Week 4 Workshop

React Course

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes
Check-In	10 minutes
Weekly Recap: Controlled & Uncontrolled Forms, Validation, Redux, and more	50 minutes
Task 1:	50 minutes
BREAK	15 minutes
Tasks 2 & 3	90 minutes
Check-Out	15 minutes

Check-In

- How was this week? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- You must complete all Exercises before beginning the Workshop Assignment. We will review the Exercises together next, along with this week's concepts.

Week 4 Recap – Overview

Some of the New Concepts You Learned This Week

- Regular Expressions
- JavaScript: Ternary Operator, Computed Property Names, Spread Syntax
- Controlled Forms
- Binding Event Handlers
- Uncontrolled Forms
- Form Validation
- MVC & Flux Architectures
- Redux
- React Redux Form & Validation

Next slides will review these concepts,
along with some of the Exercises you did this week.

Computed Property Names

ES6 computed property names – variables can be passed into the square brackets instead of the property name being hard coded. Computed property names (the square bracket syntax) = syntactic sugar; we could do this before ES6, but it took more work.

```
this.setState({
    [name]: value
});
```

Controlled Forms

- HTML form elements such as `<input>`, `<textarea>`, and `<select>` already keep their own internal state
- In a controlled form, a React class component holds the form input data in state and renders the form, and uses event handlers to keep track of changes to the form
- The component controls the form and is the "*single source of truth*" for the form data

Exercise Recap: Controlled Forms

More takeaways:

- In React, HTML attribute `for` becomes `htmlFor`
- `JSON.stringify` converts an object to flattened text string
- Optional: Receive event object as the parameter of an event handler method, e.g. handleInputChange(event)
  - Then you can access methods and properties of event object if you need them
  - ('event' here is an arbitrary name, could use 'e' or 'evt' or something else)
- `event.preventDefault` will stop event from completing its default behavior, such as stopping a form from being submitted (and thereby reloading the page)
- `event.target` will let you access the element that the event was on and its attributes, e.g. `event.target.name, event.target.value`

Uncontrolled Forms

- Generally you want to use controlled forms in React
- For small, simple forms, or when you are mixing React with non-React code, you may wish to use uncontrolled forms
- Uncontrolled forms let the DOM handle the form data as usual
- Instead of using an event handler for state updates, you use a React ref to get form values from the DOM
  - With Reactstrap, use innerRef

Using refs (innerRef in Reactstrap)

When we rendered our Header component, we passed a callback function into an innerRef attribute of <Input>, like this:

```
<Input type="text" id="username" name="username"
innerRef={(input) => this.username = input} />
```

This allowed the component to access the value of this input elsewhere as: `this.username.value` without having to set up event handlers, binding, event.target, etc. Use sparingly.

MVC – Model View Controller

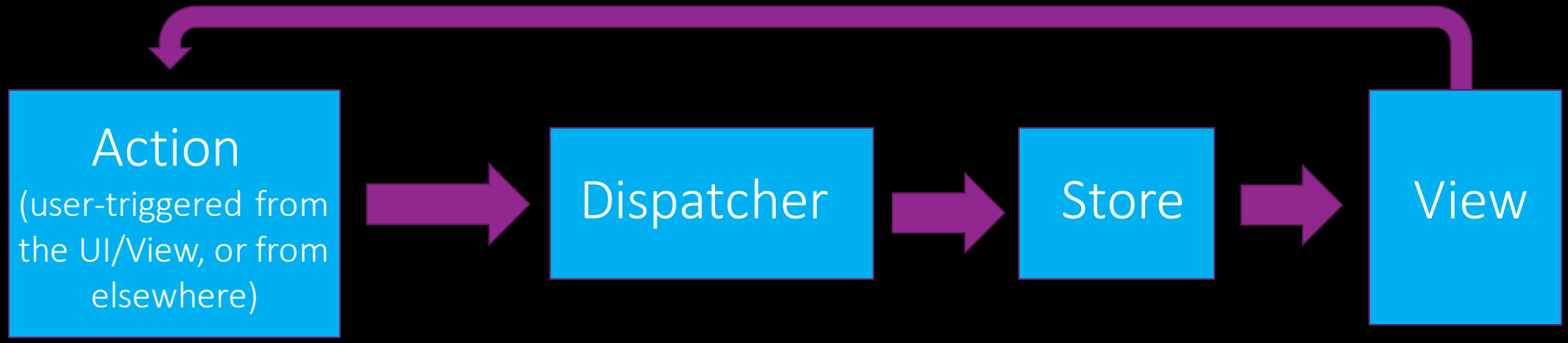
- Software architecture pattern
  - *not* a library or framework, a conceptual approach for writing software
- Key feature of MVC: *Separation of Concerns* – independent development, testing & maintenance for each concern:
  - Model – manages data/logic
  - View – the user interface (UI)
  - Controller – handles user input and updates model/view
- Widespread pattern used in both desktop and web app development
  - Thus, while React doesn't use MVC, at least be aware of this pattern
  - First proposed 1978 for designing GUIs
  - Many variants – MVVM (Model View ViewModel), MVW (Model View Whatever), etc

