Week 5 Day 3

Redux and React Testing

Flux Design Pattern

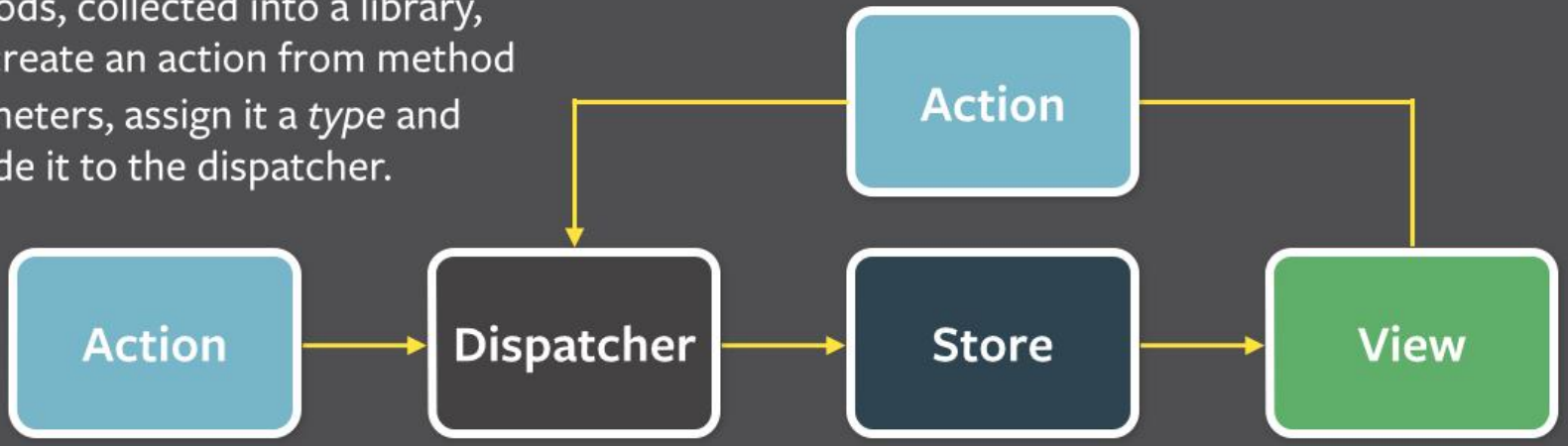
REVATURE

- Architecture to counteract Reacts one-way data flow
  - Pattern for passing data through an app
- Similar to context by using a central “store”
  - The store share application state
  - Uses dispatchers to send actions
  - Uses reducers to update the state

Flux Design Pattern: App Flow

REVATURE

Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.



Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

Source: https://facebook.github.io/flux/img/overview/flux-simple-f8-diagram-explained-1300w.png

Flux Design Pattern: Components

REVATURE

- Store:
  - Manages the application State
- Dispatcher:
  - Manages the flow of the application data
- Actions:
  - Object which describes the action to take on the state, with a possible payload
- Reducer:
  - Function that calculate the new state based on the action and old state
- View
  - The component

Flux: Pros and Cons

REVATURE

Benefits:

- State is global
  - No state lifting
  - Components subscribe to what they need
  - No prop drilling
- Single Source of Truth
- State is immutable directly

Disadvantages:

- More complicated
- More boiler plate*
- Does not work with async logic flow out of the box*
- Asterisks because redux toolkit improves these

Redux and React Redux

REVATURE

- Redux is a library for implementing flux in ANY JS application
  - npm i redux
- React-Redux provides redux implementation for react
  - npm i react-redux
- Redux Toolkit recommended way to use redux
  - npm i @reduxjs/toolkit
- Redux DevTools a browser extension to view changes in your state

Redux: Application Flow

REVATURE

1. The application store, holds some state, and references to reducers to manipulate the state
2. Use interacts with the page, causing an action to be dispatched.
3. The action is dispatched to the correct reducer
4. Reducer determines how to handle the action
5. The reducer returns the new state, and the store is updated
6. All components subscribed to the state are notified and updated

