LAB 11a

BEGINNING REACT

What You Will Learn

- How to use JSX to create components
- How to populate React components using props
- How to do conditional rendering in React
- How to add a behavior to a React components

Note

This chapter's content has been split into three labs: Lab11a, Lab11b, Lab11c.

Approximate Time

The exercises in this lab should take approximately 30 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

Date Last Revised: February 25, 2024
Revisions: Dramatically simplified this first lab; complexity will be moved to Lab11b

PREPARING DIRECTORIES

1 The starting `lab11a` folder has been provided for you (within the zip folder downloaded from Gumroad).

*Note: these labs use the convention of `blue background` text to indicate filenames or folder names and **`bold red`** for content to be typed in by the student.*

BEGINNING (PURE) REACT

This lab walks you through the creation of a few simple pure React applications. By "pure", I mean that at the start of this lab, you will be not using any build tools such as `vite` or `create-react-app`. Instead, you will begin with the simplest approach: using `<script>` tags to reference the React libraries and a `<script>` tag that will enable JSX conversion to occur at run-time. While this approach is certainly slower and not what you would do in a real-world application, it simplifies the process when first learning.

In the next React lab, you will take a better approach that puts each component in a separate file and which uses `vite` along with `npm` to compile and bundle the application.

Exercise 11a.1 — USING JSX

1 Examine `lab11a-ex00.html` in the browser. This file partially illustrates some of the layout we will be implementing in React (this file is just simple HTML). It uses the Bulma CSS framework, which is a lightweight and clean framework. Why are we using it? No real reason, other than to use something different!

2 Copy the first `<article>` element to the clipboard (this will save you typing in step 4).

3 Open `lab11a-ex01.html` in a code editor. We will be adding React functionality to this base page.

Notice that it already has the React JS files included via `<script>` tags at the top of the document. Notice also that it is using the Babel script library to convert our React JSX scripts at run-time. This is fine for learning and simple examples (such as this lab), but isn't a suitable approach for a production site. Later, we will make use of CLI tools that convert the JSX into JavaScript.

4 Add the following JavaScript code to the head. Notice the `type="text/babel"` in the script tag. This is necessary because you will be entering JSX and not JavaScript in this tag.

```
<script type="text/babel">

/* There are several ways of creating a React component.
   Here we are using a ES6 class */
function Company() {
  // this is not JS but JSX
  return (
    <figure>
          <img src="images/FB.svg" >
    </figure>
  )
}
/*
   We now need to add the just-defined component to the browser DOM
*/
const container = document.querySelector('#react-container');
ReactDOM.createRoot(container).render(<Company />);

</script>
```

Note that JSX is class sensitive and follows XML rules. Also, the class attribute has been changed to className.

5 Test in browser. NOTE: This code won't work.

6 Go to the JavaScript console and examine the error message.

React error messages are not always very clear. The problem is that React wants the `<img>` tag to have a closing end tag. Why? JSX is an XML-based syntax (just like the old XHTML was), and thus your JSX must follow XML syntax rules: case sensitive, all tags must be closed, and all attributes in quotes.

7 Fix the code by adding a close tag to the `<img>` element:

```
<img src="images/FB.svg" />
```

8 Save and test in the browser. It should display a single `<Company>` element (which is for now simply an image).

9 Add the following and test. Note: While this appears to work, check the console to examine the warning message.

```
<figure class="image is-128x128">
```

In JSX "class" is a reserved word, so you have to change it to something else.

10 Change the attribute as follows and test. It should now work.

```
<figure className="image is-128x128">
```

11 Copy the `<figure>` element and paste it after the existing `<figure>`. Thus the component will try to return two `<figure>` elements.

12 Save and test in the browser.

It won't work.

13 Go to the JavaScript console and examine the error message.

A rendered React component must have a single root element. Right now it has two <figure> elements so it won't work.

14 Modify the component by adding the following code and test.

```
function Company() {
    return (
      <article className="box media ">
        <div className="media-left">
          <figure className="image is-128x128">
            <img src="images/FB.svg" />
          </figure>
        </div>
        <div className="media-content">
          <h2>Facebook</h2>
          <p><strong>Symbol:</strong> FB</p>
          <p><strong>Sector:</strong> Internet Software and
                    Services</p>
          <p><strong>HQ:</strong> Menlo Park, California</p>
        </div>
        <div className="media-right">
          <button className="button is-link">Edit</button>
        </div>
      </article>
    );
}
```

The result should look similar to that shown in Figure 11a.1.

