LAB 11c

Extending React

What You Will Learn

- How to make use of styled components
- How to integrate third-party React components
- How to use ContextProvider as an alternative state mechanism

Note

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

Approximate Time

The exercises in this lab should take approximately 90 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 7, 2022

Preparing Directories

1 This lab has additional content contained within the provided `lab11c` folder. You will need to copy this additional content in this folder as described in the exercises below.

*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.*

In the first part of this lab, you will again use `create-react-app`.

Exercise 11c.1 — Setting up using create-react-app

1 Using your terminal, ensure you are in your `lab11c` folder.

2 Assuming you have completed lab11b, you should have create-react-app installed.

Run the following command if you have npx:

```
npx create-react-app my-app
```

Run the following command if you do not have npx:

```
create-react-app my-app
```

It will take a few minutes to finish this process.

3 In your terminal, switch to `my-app` folder.

4 Type and run the following command:

```
npm install --save react-router-dom
```

5 To run and test our project, you need to switch to the `my-app` folder and run the project. This will start its own development web server (using node). To do this, run the following command from the terminal:

```
npm start
```

This should display a message saying compilation was successful. It might also automatically open a browser tab with the sample starting react app visible.

6 Copy the contents of the `cra-src` folder in the supplied `lab11c` folder into the `src` folder of `my-app`.

7 Verify content works in browser. Try clicking the buttons.

Styled Components

There are several ways to provide styles to React. So far, you have simply used one or more CSS files. Eventually, these CSS files are combined together into one file during the build step. This means you have to be careful not to overwrite definitions, be aware of the cascade, etc. An alternate approach is to make use of a React styling library that allows you to define styles via JavaScript within your components. Perhaps the most popular of these is styled components (`https://styled-components.com/`).

Exercise 11c.2 — Using Styled Components

1 Type and run the following command:

```
npm install --save styled-components
```

2 Examine `PaintingItem.js` and `PaintingItem.css`. We are going to replace the CSS with styled components.

3 In `PaintingItem.js` comment out the importing of the CSS file:

```
//import './PaintingItem.css';
```

4 Add the following reference to style-components to the top of this component.

```
import styled from 'styled-components';
```

4 In `PaintingItem`, add the following:

```
const Card = styled.div`
   box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
   padding: 16px;
   text-align: center;
   background-color: #f1f1f1;
`;
```

The styled object is defined within styled-components. It uses tagged template literal syntax. This is equivalent to a call to a function, that is, `styled.div(` ... `)`.

5 Modify the `return` statement as follows and test.

```
return (
   <Card>
      <figure>
         <img src={url} alt={props.painting.Title} />
         < figcaption >{props.painting.Title}</ figcaption >
      </figure>
   </Card>
);
```

The `styled` functions return React functional components.

6 In `PaintingItem`, add the following additional style objects:

```
const Image = styled.img`
   width: 200px;
`;
const Caption = styled.figcaption`
   font-size: 0.75rem;
   width: 200px;
`;
const Figure = styled.figure`
   margin: 0;
   padding: 0;
`;
```

7 Modify the return statement as follows and test.

```
return (
   <Card>
      <Figure>
         <Image src={url} alt={props.painting.Title} />
         <Caption>{props.painting.Title}</Caption>
      </Figure>
   </Card>
);
```

