Managing Application State with Redux

Overview of Redux

Manages underlying data

Application state can be easily *accessed*

Changing application state occurs only via actions

Redux state is provided to React components via *React-Redux*, a small connector library

For a full-length course on Redux, consider *Mastering Flux and Redux*

https://www.pluralsight.com/courses/flux-redux-mastering

Coming Up...

Create default application state as JSON file for development

Create basic Redux store to provide state to application as necessary

Changes to state (mutations) will be added later

Adding a Dashboard Component

Coming Up...

Add React dashboard component to add as a “home page” for end user

Dashboard will take application state that exists in DB and turn it into components that end user can interact with

Connect dashboard to Redux store

Routing and Navigation

Routing and Navigation

"Routing" is a term for when the form of the application is affected by the URL bar

`react-router` determines which React component to display based on URL

```
http://myapp/
user-1/
task-1/
edit
```

Good use of routing allows a lot of information to be codified in URL

Coming Up...

Add “main” component whose contents will change based on URL

Create new navigation component to go alongside dashboard

Additional routes will be added later

Adding New Tasks

Adding New Tasks

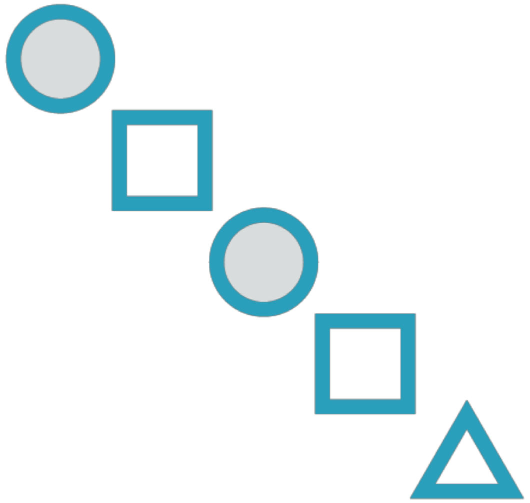
Reducer must be updated to allow tasks array to be changed

Tasks need random ID, reducers can't be random, therefore Saga or Thunk is needed

Updated state is reflected automatically in React component appearance

Sagas in Brief

Sagas run in the background of Redux applications



Respond to actions by generating "*side-effects*" (anything outside the app)

```
function*
```

One of only a few places where generator functions are found

Generators in Brief

Standard JavaScript functions (non-generator) return a single value, instantly

Generators can return any number of values, not just one

Generator values can be returned at a later time (asynchronously)

```
function* myGenerator(){

  let meaning = 42;

  while (true) {
    meaning += 1;
    yield meaning;
  }
}
```

- function*indicates special generator function type
- generator contains normal javascript code
- while (true) loops can exist in generator functions
- Yield keyword returns value to the generator’s caller (can return many values)
- Yields 43, 44, 45…

For a full-length course on Redux Saga, consider *Redux Saga*

https://www.pluralsight.com/courses/redux-saga

Demo

Create saga to generate random task ID, create task dispatch action containing details

We will create a “mock” saga which does not actually interact with the server (we have not written the server yet)

Update store reducer to recognize action and update tasks array accordingly

React components will update their appearance automatically

Implementing Task Details Route Part 1: Displaying Data

Using Mock Files During Development

Files with .mock extension indicate the file does not contain the true business logic

Used to reduce complexity (e.g., does not depend on server)

Mocks are commonly used in testing frameworks such as Jest

Demo

Add route which displays the details of a single task

Route will implement forms and buttons to allow user to change data

Router will be used to indicate which task should be viewed

Interactions which mutate the state will be added later

Implementing Task Details Route Part 2: Mutating Data

Demo

Add methods which *dispatch* actions when form elements of the task detail are interacted with

Add clauses to Redux reducer which causes state to be changed in response to relevant action

Front End Summary

Webpack is useful as it allows us to write code using imports and with JSX

Redux is a reliable and convenient way to store and manage our application state

React components often contain forms used by the end user

Using React-Redux, React components can update automatically to reflect data

Coming up in the Next Module…

Set up a server with Express

Install MongoDB and configure Node to communicate with it via Express

Create a simple REST API that will let us persist data on our server