Zach Gharst Directed Readings – Full Stack Open CS 497

Part 6 & Part 7

Introduction

Building the table seen in the previous reports for the deliverables takes a considerable amount of time (around an additional 5 minutes per exercise, with 45+ exercises per module). At first, I thought that perhaps there will be enough comments/screenshots/etc needed for each exercise, but the exercises range anywhere from a 5 minute task to an hour task. The deliverables section now has a link to the GitHub commits, and lists anything notable only.

The blog list application has been deployed to:

https://calm-spire-00374.herokuapp.com/

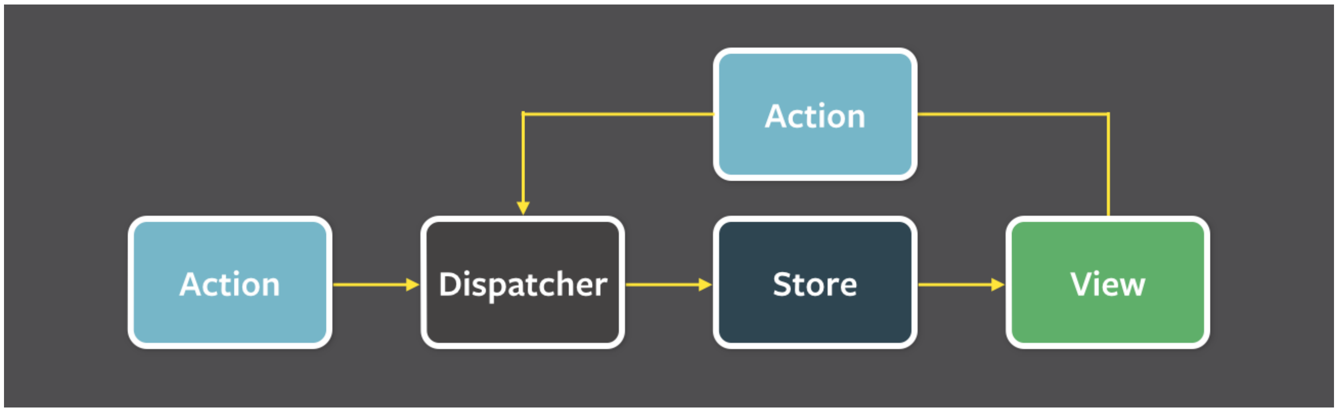
Username: ZDGharst

Password: secret

Self Evaluation:	[✓ Mastery] / Progressing / Needs Improvement
Due date:	2021-10-15
Completion:	2021-10-15
GitHub:	https://github.com/ZDGharst/FullStackOpen

Part 6 – State management with Redux

Part6a is concerned with teaching students about Flux architecture. Specifically, the Flux architecture used in this course is Redux. Flux architecture is intended to make state management easier: actions are dispatched to a store by components, then the view is updated as needed. Previously, components handled their own state and were handed states by other components using props (parameters); now, state is managed application wide in a store if that functionality is needed.



Even though Facebook has their own implementation for Flux architecture, they use Redux now. Actions are objects that contain their action type and the data; a reducer takes in the application state, an action, and outputs the new state after the action. Reducers are never explicitly used by developers; reducers are used when the store is created. By convention, state should be immutable even if the programming language doesn't force it to be. Action creators are helper functions that create action objects, then dispatch the action from there. Using the react-redux library allows for hooks to be used within React: selector and dispatch.

Within *Part6b*, additional reducers are created and combined to store and operate on complex state. A second reducer is made for handling the filter within the notes app, and the existing reducer is combined with the new one. Lastly, I installed Redux DevTools to debug and view the state/actions.

For *Part6c*, I connected the redux store to the backend server such that the actions retrieved, updated, and created within the backend as well as the frontend application. This required asynchronous actions, which aren't supported by default in react-redux.

Lastly, *Part6d*, has me relearn everything I've done so far for redux, but using an older version of redux that uses classes and object-oriented programming rather than functional programming with hooks. The connect method is used to create an instance of a store class, which is then mapped directly to components' props for access to the state and dispatch. Action creators are referenced as props and mapped. Lastly, the section refreshes the student about Presentational Components vs Container Components.

Additional notes can be found in the Part 6 README.

Part 7 – React router, custom hooks, styling app with CSS and webpack

This section of the course is the end for the original version of the course. As such, there are a lot of sections that are miscellaneous.

Part7a is only about React router. React router allows for a single page application to simulate the changing of view according to the user's URL bar. There are links that change the URL history state, and views are rendered according to the URL history. Even though no HTTP GET requests are made, it allows for bookmarking and linking to parts of the application. Each view could be implemented as its own components, and components are only rendered on that view. The React router library makes this easy. There is support for browser state to be automated when a link is clicked (through a custom Link component), support for parameters in the URL, and support for redirects.

Part7b covers making custom hooks. Custom hooks are useful for preventing the reuse of logic without making classes. Hooks have strict usage requirements and help keep rendering numbers low.

UI frameworks and styles are in *Part7c*. This part covers the libraries like Bootstrap (from Twitter), MaterialUI (from Google), and more. It's extremely dense, but it covers a lot of good information.

Lastly, *Part7d* covers the configurations which are handled for the developer when the create-react-app command is used. This covers everything from building the main JavaScript file, compile JSX to JavaScript using loaders (babel), adds polyfill for async/await, transpiles JavaScript code into an older form (from ES7 to ES5), loads CSS from inline components, maps source code to compiled code for debugging, and minifies the code. This part also covers the object-oriented side of React from older versions, just in case a developer who learns a functional style has to go back and work on legacy code.

Additional notes can be found in the Part 7 README.

Deliverables

For a completed list of exercises, please view the commits on the master branch here. There is a history browser, diff viewer, and the git messages explicitly define exercises.

All 23 exercises were completed in this section.

- Exercise 6.2: Mistakenly commited with the message "Complete exercise 5.2". Commit hash is 27cd6b0.
- Exercise 7.21: Doesn't have a GitHub commit because it's asking for feedback on course website.

Supplementary Material

The 100% correct way to structure a React app (or why there's no such thing)

Error Boundaries

React/Redux: pitfalls and best practices

7 Amazing Developer Tools that you're not using yet

Object Oriented vs Functional Programming with TypeScript

Abhinav Rastogi: Scaling NodeJS beyond the ordinary | JSConf Iceland 2018