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

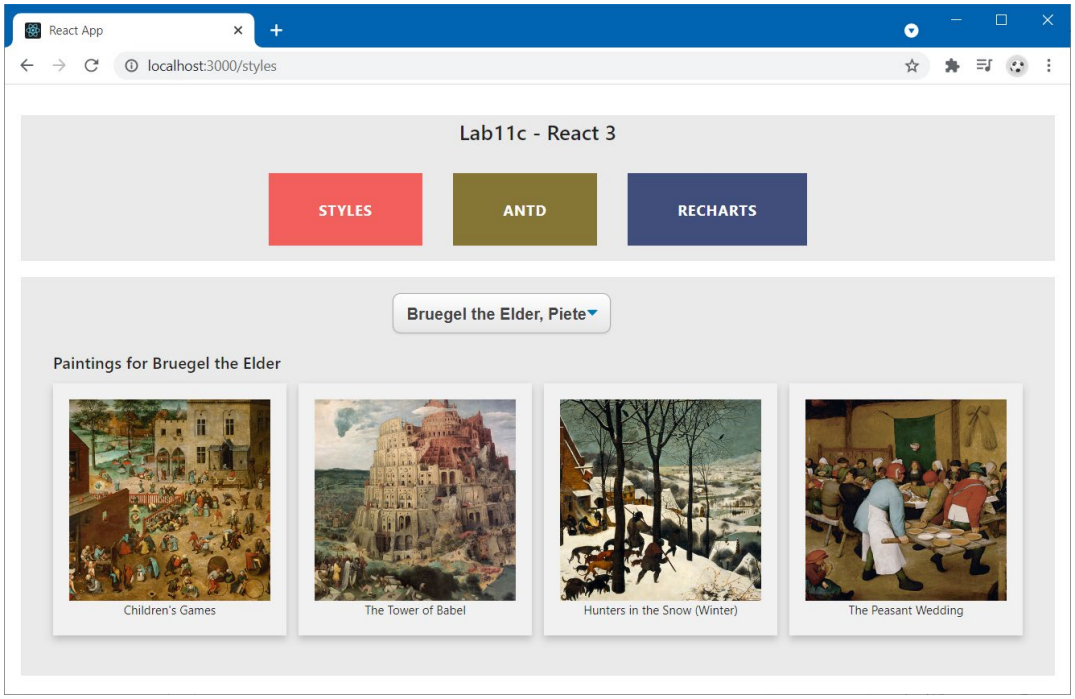

Figure 11c.1 – Using styled components

Exercise 11c.3 — Extending Styled Components

1 Edit `Navigation.js` by copying the `.btn` styles from `Navigation.css` into the `Navigation` function.

```
const NavButton = styled.button`
    border: none;
    background: none;
    cursor: pointer;
    padding: 25px 50px;
    display: inline-block;
    margin: 15px 15px;
    text-transform: uppercase;
    letter-spacing: 1px;
    font-weight: 700;
    outline: none;
    color: #fff;
`;
```

2 Comment out the reference to `Navigation.css` and add reference to style-components to the top of this component.

```
//import './Navigation.css';
import styled from 'styled-components';
```

3 Create an extension of this `NavButton` as follows in `Navigation.js` (copy the `.btn-1` and `.btn-1:hover` properties from `Navigation.css`).

```
const StylesButton = styled(NavButton)`
    background: #F25F5C;
    &:hover {
        background: #9B3D3B;
    }
`;
```

4 Modify the button reference by using the new styles:

```
<Link to='/styles'>
    <StylesButton>Styles</StylesButton>
</Link>
```

5 Test in browser.

6 Create unique buttons for the other three buttons using the same techniques in steps 3 and 4. Test.

One of the latest additions to CSS are CSS Modules, which is a CSS file in which all class names and animation names are scoped locally by default. They allow you to use the same CSS class name in different files without worrying about naming conflicts (for instance, a class named `box` in a later file replaces any earlier definitions of it).

Exercise 11c.4 — Styling using CSS Modules

1. Create a new file named `NavButton.module.css`. In this new file, copy and paste all the content from `Navigation.css`.

2. Change the first CSS rule as follows:

```
button {
   border: none;
   background: none;
   cursor: pointer;
   padding: 25px 50px;
   display: inline-block;
   margin: 15px 15px;
   text-transform: uppercase;
   letter-spacing: 1px;
   font-weight: 700;
   outline: none;
   color: #fff;
}
```

   Because we are making this change in a CSS module, we don't have to worry about changing the style of other <button> elements: it will only apply to the <button> elements in the files using this CSS module.

3. In `Navigation.js`, comment out everything from the importing `styled-components` to the closing `}` for the `Navigation` function.

4 Add the following code.

```
import styles from './NavButton.module.css';

const Navigation = (props) => {
    return (
        <nav>
            <Link to='/styles'>
                <button className={styles.btn1}>Styles</button>
            </Link>
            <Link to='/antd'>
                <button className={styles.btn2}>Antd</button>
            </Link>
            <Link to='/recharts'>
                <button className={styles.btn3}>
                   Recharts</button>
            </Link>
        </nav>
    );
};
```

Notice the reference to the CSS module file in the className references.

5 Test.

It should work just the same as the previous exercises.

6 In the browser, use the Inspect tools to examine the generated HTML. In my browser, one of the buttons is rendered as follows.

```
<button class="NavButton_btn1__2uUe6">Styles</button>
```

Notice how the module classes have been given unique names.