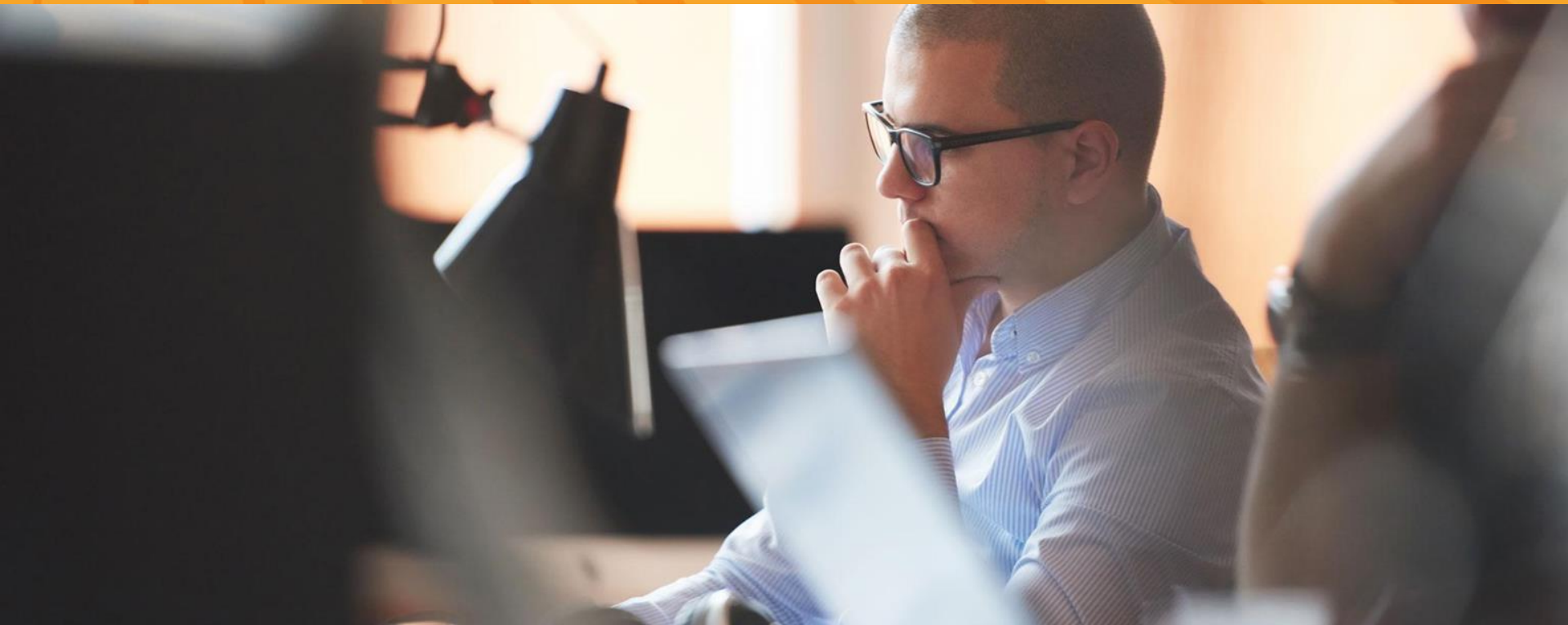
Redux: Thunks

REVATURE

- Thunks are a special function used to return data at a later date
  - Promises are built with thunks
- Redux uses Thunks because of unwanted side effects with Promises
- When using “vanilla” redux you must implement these manually
- Redux toolkit has built in functionality for Thunks

REVATURE

React Todo App with Redux



React Testing: Jest

REVATURE

- Open-source unit testing framework for React
  - Built by Facebook
  - Lightweight
  - Mocking capabilities
  - Ability to snapshot components at anytime
  - Easily test the DOM

Jest: Testing

REVATURE

- Define test suites in a `Component.test.ts` file
- The suite is defined by the describe() function
- Individual tests are described inside the test function with it() function
- Provided by jest are several evaluation functions to test your code
  - expect()
  - toBe()
  - toBeTruthy() / toBeFalsey()

Jest: Mocking

REVATURE

- Similarly to Mockito, Jest provides functionality to mock functions, return values and modules
  - Mock functions/return values when we only need the data returned
- Mock functions with jest.fn()
- Spy on functions with jest.spyOn()
- Mock modules with jest.mock()

React Testing Library

REVATURE

- Reacts replacement for Enzyme
  - Enzyme is no longer supported past React 17
- Allows you to “mount”/render your component
- Used to verify elements are displayed properly in the component

REVATURE

Testing in React