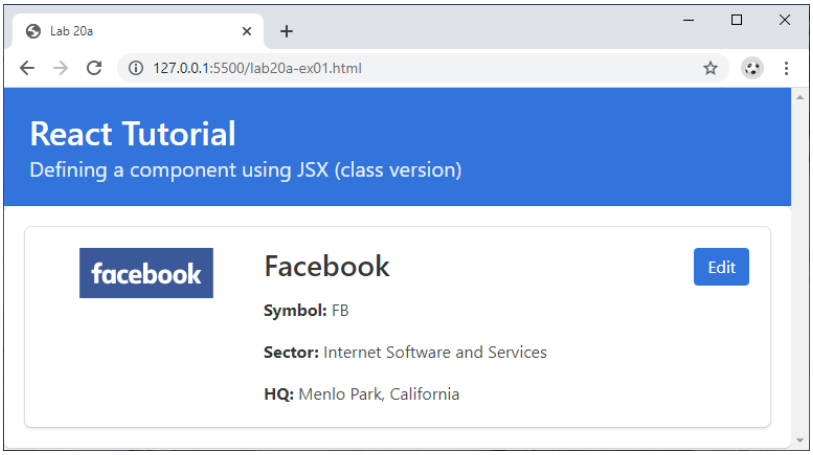

Figure 11a.1 – Finished Exercise 11a.1

Exercise 11a.2 — Combining Components

1 Open `lab11a-ex02.html`.

In this exercise, you will encounter another way to create React components.

2 Add the following code:

```
/* different ways to create functional components */
const OkButton = () => {
  return <button className="button is-primary">Ok</button>
}
const AnythingButton = () => {
  const label = "Anything";
  return <button className="button">{label}</button>
}
```

A react component is like any other JavaScript function so it can contain any valid JavaScript. Notice the syntax for including the content of a variable in the markup.

3 Add the following code.

```
/* Combining components */
const SimpleHeader = function() {
  return (
    <div className="buttons">
      <OkButton />
      <AnythingButton />
    </div>
  )
}
```

Components can be used within other components.

4 Modify the `createRoot.render` call as follows:

```
ReactDOM.createRoot(container).render(SimpleHeader());
```

5 Test in browser. The result should display the new component.

6 Modify the `createRoot.render` call as follows:

```
ReactDOM.createRoot(container).render(<SimpleHeader />);
```

Here we are passing the component returned from the SimpleHeader function. It will have the same output, it's just illustrates that the markup is an alternate syntax

7 Add the following new component.

```
const alternate = React.createElement( 'h2', {className: 'is-size-1'},
  'This is an alternate version of a component'
);
```

You will rarely do this: it is here just to illustrate how React converts a component into HTML behind the scenes.

8 Modify the `createRoot.render` call as follows:

```
ReactDOM.createRoot(container).render(alternate);
```

The Babel transcompiler (which converts from JSX to vanilla JavaScript) will convert the code from steps 2 and 3 into something similar to that shown in step 7.

Exercise 11a.3 — Component Parameters

1 Open `lab11a-ex03.html` and modify the following.

```
const AnythingButton = (props) => {
  return <button className="button">{props.label}</button>
}
```

React passes the component's attributes as an object to the components. While we could have used any parameter name, `props` is the conventional name because with class components, it had to be called props.

2 Modify the component as follows and test.

```
const SimpleHeader = function() {
  return (
    <header>
      <div className="buttons">
        <AnythingButton label="Something" />
      </div>
    </header>
  )
}
```

You can use any attribute name.

3 Modify the component as follows and test.

```
const AnythingButton = (props) => {
  const cls = "button is-" + props.size;
  return <button className={cls}>{props.label}</button>
}
```

4 Modify the component as follows and test.

```
<AnythingButton label="Something" size="large" />
```

You can pass any number of values to a component.

