- You must complete all Exercises before beginning the Workshop Assignment. We will review the Exercises together next, along with this week's concepts.

Week 3 Recap – Overview

New Concepts You Learned This Week

• Object Destructuring • Presentational vs Container Component Types • Functional Components	• Virtual DOM • React Router • Router Parameters • Single Page Applications

Next slides will review these concepts,
along with most (not all) of the Exercises you did this week.

JavaScript Object Destructuring

- Destructuring assignment syntax – new in ES6, can be used for arrays and objects, easier way to extract properties from an object and set them as variables
- Examples:

```
const card = {
    suit: "Diamonds",
    number: 12
};
```

```
const {suit, number} = card;
```

```
const suit = card.suit;
const number = card.number;
```

```
function logCard({suit, number}) {
    console.log(suit, number);
}
logCard(card);
```

```
function logCard(card) {
    console.log(card.suit, card.number);
}
logCard(card);
```

Component Types: **Presentational vs Container**

One set of informal classifications of component types:

- Presentational
  - Front end concerns:
    Renders view based on received props, repository for markup & style
  - Little to no local state
- Container components
  - Back end concerns: storing/updating app state, handling data
  - Passes props to presentation components
- Think about this separation of concerns when designing your React app
- Not a dogmatic rule - you do not have to follow this rigidly

Functional Components

- A.k.a. function components
- Simpler than class components –
- Use when you don't need state or lifecycle methods
- No constructor, no render()
- Simply functions that:
  - Accept a single argument: props (sometimes destructured)
  - Return a single top-level JSX element
- Example:

```
function Hello(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- You can also write functional components as an arrow function:

```
const Hello = (props) => {
  return <h1>Hello, {props.name}</h1>;
}
```

`Remember: Props` is used without the ***`this`*** keyword in functional components. In class components, you must use `this.props`

React Virtual DOM

- Core concept of React
- Not unique to React, used in Vue.js and elsewhere
- Lightweight representation of the browser DOM kept in memory by the app
- Faster & easier to update – think of a model of a house vs the house
- ReactDOM library handles 'reconciliation' - syncs browser and virtual DOMs, you do not update browser DOM directly
- Compares previous state to current and only updates the browser DOM for node branches where there have been changes, leaves alone branches without changes, batch updates and other ways to make updates more efficient

React.Fragment

- <React.Fragment> or <>
  - React requires you to return/render only ONE element/node (which can have multiple children) from a component.
  - If you want to wrap multiple children but don't have a need for a new node, you can use a React Fragment, which does not create an extra node.
  - Shorthand version <></> is newer and not all tools support it yet, React.Fragment is safer for now.
  - Examples (longhand & shorthand forms):

```
return (
  <h1>Header 1</h1>
  <h2>Header 2</h2>
)
```

```
return (
  <React.Fragment>
    <h1>Header 1</h1>
    <h2>Header 2</h2>
  </React.Fragment>
)
```

same

```
return (
  <>
    <h1>Header 1</h1>
    <h2>Header 2</h2>
  </>
)
```

React Router – <Switch> <Route> <Redirect>

You may recall `Switch/Case/Default` statements in regular ("vanilla") JavaScript – same idea.

- `<Switch>` groups routes together, goes through each one until right route is found
- `<Route>` sets up route path and components for that path
- `<Redirect>` sets default route if none are found
- `exact` looks for exact match
- Use `component` when you're just routing to a particular view as in HomePage below, use `render` along with an inline arrow function when you need to pass in props, as with Directory below

```
<Switch>
    <Route path='/home' component={HomePage} />
    <Route exact path='/directory' render={() => <Directory campsites={this.state.campsites} />} />
    <Redirect to='/home' />
</Switch>
```

React Router: <Link> and <NavLink>

- `<Link>` element creates links to a route, will render as `<a>`
- `<NavLink>` creates a special link, attaches *active* class to link when it matches the current location

```
<li><Link to='/home'>Home</Link></li>
<li><Link to='/directory'>Directory</Link></li>
<li><Link to='/aboutus'>About</Link></li>
<li><Link to='/contactus'>Contact</Link></li>
```

```
<NavItem>
    <NavLink className="nav-link" to="/home">
        <i className="fa fa-home fa-lg" /> Home
    </NavLink>
</NavItem>
```

React Router Parameters

- <Route> component will automatically pass to its component a built-in prop object with a property called `match`, which itself is an object with a property called `params`.
- In <Route> path, use `:` to specify a dynamic path, e.g. `path=directory/:campsiteId'`
- `campsiteId` above is a dynamic route parameter we can then access via `match.params`
- Example: if user went to URL <your site>/directory/2, then `match.params.campsiteId` would be 2 – you can then use this to get the right campsite information.

Exercise Review: React Router Parameters

- In this exercise, you:
  - In Directory, added `<Link>`s that use template literals to dynamically make each campsite into a link, e.g. /directory/0, /directory/1:
    `<Link to={`/directory/${campsite.id}`} >`
  - In Main, added Route path using dynamic route **parameter** – the parameter follows a colon, you can name it as you like:
    `<Route path='/directory/:campsiteId' component={CampsiteWithId} />`,
  - Implemented `function campsiteWithId({match})`
    - Uses `match.params.campsiteId` to access the parameter – e.g. if the route is /directory/2, then match.params.campsiteId contains 2.
    - Returns CampsiteInfo for the right campsite (passing number stored in match.params.campsiteId as props to CampsiteInfo).

This way you don't need to hard code the route, automatically creates the CampsiteInfo view for the campsite you click on – even if you add, remove, or reorder the campsites

Code Challenges & Quiz

- If there is time remaining from the hour allotted for review, and any of the Code Challenges have not been reviewed, or there are questions about the Quiz, take this time to go over them.

Workshop Assignment

- It's time to start the workshop assignment!
- Sit near your workshop partner.
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - 10-minute rule does *not* apply to talking to your partner, you should consult each other throughout.
- Follow the workshop instructions very closely
  - both the video and written instructions.
- Talk to your instructor if any of the instructions are unclear to you.

Check-Out

- Wrap up – Feedback/Discussion/Retrospective
  - What went well
  - What didn't go well
  - Suggestions for the instructor
- Review more of Week 3, start Week 4, or work on your Portfolio Project