LAB 11c

Extending React

What You Will Learn

- How to use React Router
- 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 60 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

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

Date Last Revised: March 15, 2024
Revisions: Switched to Vite, Adding routing, modified Context example

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 `Vite`.

Exercise 11c.1 — Setting up using Vite

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

2 Run the following command:

```
npm create vite@latest lab11c-styling-app -- --template react
```

The extra double-dash is needed. This will create a scaffolded react application in a folder named lab11b-react-app. It will also install the create-vite application the first time it is run.

3 In your terminal, switch to `lab11c-styling-app` folder.

4 Run the following command:

```
npm install
```

This will take a few minutes as it will download and install all the dependencies listed in the `package.json` file.

5 To run and test our project, you need to switch to the `lab11c-styling-app` folder and run the project via the following command from the terminal:

```
npm run dev
```

This should display a message saying compilation was successful.

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

7 Verify content works in browser.

Using React Router

React Router is a package that allows a developer to configure routes. What is a route? In normal HTML, you use hyperlinks (anchor tags) to jump from page to page in your application. But in a single-page application, there is only one page. Using React Router, you can turn links to different pages into links to display React components (on the same page).

The React Router package includes quite a few components. In this lab, we will use just three: `<BrowserRouter>`, `<Route>`, and `<Link>`.

Exercise 11c.2 — Adding the React Router

1 In the Terminal, press Ctrl-C and stop the batch job. This should stop the server and return you to the terminal prompt. We are doing this because we are going to install some additional npm modules.

We can later, at any time, restart the server by running `npm run dev`.

2 Type and run the following command:

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

3 In the `src` folder, modify the `main.jsx` file as follows.

```
import { BrowserRouter} from 'react-router-dom';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
)
```

4 Add the following to the top of `App.jsx`:

```
import { Routes, Route } from 'react-router-dom';

const App = (props) => {
```

5 Add the following to the returned markup:

```
<main>
  <Routes>
    <Route path='/styles' element={<HomeStyles />} />
    <Route path='/antd' element={<HomeAntd />} />
    <Route path='/recharts' element={<HomeRechart />} />
  </Routes>
</main>
```

These routes describe which components to display based on the path request.

6 Examine `HomeStyles.jsx`, `HomeAntd.jsx`, and `HomeRechart.jsx`.

The last two components are simple placeholder components right now.

7 Add the following to the top of `Navigation.jsx`:

```
import { Link } from 'react-router-dom';
```

8 Add the following to the returned markup:

```
<nav>
    <Link to='/styles'>
        <button className="btn btn1">Styles</button>
    </Link>
    <Link to='/antd'>
        <button className="btn btn2">Antd</button>
    </Link>
    <Link to='/recharts'>
        <button className="btn btn3">Recharts</button>
    </Link>
</nav>
```

The value of the `to` attribute maps to the value of the `path` attribute in the `<Route>` element. Essentially, the `<Link>` element converts a link or buttons within it by adding JavaScript that will display a component.

9 Test. It should default to displaying the `HomeStyles` component, and only display the `HomeAntd` or `HomeRechart` components when you click the relevant buttons. Notice the URL in the browser as you click: it changes the path/route.

Styled Components

There are several ways to provide styles to React. In lab11b (and so far in this lab) you used custom CSS or classes defined in a CSS library like Bulma or Tailwind. In the previous example in this lab, each component had its own custom CSS file; these 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 to custom styling 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.3 — Using Styled Components