5 Add the following array and pass it to `SimpleHeader`:

```
const img = {
  classes: "image is-128x128",
  filename: "images/AAPL.svg",
  alt: "Apple Logo"
}
const container = document.querySelector('#react-container');

/* You will also need to modify this line */
const root = ReactDOM.createRoot(container);
```

```
root.render( <SimpleHeader logo={img} /> );
```

Notice that you can pass any object within props.

6 Modify the component as follows and test.

```
const SimpleHeader = function(props) {
  return (
    <header>
      <img src={props.logo.filename} alt={props.logo.alt}
           className={props.logo.classes} />
      <div className="buttons">
        <AnythingButton label="Something" size="large" />
      </div>
    </header>
  )
}
```

7 Let's separate out the `<img>` element out of `SimpleHeader`, by making the following changes and test:

```
const Logo = (props) => {
  return (
    <img src={props.filename} alt={props.alt}
           className={props.classes} />
  )
}
const SimpleHeader = function(props) {
  return (
    <header>
      <Logo filename={props.logo.filename} alt={props.logo.alt}
            classes={props.logo.classes} />
      <div className="buttons">
        <AnythingButton label="Something" size="large" />
      </div>
    </header>
  )
}
```

8 There is an alternate way to access a component's passed attribute values, which is shown below in the following changes:

```
const Logo = ({filename,alt,classes}) => {
  return (
    <img src={filename} alt={alt} className={classes} />
  )
}
```

Essentially, you are using ES6 object destructuring syntax to put the different props properties into separate variables.

Exercise 11a.4 — Reusing Components

1 Open `lab11a-ex04.html`.

Notice that it contains a Company component along with a data array.

2 Modify the `Company` component as follows.

```
const Company = (props) => {
    return (
      <article className="box media ">
        <div className="media-left">
              <figure className="image is-128x128">
                <img src={"images/" + props.symbol + ".svg"} />
              </figure>
        </div>
        <div className="media-content">
            <h2></h2>
            <p><strong>Symbol:</strong> {props.symbol}</p>
            <p><strong>Sector:</strong> {props.sector}</p>
            <p><strong>HQ:</strong> {props.hq}</p>
        </div>
      </article>
    );
};
```

3 Modify the `render` call as follows and test.

```
ReactDOM.createRoot(container).render(
     <Company symbol="FB" sector="Internet Software"
              hq="Menlo Park, California" />
);
```

4 Modify as follows and test.

```
ReactDOM.createRoot(container).render(
  <Company symbol="FB" sector="Internet Software"
           hq="Menlo Park, California" >
     Facebook
  </Company>
);
```

Here we are changing to the component's markup so that it contains a nested element: in React terminology we are adding a child element.

5 Modify the component as follows and test.

```
        <div className="media-content">
            <h2>{props.children}</h2>
```

The `children` property provides access to any content added as nested content.

6 Modify as follows and test.

```
ReactDOM.createRoot(container).render(
  <section  className="content box">
    <Company symbol="FB" sector="Internet Software"
            hq="Menlo Park, California" >
      Facebook
    </Company>
    <Company symbol="AAPL" sector="Information Technology"
            hq="Cupertino, California">Apple</Company>
  </section>
);
```

7 Modify as follows.

```
<Company data={{name:"Facebook", symbol:"FB",
   sector:"Internet Software",
   hq:"Menlo Park, California"}} />

<Company data={{name: "Apple", symbol:"AAPL",
    sector:"Information Technology",
    hq:"Cupertino, California"}} />
```

8 Modify as follows and test.

```
<div className="media-left">
      <figure className="image is-128x128">
        <img src={"images/" + props.data.symbol + ".svg"} />
      </figure>
</div>
<div className="media-content">
    <h2>{props.data.name}</h2>
    <p><strong>Symbol:</strong> {props.data.symbol}</p>
    <p><strong>Sector:</strong> {props.data.sector}</p>
    <p><strong>HQ:</strong> {props.data.hq}</p>
</div>
```

This illustrates that you can pass JavaScript to a component within the curly braces.

9 Notice the array comps that has been provided in the starting file. It contains an array of company objects. Instead of passing an object literal in the attributes, let's pass an object instead, by making the following modification (and then test).

```
<Company data={comps[3]} />
```

10 Modify as follows and test.

```
ReactDOM.createRoot(container).render(
  <section  className="content box">
    { comps.map(c => <Company data={c} />) }
  </section>
);
```

This illustrates a very common coding idiom in React: namely, using the map() function on a data array in order to output multiple components. Later you will learn how to retrieve data from a Web API in React instead of using a hard-coded array.

