Mobile App Development with React Native

Lectures

- Overview, JavaScript
- JavaScript, ES6
- React, JSX
- Components, Props, State, Style
- Components, Views, User Input
- Debugging
- Data
- Navigation
- Expo Components
- Redux
- Performance
- Shipping, Testing

Projects

- Project 0
- Project 1
- Project 2
- Final Project

Mobile App Development with React Native

Jordan Hayashi

Course Information

- Website
- Slack
- Staff email

Lectures

- Short break halfway
- Have a question? Interrupt me!
  - Concepts constantly build on each other, so it’s important to understand everything.
  - If something isn’t important to know, I’ll let you know
  - Staff will be monitoring Slack during lecture
- I love live examples!
  - Live coding has its risks. Let me know if you spot an error

Lecture 0: Overview, JavaScript

Jordan Hayashi

JavaScript is Interpreted

- Each browser has its own JavaScript engine, which either interprets the code, or uses some sort of lazy compilation
  - V8: Chrome and Node.js
  - SpiderMonkey: Firefox
  - JavaScriptCore: Safari
  - Chakra: Microsoft Edge/IE
- They each implement the ECMAScript standard, but may differ for anything not defined by the standard

Syntax

```
const firstName = "jordan";
const lastName = 'Hayashi';
const arr = ['teaching', 42, true, function() {
console.log('hi') }];

// hi I'm a comment
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

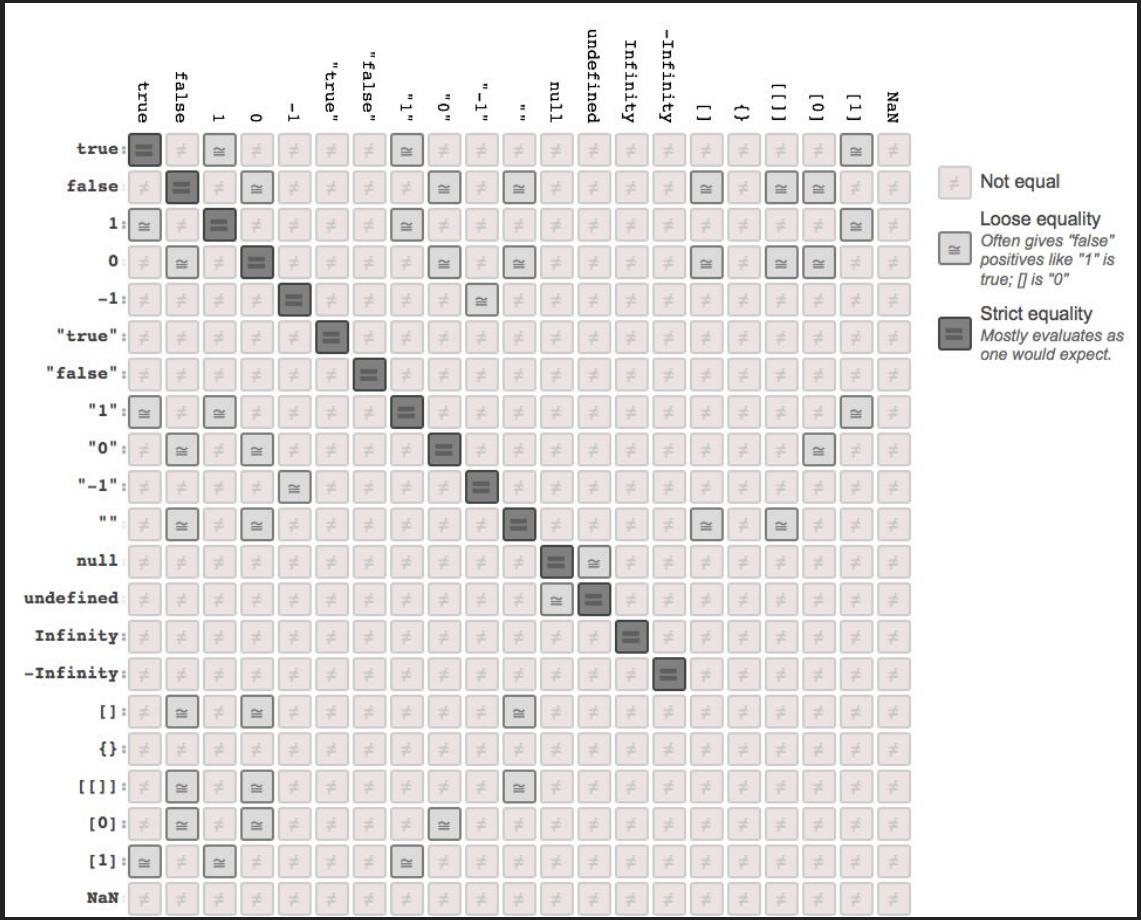
Types

- Dynamic typing