1 Type and run the following command:

```
npm install styled-components
```

2 Examine `PaintingItem.jsx` 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 **outside (before)** the function:

```
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. Visually, there hasn't been a change but now the styling is within the JavaScript and no longer in the separate CSS file.

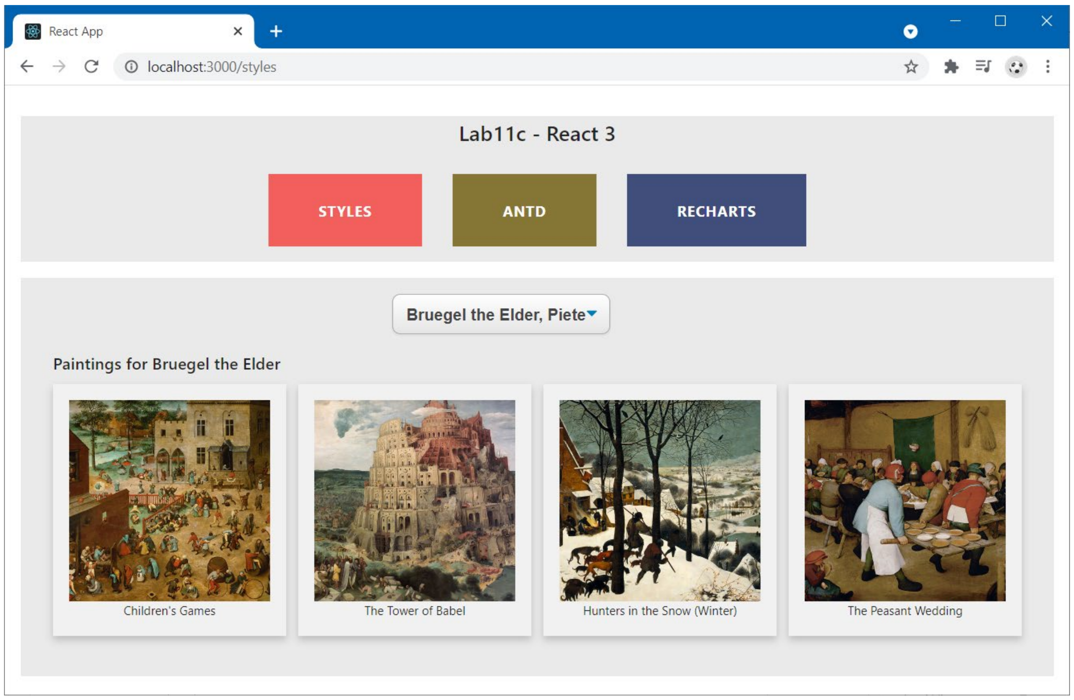


Figure 11c.1 – Using styled components

Exercise 11c.4 — Extending Styled Components

1 Edit Navigation.jsx 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 in Navigation.jsx, but before the function (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.5 — 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.6 — Using React Animations

1 Type and run the following command:

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

2 In `Navigation.jsx`, 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 before 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 (

        <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.7 — USING ANT DESIGN COMPONENTS

1 Type and run the following commands:

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

2 Visit the following URL and explore the documentation:

```
https://ant.design/components/overview
```

3 Edit `HomeAntd.js` as follows.

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

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

const HomeAntd = (props) => {

  return (
    <section>
      <h2>Antd: we will demo only a few components</h2>
      <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>
<div>Difficulty Rating: <Rate value={5} /> </div>
<Divider />
<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 `Divider` element added in step 5.

```
<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>Open Modal</Button>,
          }
  />
</Card>

<Divider />
```

8 Add the following import then test.

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

9 Modify the imports as follows:

```
import { Card, Result, Modal } from 'antd';
import { useState } from 'react';
```

10 Add the following to the `HomeAntd` function:

```
const HomeAntd = (props) => {
  const [isModalOpen, setIsModalOpen] = useState(false);

