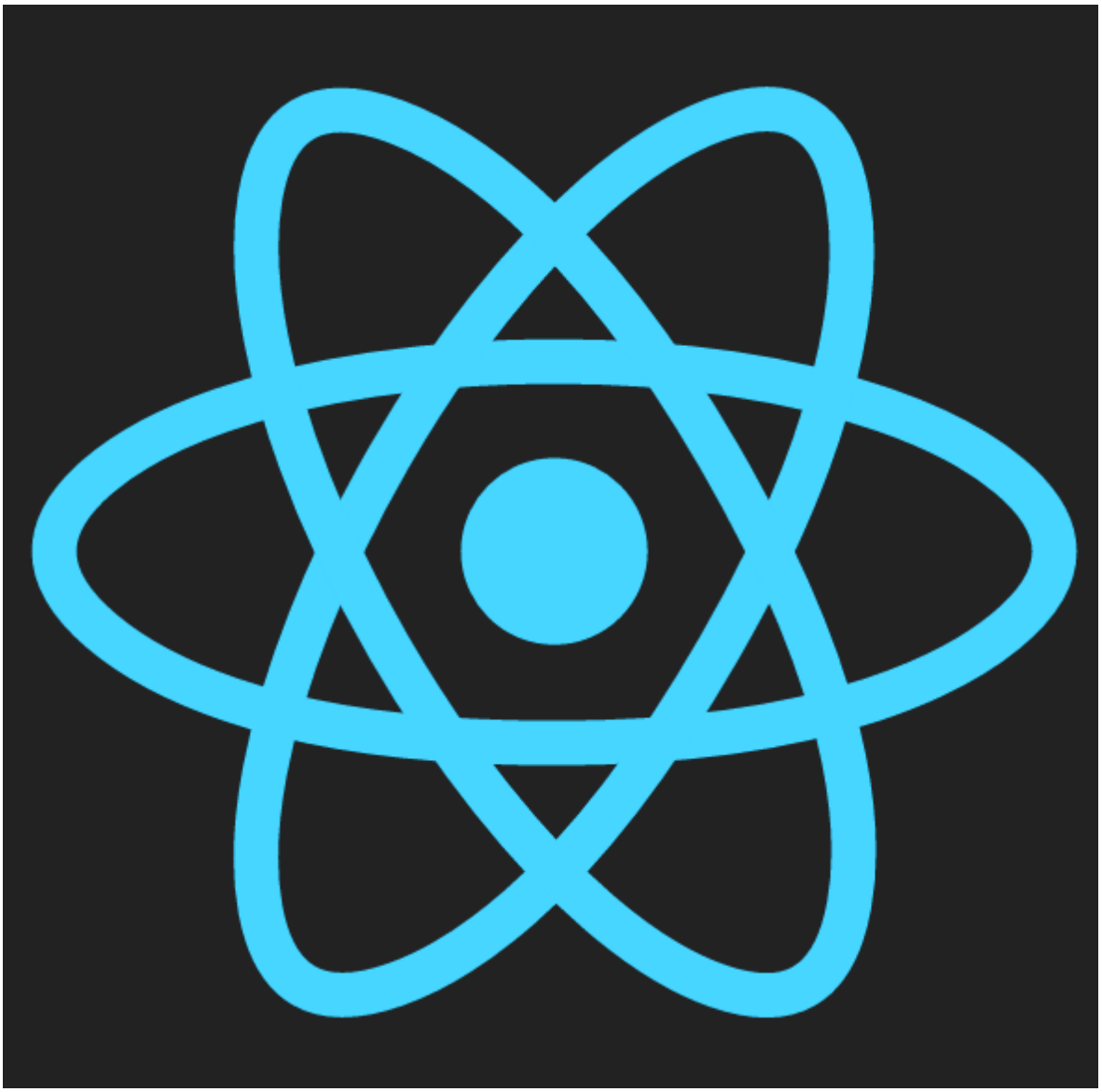
Advanced ReactJS



Completed source code for all labs (for checking your work) can be found at:

`https://github.com/chrisminnick/advanced-react`

Version 1.0.0, September 2023
by Chris Minnick

www.watzthis.com

Table of Contents

Disclaimers and Copyright Statement

Disclaimer

WatzThis? takes care to ensure the accuracy and quality of this courseware and related courseware files. We cannot guarantee the accuracy of these materials. The courseware and related files are provided without any warranty whatsoever, including but not limited to implied warranties of merchantability or fitness for a particular purpose. The use of screenshots, product names and icons in the courseware is for editorial purposes only. No such use should be construed to imply sponsorship or endorsement of the courseware, nor any affiliation of such entity with WatzThis?.