Both the styled-components and CSS Modules approaches allows for styling details to be encapsulated with the component itself. So which of these approaches is better? The styled-components approach is a JavaScript only approach. That is, no CSS files are necessary, though knowledge of CSS is still required. However, because the styling is now within the JavaScript, it realistically cannot be modified by a non-programming designer. The CSS Module approach has the benefit of keeping the CSS within CSS files, where they can be maintained and modified by a non-programming designer.

Animation

There are several well-supported React animation libraries. In the next exercise, you will make use of `react-animations`, which implements those available in the popular `animation.css` library.

Exercise 11c.5 — Using Styled Components

1 Type and run the following command:

```
npm install --save react-animations
```

2 In `Navigation.js`, comment out everything from the previous exercise, and uncomment everything from the importing `styled-components` to the closing `}` for the `Navigation` function.

3 Modify the `import` statements in `Navigation.js` as follows:

```
import styled, { keyframes } from 'styled-components';
import { slideInDown, headShake } from 'react-animations';
```

4 Add the following to the top of the `Navigation` function.

```
const slideAnimation = keyframes`${slideInDown}`;
const btnAnimation = keyframes`${headShake}`;

const AnimatedNavigation = styled.nav`
    animation: 1s ${slideAnimation};
`;
```

5 Modify the `StylesButton` function as follows:

```
const StylesButton = styled(NavButton)`
    background: #F25F5C;
    &:hover {
        background: #9B3D3B;
        animation: 1s ${btnAnimation};
    }
    &:active {
        background: #F69997;
    }
`;
```

6 Modify the `return` statement as follows:

```
return (
    <AnimatedNavigation>
        <Link to='/styles'>
            <StylesButton>Styles</StylesButton>
        </Link>
        …
    </AnimatedNavigation>
);
```

7 Test.

There should be an entrance slide-in animation of all the buttons. When you hover over the first button, another animation (headShake) should play.

Using Third-Party User Interface Components

There is a very rich ecosystem of React user-interface components that can aid in the creation of sophisticated React user experiences. Some of the most popular are Material UI (created by Google) and Ant Design (created by Alibaba), which are entire design systems. Later in this section, you will use a simpler component library, the elegant Chakra UI as a template for create-react-app. But before that, let's integrate some user-interface components into our existing project.

Exercise 11c.6 — Using Ant Design Components

1 Type and run the following commands:

```
npm install --save antd
npm install --save @ant-design/icons
```

2 Add the following to `HomeAntd.css`.

```
@import '~antd/dist/antd.css';
```

3 Edit `HomeAntd.js` as follows.

```
import React from 'react';
import './HomeAntd.css';

import { PageHeader, Divider, Space, Button } from 'antd';
import { SearchOutlined  } from '@ant-design/icons';

const HomeAntd = (props) => {

  return (
    <section>
      <PageHeader title="Antd Demo"
          subTitle="we will demo only a few components"
        />
      <Space>
        <Button type="primary" icon={<SearchOutlined />}>
            Primary </Button>
        <Button value="small">Default </Button>
        <Button type="link" danger>Danger Link</Button>
        <Button disabled>Disabled</Button>
      </Space>

      <Divider />
    </section>
  );
}
```

4 Test by clicking on the ANTD button.

5 Add the following after the `Divider` element.

```
<Divider />
<p>In stock <Switch defaultChecked /></p>
<p>Difficulty Rating <Rate value={5} /> </p>
<Space>
  <NotificationOutlined />
  <Badge dot>
    <NotificationOutlined />
  </Badge>
  <Badge count={5}  >
    <NotificationOutlined  style={{ fontSize: '18px' }}/>
  </Badge>
</Space>

<Divider />
```

6 Add the following imports then test.

```
import { Switch, Rate, Badge } from 'antd';
import { NotificationOutlined  } from '@ant-design/icons';
```

7 Add the following after the second `Divider` element.

```
<Divider />
<Card title="Complex component inside a Card" >
  <Result status="success"
          title="Successfully used Ant Design components"
          subTitle="This is an example of a Result component"
          extra={[
            <Button type="primary">Go to Website</Button>,
            <Button>Reformat Hard Drive</Button>,
          ]}
  />
</Card>

<Divider />
```

8 Add the following import then test.

```
import { Card, Result } from 'antd';
```

9 Add the following after the third `Divider` element.

```
<Divider />
<Card title="Calendar inside a Card" style={{ width: 500 }}>
  <Calendar fullscreen={false}/>
</Card>
```