  const showModal = () => {
    setIsModalOpen(true);
  };
  const handleOk = () => {
    setIsModalOpen(false);
  };
  const handleCancel = () => {
    setIsModalOpen(false);
  };
```

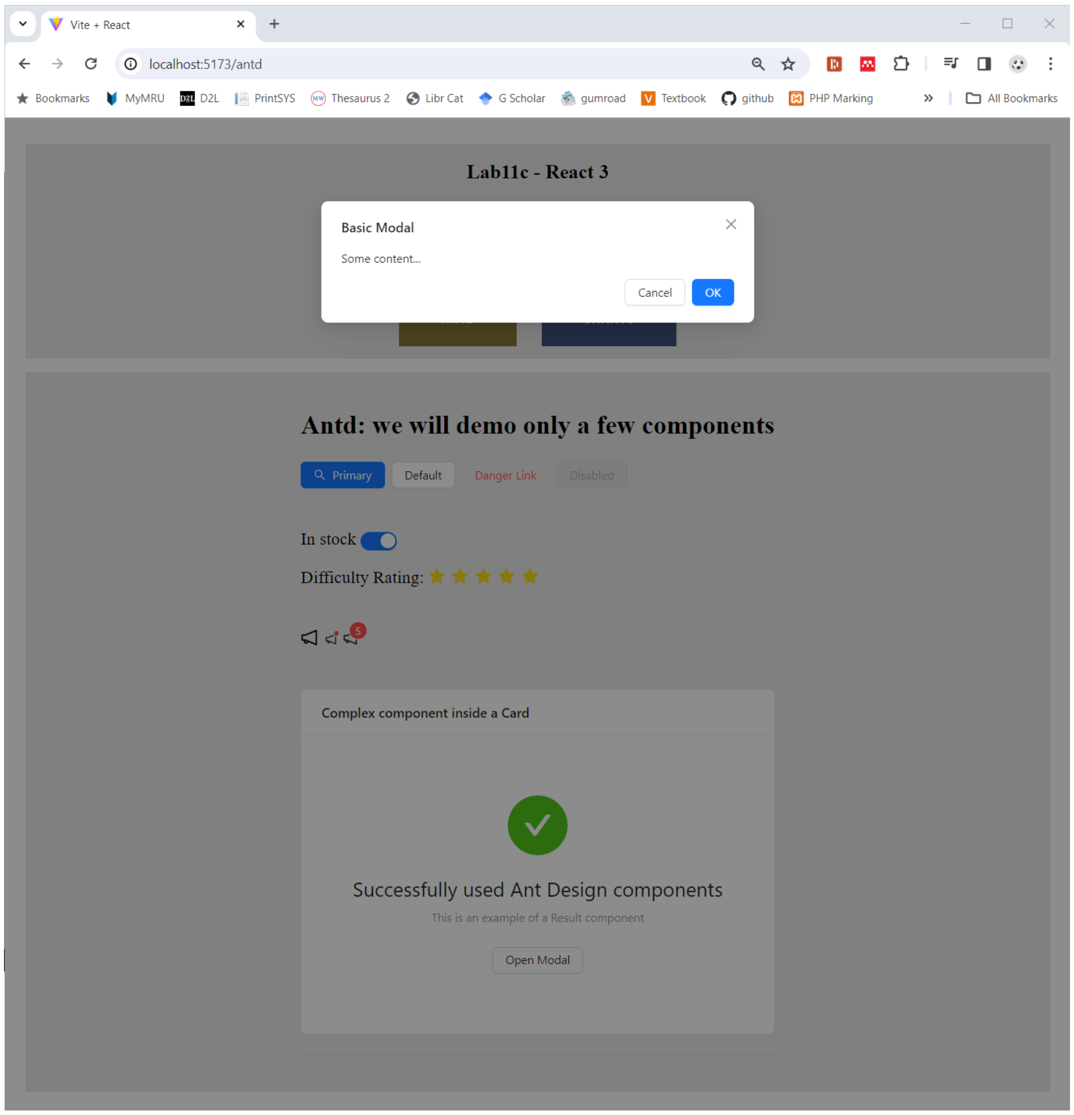
11 Modify the `Button` inside the `Result` component as follows:

```
<Button onClick={showModal}>Open Modal</Button>
```

12 Add the following component after the `Card` component and test.

```
<Modal title="Basic Modal" open={isModalOpen} onOk={handleOk}
       onCancel={handleCancel}>
  <p>Some content...</p>
</Modal>
```

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



Lab11c - React 3

Basic Modal

Some content...

Cancel OK

Antd: we will demo only a few components

Primary Default Danger Link Disabled

In stock

Difficulty Rating:

Complex component inside a Card

Successfully used Ant Design components

This is an example of a Result component

Open Modal

Figure 11c.2 – Using Ant Design components

Exercise 11c.8 — Using Recharts Components

1 Type and run the following command:

```
npm install 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

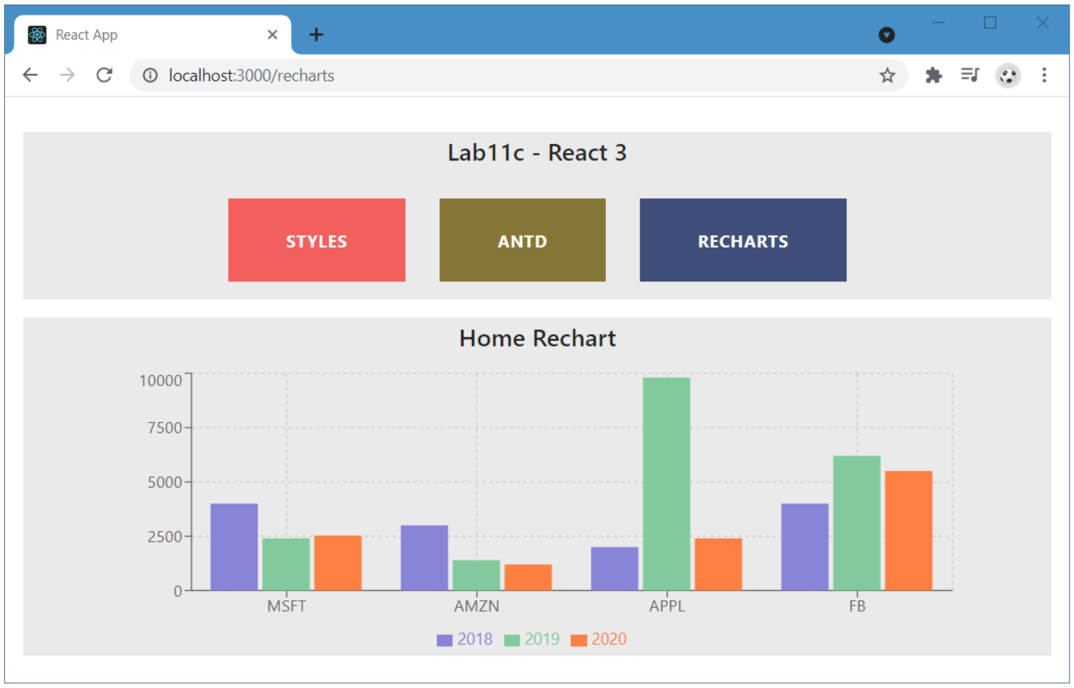


Figure 11c.3 – Using a Rechart component

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 the Context Application

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

2 Run the following command:

```
npm create vite@latest lab11c-context-app -- --template react
```

The extra double-dash is needed. This will create a scaffolded react application in a folder named lab11b-react-app. It will also install the create-vite application the first time it is run.

3 In your terminal, switch to `lab11c-context-app` folder.

4 Run the following commands:

```
npm install
npm install @chakra-ui/react react-icons
npm install @emotion/react @emotion/styled framer-motion
```

This will take a few minutes as it will download and install all the dependencies listed in the `package.json` file.

6 Copy the components within the `src-context` folder in the supplied `lab11c` folder into the `src` folder of `lab11c-context-app`.

7 Copy the `paintings.json` file within the `src-context` folder in the supplied `lab11c` folder into the `public` folder of `lab11c-context-app`.

8 Delete the contents of `index.css` and `App.css`.

9 Modify `main.jsx` as follows:

```
import { ChakraProvider } from '@chakra-ui/react'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <ChakraProvider>
      <App />
    </ChakraProvider>
  </React.StrictMode>,
)
```

10 Add the following to `App.jsx`:

```
import Header from "./Header.jsx";
import ArtBrowser from "./ArtBrowser.jsx";
import { useState, useEffect } from 'react';


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 = "/paintings.json";
      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 (
    <div>
      <Header />
      <ArtBrowser paintings={paintings} />
    </div>
  )
```