What the Flux?

- React was originally positioned as just the "View" part of MVC
- MVC does not care about data flow direction
- Facebook found that with React, being unopinionated about data flow direction caused issues at large scale
- Facebook engineers created the Flux architecture pattern to use with React in 2014
- Key feature: Unidirectional (one-way) data flow

Flux (cont)

- Flux is "a pattern for managing data flow in your application"
- Key parts: Actions, Dispatcher, Store, View



Flux vs Redux

- Key differences:

Flux	Redux
Pattern (no actual code)	Library (actual code)
Multiple stores	Single store
Mutability	Immutability
Single central dispatcher object, all actions must go through it, central manager for multiple stores	No single central dispatcher object, has a central store and a dispatcher function to dispatch actions

Redux

- Created 2015 - inspired by the *Flux* pattern (as well as the Elm programming language, the Immutable.js library, other influences)
- Flux is a *pattern*, Redux is a *library* that can be considered as an implementation of the Flux pattern
- While both Flux & Redux were developed for use with React, they are separate from React and can be used with other projects. You can also use React without using Flux pattern or Redux library!
- Redux works well with React for more complex apps with lots of components & state changes. Not a one size fits all solution – may not be needed for simple apps or mixed code cases.

Redux (cont)

- Redux is "a predictable JavaScript state container" - you will hear/read this a lot. What does this mean? Here's Redux's creator's soundbite on it:

I know about React, Redux etc.. but when someone asks you what exactly is "Predictable State Container" in Redux, what should be the answer?

5

Dan Abramov · Jan 3, 2016

It's a "state container" because it holds all the state of your application. It doesn't let you change that state directly, but instead forces you to describe changes as plain objects called "actions". Actions can be recorded and replayed later, so this makes state management predictable. With the same actions in the same order, you're going to end up in the same state.

Three Main Principles of Redux

1. *Single source of truth*
   - Single state object tree within a single **store**
2. *State is read-only*
   - Changes only made through **actions**
3. *Changes are made with pure functions* (***reducers***)
   - Takes previous state and action and returns next state without mutating the previous state (immutability)