10 Edit the previous import as follows then test.

```
import { Card, Result, Calendar } from 'antd';
```

The finished exercise should look similar to that shown in Figure 11c.2

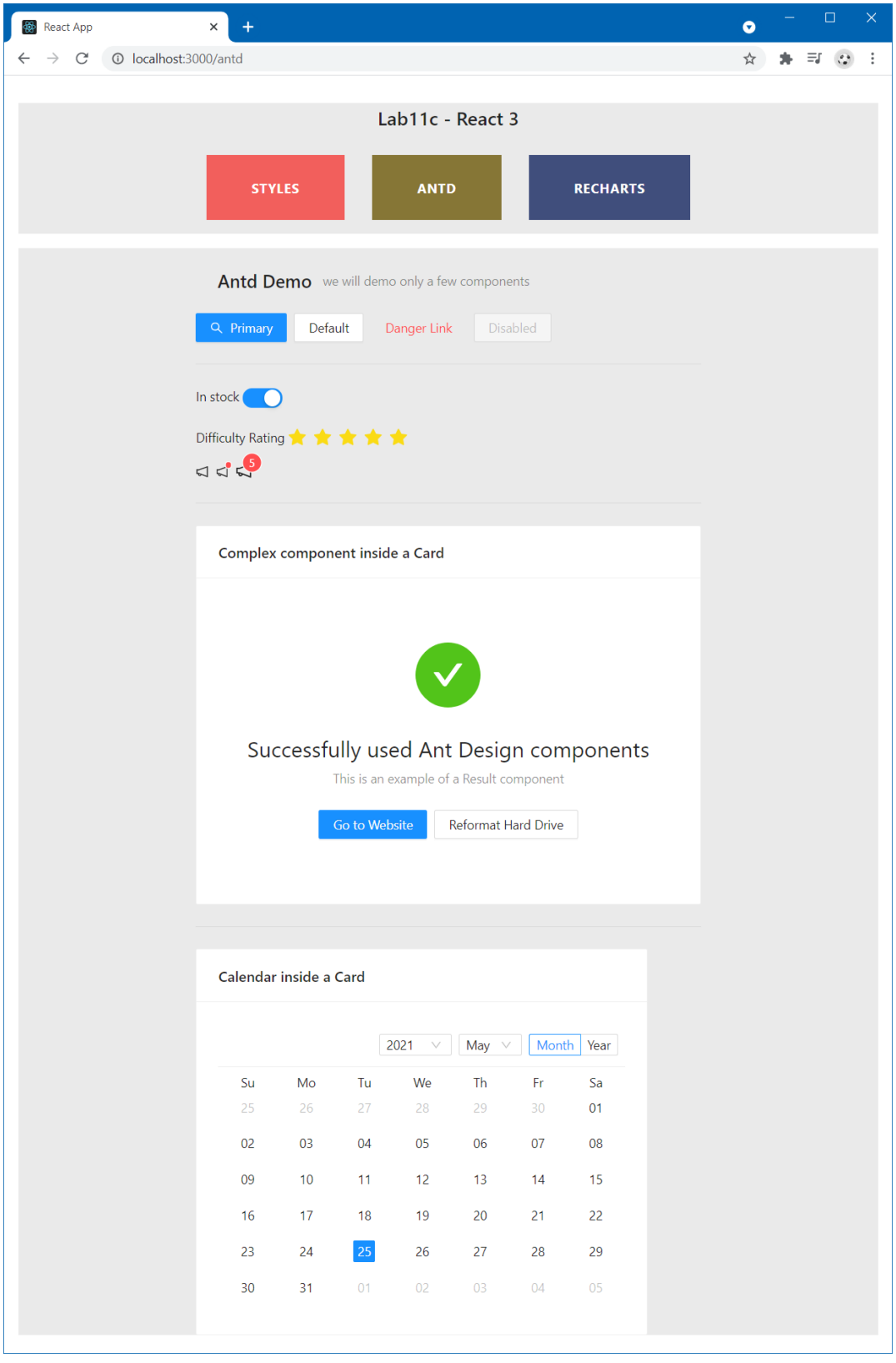

Figure 11c.2 – Using Ant Design components

Exercise 11c.7 — Using Recharts Components

1 Type and run the following command:

```
npm install --save recharts
```

2 Add the following to the top of `HomeRechart.js`.

```
import { BarChart, Bar, XAxis, YAxis, CartesianGrid,
         Tooltip, Legend} from "recharts";
```