11 Test in browser.

The result should look similar to that shown in Figure 11c.4

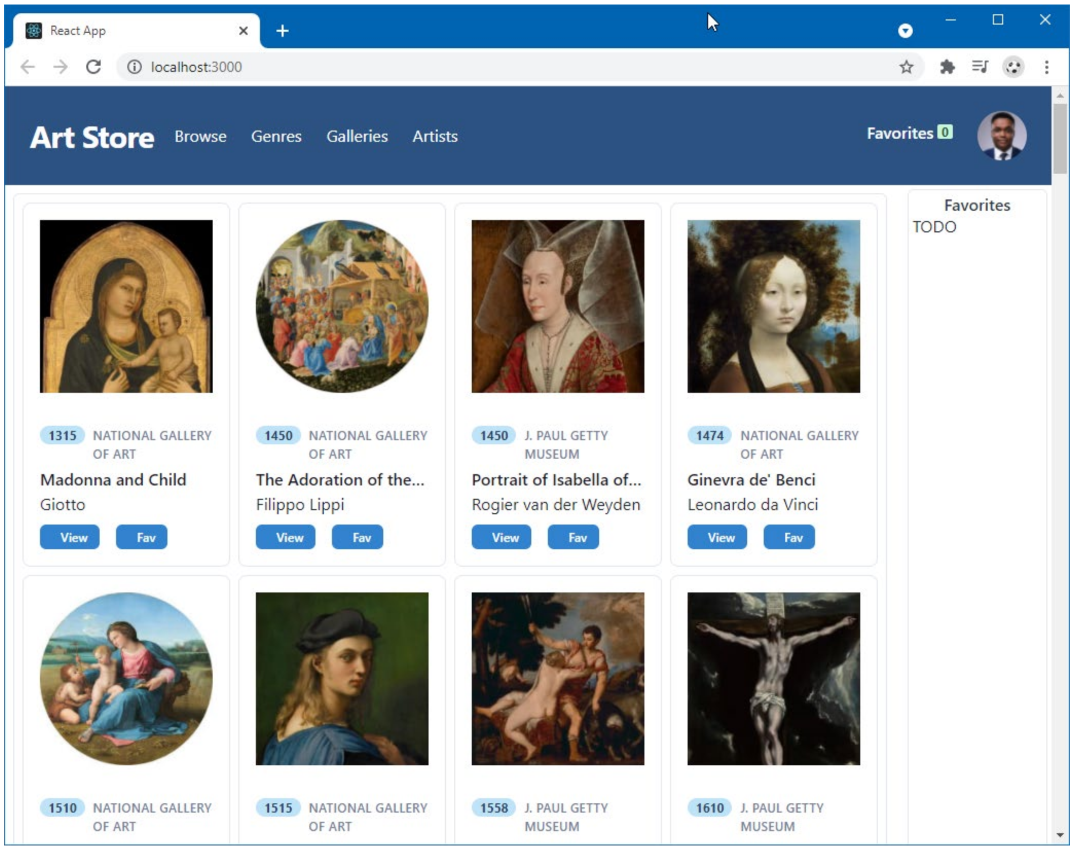

React App
localhost:3000
Art Store
Browse
Genres
Galleries
Artists
Favorites 0
Favorites
TODO
1315 NATIONAL GALLERY OF ART
Madonna and Child
Giotto
View
Fav
1450 NATIONAL GALLERY OF ART
The Adoration of the...
Filippo Lippi
1450 J. PAUL GETTY MUSEUM
Portrait of Isabella of...
Rogier van der Weyden
1474 NATIONAL GALLERY OF ART
Ginevra de' Benci
Leonardo da Vinci
1510 NATIONAL GALLERY OF ART
1515 NATIONAL GALLERY OF ART
1558 J. PAUL GETTY MUSEUM
1610 J. PAUL GETTY MUSEUM

Figure 11c.4 – Chakra-based layout

Exercise 11c.10 — Creating a Context Provider

1 Add the following code to `App.jsx`.

```
import { useState, useEffect, createContext } from 'react';
// create the context object which will hold the favorite state
export const FavoriteContext = createContext();

// create (and export) the context object which will hold the state
export const FavoriteContext = createContext();
function App() {
   ...
  // current favorites will be in state
  const [favorites, setFavorites] = useState([]);
  return (
    <div>
      <FavoriteContext.Provider
              value={{favorites, setFavorites}} >
          <Header />
          <ArtBrowser paintings={paintings} />
      </FavoriteContext.Provider>
    </div>
  );
```

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

```
import { useState, useContext } from "react";
import { FavoriteContext } from "./App";
...
const Header = props => {
   const { favorites } = useContext(FavoriteContext);
   ...
   <Text fontSize="md" color="#A0AEC0" >Favorites</Text>
   <Badge colorScheme="green" ml="1">{favorites.length}</Badge>
```