- Primitive types (no methods, immutable)
  - undefined
  - null
  - boolean
  - number
  - string
  - (symbol)
- Objects

Typecasting? Coercion.

- Explicit vs. Implicit coercion
  - `const x = 42;`
  - `const explicit = String(x);    // explicit === "42"`
  - `const implicit = x + "";       // implicit === "42"`

- `== vs. ===`
  - `==` coerces the types
  - `===` requires equivalent types



https://github.com/dorey/JavaScript-Equality-Table

Coercion, cont.

- Which values are falsy?
  - `undefined`
  - `null`
  - `false`
  - `+0, -0, NaN`
  - `""`
- Which values are truthy?
  - `{}`
  - `[]`
  - Everything else

Objects, Arrays, Functions, Objects

- ^ did I put Objects twice?
- Nope, I put it 4 times.

- Everything else is an object
- Prototypal Inheritance (more on this later)

Primitives vs. Objects

- Primitives are immutable
- Objects are mutable and stored by reference

- Passing by reference vs. passing by value

Prototypal Inheritance

- Non-primitive types have a few properties/methods associated with them
  - Array.prototype.push()
  - String.prototype.toUpperCase()
- Each object stores a reference to its prototype
- Properties/methods defined most tightly to the instance have priority

Prototypal Inheritance

- Most primitive types have object wrappers
  - String()
  - Number()
  - Boolean()
  - Object()
  - (Symbol())

Prototypal Inheritance

- JS will automatically “box” (wrap) primitive values so you have access to methods