3 Add the following to the `return`:

```
<section>
  <h2>Home Rechart </h2>
  <BarChart width={730} height={250} data={data}>
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis dataKey="name" />
    <YAxis />
    <Tooltip />
    <Legend />
    <Bar dataKey="2018" fill="#8884d8" />
    <Bar dataKey="2019" fill="#82ca9d" />
    <Bar dataKey="2020" fill="#FF8042" />
  </BarChart>
</section>
```

4 Test by clicking on the Rechart button.

The finished exercise should look similar to that shown in Figure 11c.3

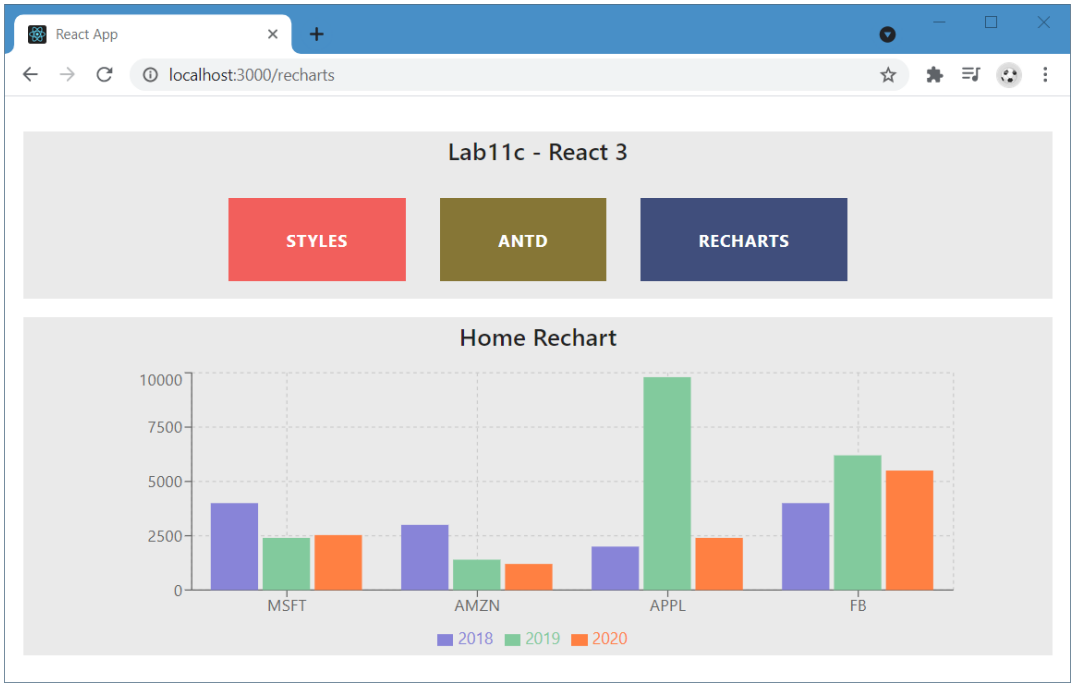

Figure 11c.3 – Using a Rechart component

Create-React-App Templates

Whenever you use the create-react-app application, it bases the new site it creates on a template. By default, it uses one named `cra-template`; you can however use a different template. For instance, there is a TypeScript version named `cra-template-typescript`. Many large design system component libraries are available as a create-react-app template, which can simplify the usage of a component library. In the next example, you will create an application that uses the Chakra component library via a template.

Exercise 11c.8 — Using a Create-React-App template

1. Using your terminal, ensure you are in your `lab11c` folder.

2. Run the following command:

```
npx create-react-app context-app --template @chakra-ui
```

Notice that this adds a template specification.

3. Run the following commands:

```
cd context-app
npm start
```

This should display the starting file in the browser.