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

6 Modify the `PaintingCard` component as follows.

```
import {useContext} from "react";
import { FavoriteContext } from './App';
...
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,
                      paintingObject: p});
        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 {useContext} from "react";
import { FavoriteContext } from './App';
...
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 {useContext} from "react";
import FavoriteItem from "./FavoriteItem.jsx";
import { FavoriteContext } from './App';
...
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 11c.5.

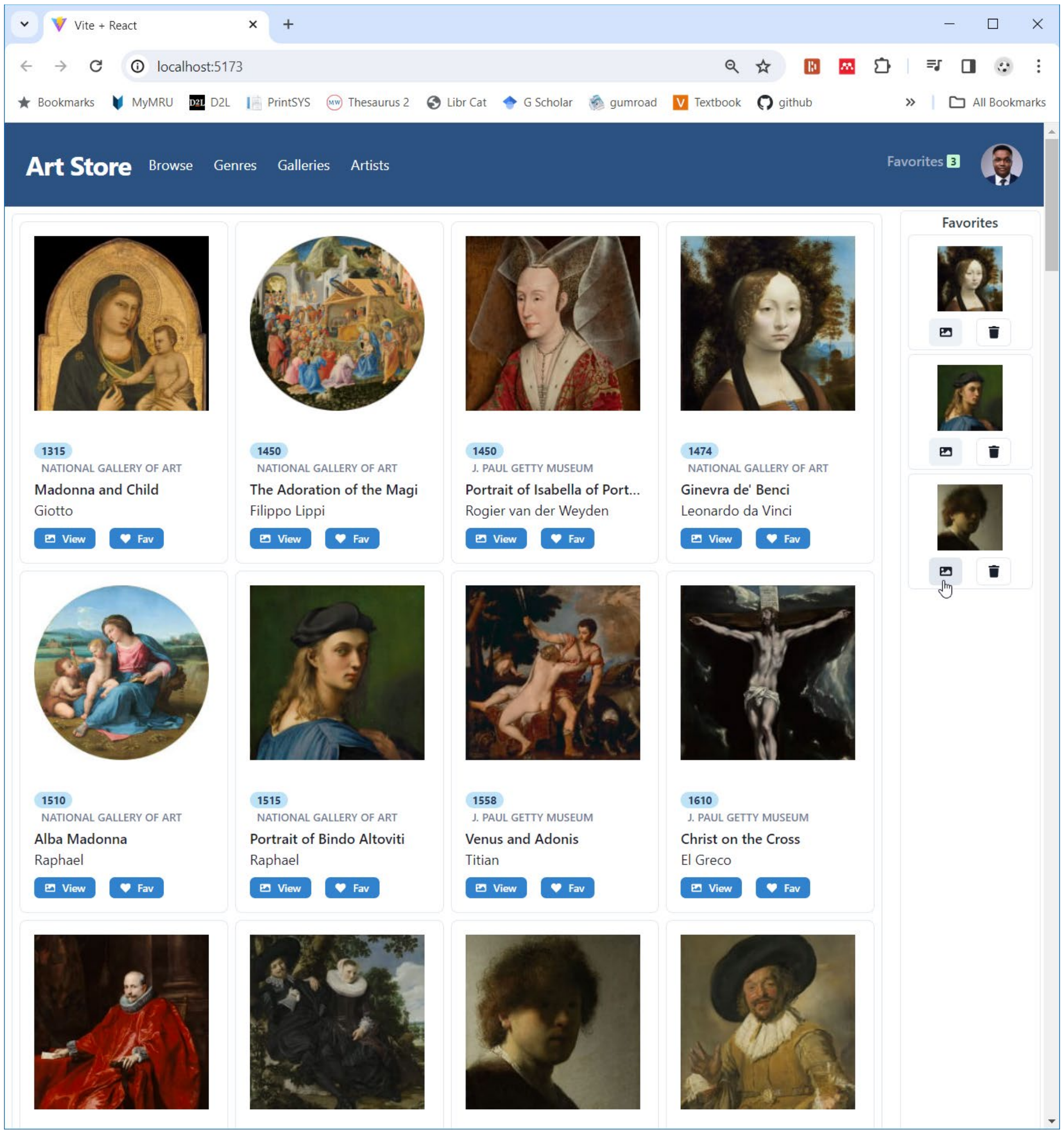


Figure 11c.5 – Finished example