Third-Party Information

This courseware contains links to third-party websites that are not under our control, and we are not responsible for the content of any linked sites. If you access a third-party website mentioned in this courseware, you do so at your own risk. We provide these links only as a convenience, and the link's inclusion does not imply that we endorse or accept responsibility for the content on those third-party websites. Information in this courseware may change without notice and does not represent a commitment by the authors and publishers.

Copyright

Help us improve our courseware.

Please send your comments and suggestions via email to info@watzthis.com

Credits

About the Author

Chris Minnick is a prolific published author, trainer, web developer, and founder of WatzThis, Inc. Minnick has overseen the development of hundreds of web and mobile projects for customers from small businesses to some of the world's largest companies, including Microsoft, United Business Media, Penton Publishing, and Stanford University.

Since 2001, Minnick has trained thousands of Web and mobile developers. In addition to his in-person courses, Chris has written and produced online courses for Ed2Go.com, Skillshare, O'Reilly Media, and Pluralsight.

Minnick has authored and co-authored books and articles on a wide range of Internet-related topics, including JavaScript, ReactJS, HTML, CSS, mobile apps, e-commerce, Web design, SEO, and security. His published books include JavaScript All-In-One For Dummies, Coding All-In-One For Dummies, ReactJS Foundations, JavaScript for Kids, Writing Computer Code, Coding with JavaScript For Dummies, Beginning HTML5 and CSS3 For Dummies, Webkit For Dummies, CIW eCommerce Certification Bible, and XHTML.

Course Requirements

To complete the labs in this course, you will need:

- A computer with MacOS, Windows, or Linux.
- Access to the Internet.
- A modern web browser.
- Ability to install software globally.
- Experience with JavaScript and React.

Lab 01: React Fundamentals

In this lab, we'll review fundamental concepts in React, including state management, component lifecycle, hooks, and class vs. function components. You'll use these concepts to implement new features for a real-time chat application.

- ☐ 1. If you haven't already, clone the repo at `https://github.com/chrisminnick/advanced-react`
- ☐ 2. Open the repository in VSCode. In it, you'll find a directory named **real-time-chat**. This is the starter project for this lab.
- ☐ 3. Open the **real-time-chat** directory. Inside it is a **client** directory and a **server** directory.
- ☐ 4. Open the **README** file and follow the installation instructions.
- ☐ 5. Once both the client and server are running, check the terminal window where you're running the server. You should see a message that the server is running, and that the server connected to the database.
- ☐ 6. Look at the client app in your browser. It should show the login page.
- ☐ 7. Click the signup link and fill it out (use a fake email address), then log in.
- ☐ 8. Open a different browser (such as Edge) and go to http://localhost:3000 and sign up / login with that browser.
- ☐ 9. You should be able to chat with yourself. Fun!
- ☐ 10. Look through the server and client to see how they work. How would you improve them?
- ☐ 11. Look at the Future Features section of the README and think about how you might implement each one.
- ☐ 12. Pick one of the features and implement it. Make sure to use a separate git branch and commit your work frequently so you can always get back to something that works.
- ☐ 13. Repeat step 11 until the time is up for the lab.

Lab 02: Routing and authentication

In this lab, you'll implement routes for user signup and authentication using an existing API.

☐ 1. Bootstrap a new React project in **/social-media** and name it client.

```
npx create-react-app client
```

☐ 2. Follow the instructions in social-media/server to install and run the server.