11 Examine the JavaScript console. You will see that there is a React warning that each child in a list needs a unique key. To fix, modify as follows and test.

```
{ comps.map(c => <Company data={c} key={c.symbol} />) }
```

Each key value in a React list must be a unique value. Because the stock symbol property is a unique value, we can use it for the key. But what if we didn't have that in our data?

12 Modify as follows and test.

```
{ comps.map( (c,indx) => <Company data={c} key={indx} />)  }
```

Here we are making use of the optional second parameter to the map function: it will contain the index of the array element, which will also be unique.

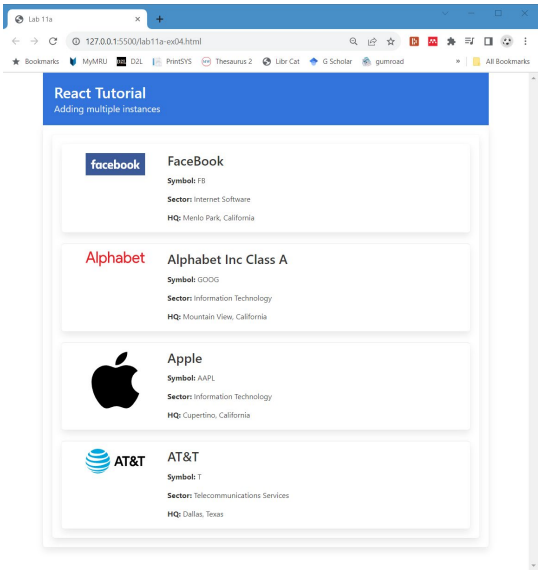

Figure 11a.2 – Finished Exercise 11a.4

In the next exercises, you will implement conditional rendering within a component. This allows you to render different markup based on some condition.

Exercise 11a.5 — Conditional Rendering

1 Open `lab11a-ex05.html`. View in browser.

Notice that it contains a variety of components already with the <App> component being the root/top element. Notice also that it uses an array of movies to provide the information for the three displayed cards.

2 Let's imagine that we want to output a different card footer depending on whether the user is logged in. Let's emulate the user's logged in status via an attribute. To do so, make the following modification:

```
ReactDOM.createRoot(container).render(<App loggedIn={false} />);
```

3 Modify the following in the `<App>` component.

```
{ movies.map(m=> <MovieCard title={m.title} id={m.id}
                      count={m.count} loggedIn={props.loggedIn}
                     tagline={m.tagline} key={m.id} />) }
```

4 Create the following component:

```
const CardFooterCount = (props) => {
    return (<p className="card-footer-item is-size-6">
                Favorited {props.count} Times!
            </p>)
};
```

5 Modify the `<MovieCard>` component as follows and test.

```
const MovieCard = (props) => {
    const conditionalFooter = () => {
        if (props.loggedIn)
            return <CardFooterButton />
        else
            return <CardFooterCount count={props.count} />
    }

    return (
        <li className="column is-one-third">
            <div className="card">
                <CardMovieImage id={props.id} alt={props.title} />
                ...
                <footer className="card-footer">
                    { conditionalFooter() }
                </footer>
            </div>
        </li>
    )
};
```

6 You can make the code even simpler by using the ternary operator, as follows:

```
<footer className="card-footer">
    { props.loggedIn ? <CardFooterButton /> :
                       <CardFooterCount count={props.count} /> }
</footer>
```

7 Comment out the `conditionalFooter` function and then test.

8 Test by changing the `loggedIn` value from Step 2 to `true`.

In the next exercises, you will add behaviors to the components.

Exercise 11a.6 — Adding A Behavior

1 Open `lab11a-ex06.html` and add the following code:

```
const CardFooterButton = () => {
    const handleClick= () => {
        alert("add to favorites");
    }

    return (
        <button className="button card-footer-item"
                onClick={handleClick} >
            <span className="icon is-small">
                <i className="fas fa-heart"></i>
            </span>
        </button>
    );
};
```

2 Test in browser by clicking on any of the card footer buttons.

This isn't all the impressive perhaps. It is possible to define event handlers in a variety of ways in React (though this is the most common).