4. In you code editor, examine `App.js` within the `src` folder.

5. Edit the import statements as follows.

```
import React, { useEffect, useState } from 'react';
import Header from "./components/Header.js";
import ArtBrowser from "./components/ArtBrowser.js";
import { ChakraProvider, theme } from '@chakra-ui/react';
```

6. Modify the rest of the function as follows.

```
function App() {
  // will store list of paintings in state
  const [paintings, setPaintings] = useState([]);
  // retrieve list of painints from localStorage or API
  useEffect( () => {
    // first see if in localStorage
    const paintingsInBrowser =
            localStorage.getItem('paintingsFromAPI');
    // if in localstorage, then use it
    if (paintingsInBrowser) {
      setPaintings(JSON.parse(paintingsInBrowser));
    }
    else {
```

```
      const url =
"https://www.randyconnolly.com/funwebdev/3rd/api/art/paintings-
nested.php?public=100";
      fetch(url)
        .then( resp => resp.json() )
        .then( data => {
         // save paintings in state
          setPaintings(data);
         // put in local storage
          localStorage.setItem('paintingsFromAPI',
                                    JSON.stringify(data) );
        })
        .catch( err => console.error(err));
   }
  }, []);

  return (
    <ChakraProvider theme={theme}>
      <Header />
      <ArtBrowser paintings={paintings} />
    </ChakraProvider>
  );
}
```

6 Create a folder in the `src` folder named `components`.

7 Copy the contents of the `chakra-components` folder in the supplied `lab11c` folder into the `components` folder created in the previous step.

8 Examine `PaintingCard.js`. You will see that it already makes use of a variety of Chakra components.

This lab is not going to step you through all the code necessary to construct the user interface using Chakra. If interested, you can examine both the components that have been provided to you as well as the online documentation for Chakra.

9 Test using the `npm start` command.

The result should look similar to that shown in Figure 11c.3. Test the View button for one of the paintings to see the Chakra Drawer in action. The favorites will be added in the next set of exercises.

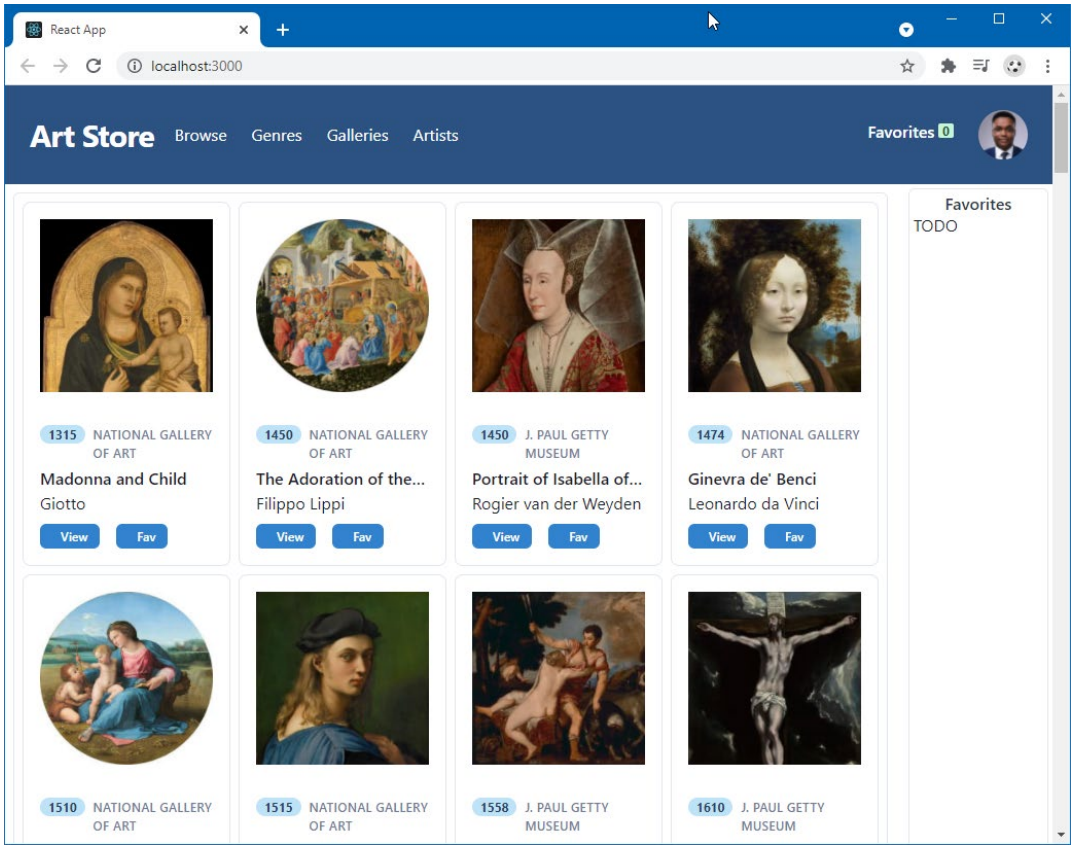

Figure 11c.4 – Chakra-based layout

Alternate Approaches to State

State in React typically requires the upper-most parent component to house the state variables and all behaviors that can modify this state. This prop-drilling tends to dramatically reduce the encapsulation of React child components, since they become dependent on their ancestors to pass in the data and behaviors they need as props. As a result, for more complex React applications, developers often make use of an alternative approach to maintaining the application's state.