☐ 3. Install `react-router-dom` into **social-media/client** and copy code and components from the real-time-chat project to implement **/signup**, **/login**, and **/logout** routes. Hint: the API for the social-media project has a **/user** endpoint that's identical to the one in the real-time-chat project.

☐ 4. Make signup, login, and logout work and redirect the user to a restricted route when they successfully login.

☐ 5. Challenge: Pull data from the **/posts** endpoint and display posts on the restricted route users are redirected to.

☐ 6. Bonus Challenge: Create a **/post/:id** route that will display a single post

☐ 7. Bonus Challenge: Make a form that users can use to create new posts and hook it up. Hint: look in the **server/api** directory for sample HTTP Requests. Install the Rest Client VSCode extension (by Huachao Mao) for VSCode to run these.

Lab 03: Converting to Redux with Redux Toolkit

In this lab, you'll start with an existing non-redux app and convert it to use Redux.

☐ 1. Install the required dependencies: `redux`, `react-redux`, and `@reduxjs/toolkit`.

☐ 2. Create a **store** directory in your project's root directory.

☐ 3. Create a **store.js** file in the store directory.

☐ 4. In **store.js**, import `configureStore` from `@reduxjs/toolkit`.

☐ 5. Create a `rootReducer` that combines your reducers.

```
import { configureStore } from '@reduxjs/toolkit';
import { combineReducers } from 'redux';
import { authSlice } from '../features/auth/authSlice';
import { postsSlice } from '../features/posts/postsSlice';

const rootReducer = combineReducers({
  auth: authSlice.reducer,
  posts: postsSlice.reducer,
});
```

☐ 6. Use `configureStore` to create a store with your `rootReducer`.

```
export const store = configureStore({
  reducer: rootReducer,
});
```

☐ 7. Import the store from your `store` module into `App`.

☐ 8. Import `Provider` from `react-redux`.

☐ 9. Wrap your `App` component with the `Provider` component from `react-redux`, passing in your store as a prop.

```
function App() {
  return (
    <Provider store={store}>
      <div className="App container">
        <Routes />
      </div>
    </Provider>
  );
}
```

☐ 10. Import `{BrowserRouter as Router}` from `react-router-dom` in `App.js` and wrap the div with `<Router>`.

```
function App() {
  return (
    <Provider store={store}>
      <Router>
        <div className="App container">
          <Routes />
        </div>
      </Router>
    </Provider>
  )
```

```
};
```

☐ 11. Update the `ProtectedRoutes` component to use the `useSelector` hook to get the user property from the Redux state:

```
import { Navigate, Outlet } from 'react-router-dom';
import { useSelector } from 'react-redux';

export const ProtectedRoute = () => {
  const { user } = useSelector((state) => state.auth);

  // Check if the user is authenticated
  if (!user) {
    // If not authenticated, redirect to the login page
    return <Navigate to="/login" />;
  }

  // If authenticated, render the child routes
  return <Outlet />;
};
```

☐ 12. Update the `routes/index.js` file to look like this:

```
import { Route, Routes as RouterRoutes } from 'react-
router-dom';
import { ProtectedRoute } from './ProtectedRoutes';

import LoginPage from '../pages/LoginPage';
import SignupPage from '../pages/SignupPage';
import LogoutPage from '../pages/LogoutPage';
import HomePage from '../pages/HomePage';
import PostsPage from '../pages/PostsPage';

const Routes = () => {
  return (
    <RouterRoutes>
      <Route path="/" element={<HomePage />} />
      <Route path="/login" element={<LoginPage />} />
      <Route path="/logout" element={<LogoutPage />} />
      <Route path="/signup" element={<SignupPage />} />
      <Route path="/posts" element={<ProtectedRoute />}>
        <Route path="/posts" element={<PostsPage />} />
      </Route>
    </RouterRoutes>
  );
};

export default Routes;
```

☐ 13. Create a **pages** directory in **src**. and create the components that you imported into **routes/index.js**. Copy over and rename any components that you have in **/components** already.

☐ 14. Create a **features** directory in **src** and subdirectories named **posts** and **auth**.