3 Modify as follows and test.

```
<button className="button card-footer-item"
        onClick={ () => { alert("add to favorites"); } }  >
```

You will occasionally see this approach, in which the handler is defined within the element. This can be convenient when the handler is only a single line of code.

Making buttons do more will require using State in React and learning how to pass messages between components. This you will learn in the next lab!

TEST YOUR KNOWLEDGE #1

In this exercise, you will create a page containing five React components as shown in Figure 11a.5. The HTML that your page must eventually render has been provided in the file `lab11a-test01-markup-only.html`. The CSS has been provided. The starting code provides an array of painting objects.

With React, you may prefer to start working first on the most "outer" component (in this case `App`), or you may decide to start working first from the most "inner" component (which in this case is `PaintingListItem`), or start with the simplest (in this case that would be the Header component). The steps below suggest one possible set of steps, though you may decide to follow your own steps.

1. Create the `Header` component. When clicked the button will simply display an alert message. Remember that you can look at `lab11a-test01-markup-only.html` to see what markup your component should render.
2. Create the `PaintingList` component. Initially just have this component render the root `<section>` element with some temporary text.
3. Create the `EditPaintingForm` component. Initially just have this component render the root `<section>` element with the `<form>` and `<div>` elements. Assume a single painting object is passed via props whose data will be displayed by the component.
4. Modify the `App` component so that it uses these three components (there is boilerplate text in the start file which indicates where they are to be located).
5. Create the `PaintingListItem` component. Assume a single painting object is passed via props whose data will be displayed by the component. When the row is clicked, display the painting title in an alert box.
6. Your `PaintingList` component is going to display multiple `PaintingListItem` components (one for each painting). Assume the entire array of paintings is passed. Use the `map()` function to render each painting object as a `PaintingListItem`. Verify this works.

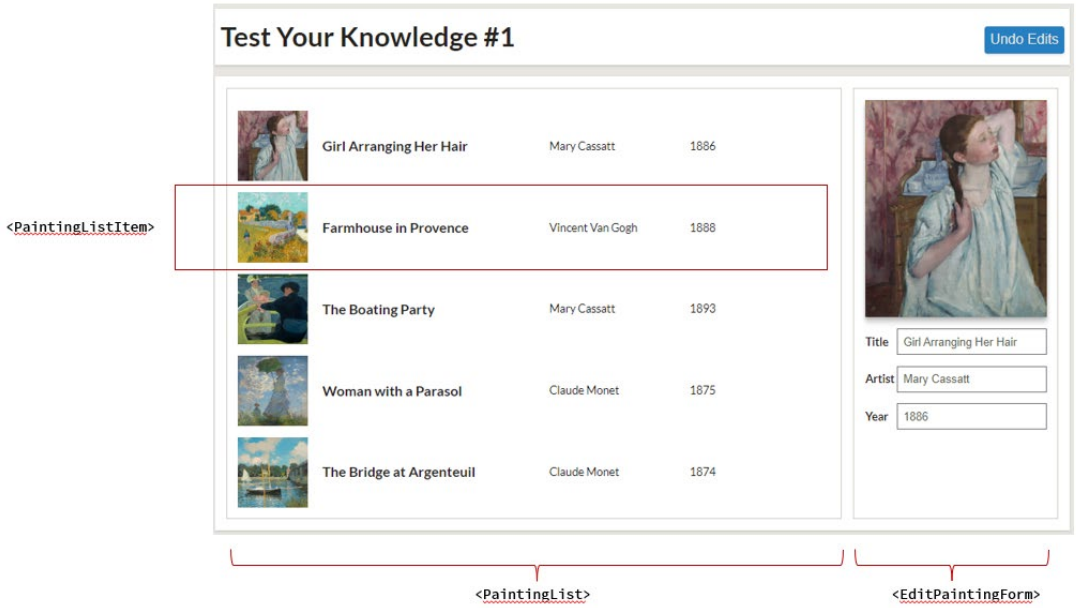

Figure 11a.5 – Components for Test Your Knowledge #1