REACT
The Marmite Framework
What is React?

• A library for building user interfaces
• Strictly concerns itself with the view
• Does not require a browser
• You need to bring your own pattern for managing data
Key Concepts

- One way binding
- Component based
- Virtual DOM
How does it work?
Getting Started

• This used to be tough but its recently been made easy:

  npm install -g create-react-app
  create-react-app hello-world

• Quick look at the output
ES6

• Many of these examples will include ES6

• Using a mixture of transpiling and polyfills Babel converts this to ES5 for broad compatibility in browsers

• We’ll look at just enough to make sense of the examples
ES6 - Destructuring

- Uses to extract data from arrays or objects into distinct variables

```javascript
var o = { x: 10, y: 20 }
var {x, y} = o
var {x: a}
console.log(x) // 10
console.log(y) // 20
console.log(a) // 10
```
ES6 - Class

- Classes in ES6 are just syntactic improvements over the existing prototype based inheritance

```javascript
class Animal {
}

class Dog extends Animal {
    constructor(name, age) {
        this.name = name
        this.age = age
    }
}

var myDog = new Dog('Tess', 5)
```
ES6 - Modules

- Take elements of CommonJS (Node) modules and AMD (RequireJS) modules

```javascript
export const myVariable = 5
export function dosomething() { ... }
export default class MyClass { ... }
```
Simple ToDo List Example

https://jsfiddle.net/jrandall/f7wn69ma/
Components

- A React application is constructed of a hierarchy of components
- Components often contain rendering code but don’t have to (and we’ll see some examples later)
Virtual DOM

• React uses the component model, properties and state to build a virtual DOM

• The virtual DOM is much faster to manipulate than the real DOM

• As components change (via properties or state) React updates its virtual DOM and uses this to calculate the most optimal way to update the real DOM
JSX

- JSX is not HTML
- It’s a shortcut for building the virtual DOM using React’s DOM API
- It transpiles to JavaScript (usually via Babel)
- It can provoke interesting reactions!!
Philosophical thoughts on JSX

• It just flips traditional templating on its head

• Rather than learn a templating DSL with “code” inside HTML JSX takes the other view - keep things in JavaScript

• Over time I’ve found myself scratching my head less over the JSX approach than I have over Angular 1’s template language
State and Properties

• Properties (props) are immutable and used to share state between components

  • Data and callbacks are passed down

  • Components use the callbacks to communicate change back up the component tree

• State is mutable and scoped within a component

• You should strive to make your components stateless and either compute the “state” from properties in render() or use the container pattern to pre-shape the properties
Component Lifecycle

Mount

- `componentWillMount()`
- `componentDidMount()`

Update

- `componentWillReceiveProps` ()
- `shouldComponentUpdate()`
- `componentWillUpdate()`
- `render()`

Unmount

- `componentWillUnmount()`
Immutability and JavaScript

• JavaScript itself has no inbuilt support for immutable data

• You can use the Immutable library to add support
  • http://facebook.github.io/immutable-js/
  • Implements immutable versions of many common data structures
  • Works well with ES6 and TypeScript, transpiles to ES3
Moving on from ToDo

• The ToDo example was simple but we’re quickly going to hit problems with those patterns:
  • Data / state management was complex
  • Everything was in one file
  • It’s not clear how to test it
Redux

• It bills itself as a predictable state container for JavaScript applications

• It’s a simplified implementation of the Flux pattern

• It’s not tightly coupled to React and you can use it with other frameworks (Angular 2 for example)

• It does however work very well with React
How does it work?

React Provider

Store

Reducer

Read Existing State

Set New State

Dispatch Action on event

Rendered View
The 3 Redux Principles

• It’s a single source of truth for your whole application
• State is read only
• Changes are made with pure functions called reducers
Store

- The store holds the state for the application
- Allows state to be retrieved through `getState()`
- Facilitates changes to state through the dispatch of actions
- Allows for listeners to be registered
State in Redux

- Should be thought of as a serializable model
- Don’t form none-hierarchically links between objects but use references (IDs etc.)
- If you can take state from a service or storage and place it directly in the store that's a good rule of thumb
Actions

• Simple payloads of data

• Should contain a type property

• Actions are created by action creators: functions that return an action. Though with middleware not always
Reducers

• Reducers are pure functions that take the existing state and an action and return the new state:
  \((\text{existingState}, \text{action}) \Rightarrow \text{newState}\)

• State is immutable so the reducer must base the new state on a copy of the existing state (more on this later) - it cannot change the existing state

• Because deep copying is expensive its common to reuse objects that haven’t changed in the new state tree
Container Components

- Container components are used to connect UI components to the state tree
- Structure data and behaviour to presentation components
- Leave presentation components to concentrate purely on presentation and have no dependencies on the rest of the application
Returning New State

- When dealing with complex models this can get difficult

- `Object.assign` is a common option but that can lead to quite complex code as you balance copying with reusing existing objects

- There is an add-on package for React that helps with this
update()

- Get it from npm:
  `npm install react-addons-update —save`
- Uses a Mongo like syntax for updating state
- Example: ToDo sample reworked to use update
Redux Middleware

- Middleware is run after an action is dispatched and before it reaches a router.
- Within middleware you have access to the dispatch() and getState() methods of the store.
- Can be used to observe to wrap around the action and reduce process or get involved with it.
Tools, Testing and Building
Tools

(and a more complicated example)

• React Developer Tools

• Redux Developer Tools
Testing

• Using React and Redux leads to a clean separation of concerns and a structure that lends itself to testing

• Jest is the Facebook framework for testing React applications

• Jest mocks dependencies by default. You can set application wide exclusions and per test exclusions.
Testing Redux

• Most of your testing will be focused on reducers
• As they are pure functions they are simple to test
  • Construct pre-state
  • Execute reducer
  • Run expectations against returned state
• Quick example!
Testing React Components

• When testing presentational components you’re normally interested in verifying that given state $x$ output $y$ is rendered and doesn’t change

• You could verify this using the virtual DOM

• However Jest includes a “snapshot” feature to save you a lot of typing

• Example!
Deploy to Azure with VSTS

- VSTS includes everything you need to build and deploy React apps

- Example!
Thanks

• Contacting me:
  • Email: james@accidentalfish.com
  • Twitter: @azuretrenches
  • GitHub: https://github.com/jamesRandall/
  • Blog: http://www.azurefromthetrenches.com

• Slides and sample code will be online in the next few days