- ☐ 15. Create a file named **authSlice.js** in **features/auth/**.
- ☐ 16. Copy the code for **authSlice.js** from **/solutions/social-media/client/features/auth/authSlice.js**
- ☐ 17. Update the `LoginPage`, `SignupPage`, and `LogoutPage` by referring to **/solutions/social-media/client** in the GitHub repo.
- ☐ 18. Copy **api/api.js** from **solution/social-media/client**.
- ☐ 19. Start the server (by running **npm start** in the **/server** directory).
- ☐ 20. Start the client app.
- ☐ 21. Test your app in the browser and fix any errors that prevent it from running.
- ☐ 22. Sign up for a new account, Log in, then log out to make sure it all works.
- ☐ 23. Install the Redux Dev Tool (if you don't have it already) and repeat the above steps with the Dev Tool open.
- ☐ 24. Add a form to the **Posts** page for adding new posts and connect it to Redux and the API.
- ☐ 25. Make the **Posts** page display a list of the current posts.
- ☐ 26. Make clicking a single post navigate to a new route, **posts/[id]** that displays just that post.
- ☐ 27. Implement or update additional features as time permits.

Lab 04: Microfrontends with Single SPA

In this lab, you'll use the Single SPA framework to create a microfrontend.

☐ 1. In an empty directory that's not inside of any other Node project (no **package.json** at a higher level), open a new terminal window and invoke `create-single-spa`.

```
npx create-single-spa --moduleType root-config
```

☐ 2. Answer all the questions that `create-single-spa` asks, choosing the defaults whenever possible.

☐ 3. Run **npm start** in your new project and open a browser to **http://localhost:9000**.

You now have a root config and an example application. You'll see some instructions for what to do next in the sample application that's running at port 9000. Read through those instructions. We're going to use Single SPA to run two React applications and share dependencies between them.

☐ 4. Open a new terminal window and generate a single-spa application by running:

```
npx create-single-spa --moduleType app-parcel
```

☐ 5. When you're asked for a directory for the application, name it something creative like 'app1'

☐ 6. Once it finishes, cd to your new directory and run **npm start**.

☐ 7. Open a browser and go to the localhost post that it gives you when it starts up. (probably localhost:8001).

☐ 8. Read through this page, but don't follow these instructions just yet.

☐ 9. Open **src/index.ejs** in your root config (not in your app1 subdirectory) and find the script element with `type="systemjs-importmap"`. Since all of our microfrontends will use React, we need to add React and ReactDOM to this import map.

☐ 10. Go to **https://cdnjs.com/libraries/react** and get the latest link for the React library (it should have **umd** in the URL) and add it to the **importmap**, then do the same for the ReactDOM library (you can just copy the same url and change "react" to "react-dom" in the URL.

☐ 11. When you're finished, your importmap should look like this:

```
    <script type="systemjs-importmap">
      {
        "imports": {
          "single-spa": "https://cdn.jsdelivr.net/npm/single-
spa@5.9.0/lib/system/single-spa.min.js",
          "react":
"https://cdnjs.cloudflare.com/ajax/libs/react/18.2.0/umd/react.production.min.js",
          "react-dom": "https://cdnjs.cloudflare.com/ajax/libs/react-
dom/18.2.0/umd/react-dom.production.min.js"
        }
      }
    </script>
```

☐ 12. Open **src/microfrontend-layout.js** and find the `<route>` element. Add your new application as a 2nd application. You can get the value for the name property from the **package.json** file in your **app1** directory. For example:

```
<application name="app1/@minnick/my-first-
app"></application>
```

☐ 13. In your root config's **src** directory, open **index.ejs** and add your application to the `importmap`. Note that there are two importmaps in the file, and you should add your application to both. Here's an example of what you should add:

```
"@minnick/my-first-project": "//localhost:8081/minnick-my-first-project.js",
```

☐ 14. Stop both your root config and you application and restart them. In your browser, you should now see a message saying that your application is mounted. It will look like this:

```
@minnick/my-first-project is mounted!
```

☐ 15. Remove the sample application from your **src/index.ejs** so your new application is the only one being rendered.

☐ 16. Create a second application and render that one in addition to the first.