With the release of React Hooks in 2019, the `useContext()` hook provides a way to centralize data state into a single location known as a `context` which is available to both functional and class components. This requires first creating a context provider, which provides access to the state stored in the `context` object. All children of this provider will then have access to this centralized state.

Exercise 11c.9 — Creating a Context Provider

1 Create a new folder named `contexts` within the `src` folder.

2 Within this new folder, create a file named `FavoriteContext.js`.

3 Add the following code.

```
import React, {useState, createContext} from 'react';
// create the context object which will hold the state
export const FavoriteContext = createContext();
// create the object which will provide access to this context
const FavoriteContextProvider = (props) => {
  const [favorites, setFavorites] = useState([]);

  return (
   <FavoriteContext.Provider value={{favorites, setFavorites}} >
      {props.children}
   </FavoriteContext.Provider>
  )
}
export default FavoriteContextProvider;
```

4 Modify the `App` component as follows.

```
import FavoriteContextProvider from
         './contexts/FavoriteContext.js';
...
  return (
    <FavoriteContextProvider>
      <ChakraProvider theme={theme}>
        <Header />
        <ArtBrowser paintings={paintings} />
      </ChakraProvider>
    </FavoriteContextProvider>
  );
```

By wrapping our entire application in our context provider, it will be available throughout the application.

5 Modify the `Header` component as follows and test.

```
import React, {useContext} from "react";
import { FavoriteContext } from
           '../contexts/FavoriteContext.js';
...
const Header = props => {
  const { favorites } = useContext(FavoriteContext);
...
    Favorites
    <Badge colorScheme="green" ml="1">{favorites.length}</Badge>
```

Any component can now access the state variables "stored" within our FavoriteContext.

6 Modify the `PaintingCard` component as follows.

```
import React, {useContext} from "react";
import { FavoriteContext } from
          '../contexts/FavoriteContext.js';
...
const PaintingCard = (props)  => {
   const { favorites, setFavorites } =
           useContext(FavoriteContext);
...
   const addFav = () => {
      // make sure not already in favorites
      let f = favorites.find( f => f.id === p.paintingID);
      // if not in favorites then add it
      if (! f) {
        const newFavs = [...favorites];
        newFavs.push({id: p.paintingID,
                      filename: p.imageFileName,
                      title: p.title});
        setFavorites(newFavs);
      }
   }
...
```

This code implements the function that adds a painting to the favorites list. Notice once again that it retrieves and manipulates the state through the context provider.

7 Modify the `FavoriteItem` component as follows.

```
import React, {useContext} from "react";
import { FavoriteContext } from
          '../contexts/FavoriteContext.js';
...
const FavoriteItem = (props)  => {
   const { favorites, setFavorites } =
      useContext(FavoriteContext);
...
   const removeFav = () => {
      const newFavs = favorites.filter( f => f.id !== item.id )
      setFavorites(newFavs);
   }
...
```

8 Modify the `Favorites` component as follows.

```
import React, {useContext} from "react";
import FavoriteItem from "./FavoriteItem.js";
import { FavoriteContext } from
          '../contexts/FavoriteContext.js';
...
const Favorites = ()  => {
  const { favorites } = useContext(FavoriteContext);

  return (
    <Box border="1px" borderRadius="md" borderColor="gray.200"
         m={1} p={1} as="section">
      <Flex align="center" justify="center"
            direction="column">
         <Heading as="h3" size="sm"  color="gray.700"
               fontWeight="500">Favorites</Heading>
      </Flex>
     { favorites.map(
          f => <FavoriteItem item={f} key={f.id} /> ) }
      </Box>
   );
}
```

9 Test.

Notice that adding an item to favorites will update both the list of favorites and the favorite count in the header. The finished results should be similar to that shown in Figure 11.17 page 591 of Chapter 11.