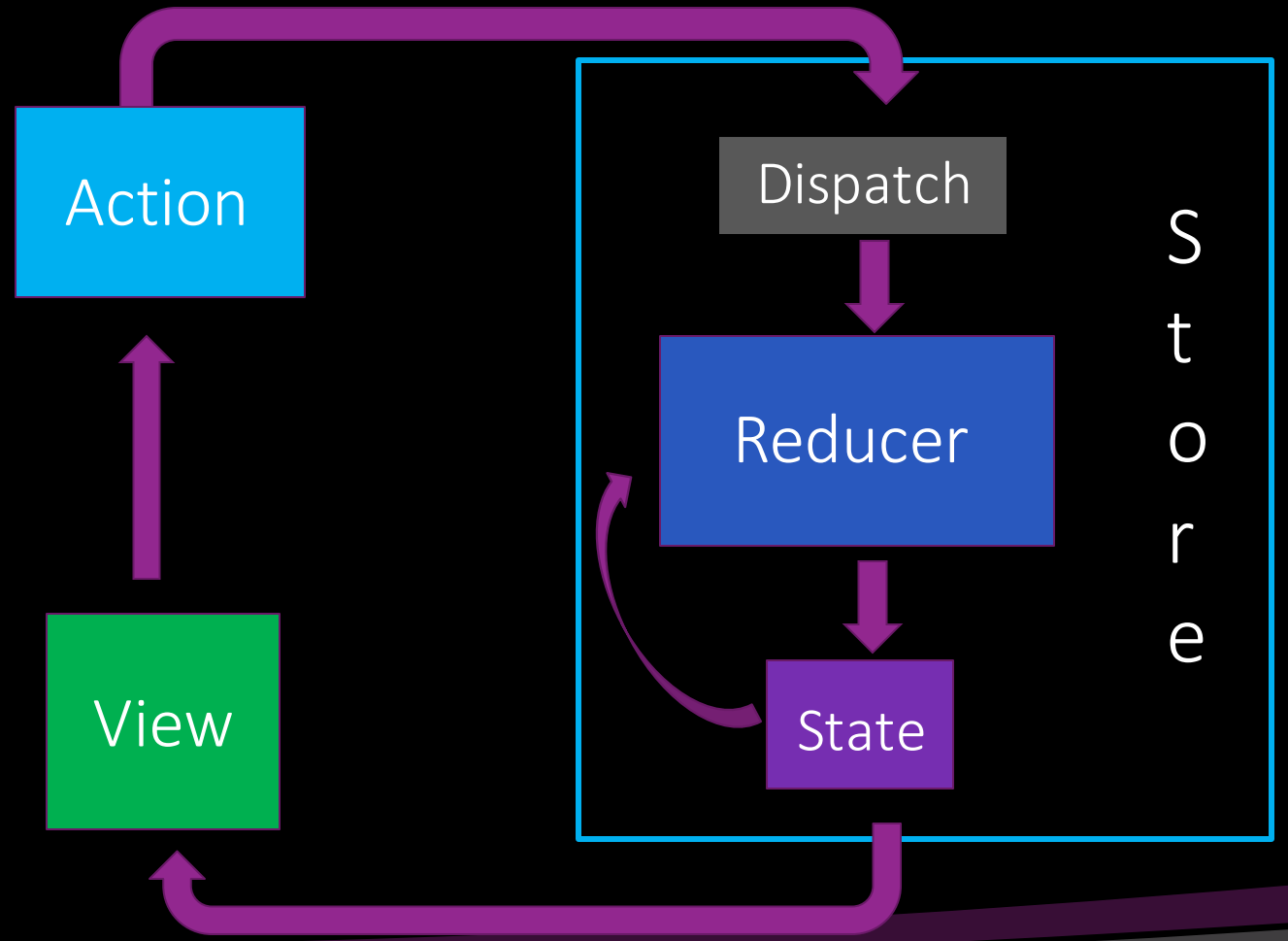
Redux (cont)

- *State*: Stored as JS object in the Redux Store
- *Action*: JS object that prescribes what state change to make
- *Reducer*: Pure function that takes current state and action, then returns new state
  - Updates state immutably (does not modify inputs)
  - Cannot call any impure functions like Math.random(), nor perform any side effects like API calls
  - Called *reducer* because it reduces multiple inputs (current state, action) to a single return value (similar to a function that you might use with Array.reduce)

Redux Store

- Holds the current state value
- Created using `createStore()`
- Supplies 3 methods:
  - `dispatch()`: begins state update by dispatching the provided action
  - `getState()`: returns the current stored state value
  - `subscribe()`: accepts a callback function that will be run every time an action is dispatched (you will not use this method typically, the connect() function from react-redux calls it for you)

Redux Data Flow



React with Redux

- react-redux library
  - The `connect()` function connects a React component to the Redux store. Under the hood, it wraps your component in a container component that subscribes to the store. It takes 4 optional arguments, the first 2 of which are:
    - `mapStateToProps()`: a function that's called every time store state changes, passes store state to wrapped component as props. You must include this argument in order to access store state.
    - `mapDispatchToProps()`: is used for dispatching actions to the store. We have not used this yet but we will next week.
  - Surround your root component with `<Provider>`
    - Must take the store as an attribute: `<Provider store={store}>`
    - Makes it possible for connected nested components to access the store

Exercise Recap: React Redux Form Validation

In this exercise, you:

- Reinstated form validation in Contact form by using `validators` attribute in Control and `<Errors>` from React Redux Form
- Used another ***regex*** for validating email – think about how much code it would take to write this check without using regex:

```
const validEmail = (val) => /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i.test(val);
```

Workshop Assignment

- It's time to start the workshop assignment!
- Sit near your workshop partner.
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - 10-minute rule does *not* apply to talking to your partner, you should consult each other throughout.
- Follow the workshop instructions very closely
  - both the video and written instructions.
- Talk to your instructor if any of the instructions are unclear to you.

Check-Out

- Submit your assignment file: **CampsiteInfoComponent.js**
- Wrap up – Retrospective
  - What went well
  - What could improve
  - Action items
- Start with Week 5, review more Week 4, or work on your Portfolio Project