```
42.toString()          // Errors
const x = 42;
x.toString()           // "42"
x.__proto__            // [Number: 0]
x instanceof Number    // false
```

Prototypal Inheritance

- Why use reference to prototype?
- What’s the alternative?
- What’s the danger?

Scope

- Variable lifetime
  - Lexical scoping (var): from when they’re declared until when their function ends
  - Block scoping (const, let): until the next } is reached
- Hoisting
  - Function definitions are hoisted, but not lexically-scoped initializations
- But how/why?

The JavaScript Engine

- Before executing the code, the engine reads the entire file and will throw a syntax error if one is found
  - Any function definitions will be saved in memory
  - Variable initializations will not be run, but lexically-scoped variable names will be declared

The Global Object

- All variables and functions are actually parameters and methods on the global object
  - Browser global object is the `window` object
  - Node.js global object is the `global` object

Lecture 1: JavaScript, ES6

Jordan Hayashi

Previous Lecture

- Types
- Coercion
- Objects
- Prototypal Inheritance
- Scope
- JS Execution
- Global Object
- Closures...

ES5, ES6, ES2016, ES2017, ES.Next

- ECMAScript vs JavaScript
- What do most environments support?
- Transpilers (Babel, TypeScript, CoffeeScript, etc.)
- Which syntax should I use?

Closures

- Functions that refer to variables declared by parent function still have access to those variables
- Possible because of JavaScript’s scoping

Immediately Invoked Function Expression

- A function expression that gets invoked immediately
- Creates closure
- Doesn’t add to or modify global object

First-Class Functions

- Functions are treated the same way as any other value
  - Can be assigned to variables, array values, object values
  - Can be passed as arguments to other functions
  - Can be returned from functions
- Allows for the creation of higher-order functions
  - Either takes one or more functions as arguments or returns a function
  - map(), filter(), reduce()

Synchronous? Async? Single-Threaded?

- JavaScript is a single-threaded, synchronous language
- A function that takes a long time to run will cause a page to become unresponsive
- JavaScript has functions that act asynchronously
- But how can it be both synchronous and asynchronous?

Asynchronous JavaScript

- Execution stack
- Browser APIs
- Function queue
- Event loop

Execution Stack

- Functions invoked by other functions get added to the call stack
- When functions complete, they are removed from the stack and the frame below continues executing

Asynchronous JavaScript

- Execution stack
- Browser APIs
- Function queue
- Event loop

Asynchronous JavaScript

- Asynchronous functions
  - setTimeout()
  - XMLHttpRequest(), jQuery.ajax(), fetch()
  - Database calls

Callbacks

- Control flow with asynchronous calls
- Execute function once asynchronous call returns value
  - Program doesn’t have to halt and wait for value

Promises

- Alleviate “callback hell”
- Allows you to write code that assumes a value is returned within a success function
- Only needs a single error handler

Async/Await

- Introduced in ES2017
- Allows people to write async code as if it were synchronous

this

- Refers to an object that’s set at the creation of a new execution context (function invocation)
- In the global execution context, refers to global object
- If the function is called as a method of an object, `this` is bound to the object the method is called on

Setting `this` manually

- bind(), call(), apply()
- ES6 arrow notation

Browsers and the DOM

- Browsers render HTML to a webpage
- HTML defines a tree-like structure
- Browsers construct this tree in memory before painting the page
- Tree is called the Document Object Model
- The DOM can be modified using JavaScript

Assignment

- Create a TODO app
- Will use JS DOM manipulation

Lecture 2:
React, Props, State

Jordan Hayashi

Previous Lecture

- ES6 and beyond
- Closures
- IIFEs
- First-Class Functions
- Execution Stack
- Event Loop
- Callbacks
- Promises and Async/Await
- `this`

Classes

- Syntax introduced in ES6
- Simplifies the defining of complex objects with their own prototypes
- Classes vs instances
- Methods vs static methods vs properties
- `new`, `constructor`, `extends`, `super`

React

- Allows us to write declarative views that “react” to changes in data
- Allows us to abstract complex problems into smaller components
- Allows us to write simple code that is still performant

Imperative vs Declarative

- How vs What
- Imperative programming outlines a series of steps to get to what you want
- Declarative programming just declares what you want

React is Declarative

- Imperative vs Declarative
- The browser APIs aren’t fun to work with
- React allows us to write what we want, and the library will take care of the DOM manipulation

React is Easily Componentized

- Breaking a complex problem into discrete components
- Can reuse these components
  - Consistency
  - Iteration speed
- React’s declarative nature makes it easy to customize components

React is Performant

- We write what we want and React will do the hard work
- Reconciliation - the process by which React syncs changes in app state to the DOM
  - Reconstructs the virtual DOM
  - Diffs the virtual DOM against the DOM
  - Only makes the changes needed

Writing React

- JSX
  - XML-like syntax extension of JavaScript
  - Transpiles to JavaScript
  - Lowercase tags are treated as HTML/SVG tags, uppercase are treated as custom components
- Components are just functions
  - Returns a node (something React can render, e.g. a `<div />`)
  - Receives an object of the properties that are passed to the element

Props

- Passed as an object to a component and used to compute the returned node
- Changes in these props will cause a recomputation of the returned node (“render”)
- Unlike in HTML, these can be any JS value

State

- Adds internally-managed configuration for a component
- `this.state` is a class property on the component instance
- Can only be updated by invoking `this.setState()`
  - Implemented in React.Component
  - setState() calls are batched and run asynchronously
  - Pass an object to be merged, or a function of previous state
- Changes in state also cause re-renders

todoApp.js

But why limit React to just web?

React Native

- A framework that relies on React core
- Allows us build mobile apps using only JavaScript
  - “Learn once, write anywhere”
- Supports iOS and Android

Lecture 3: React Native

Jordan Hayashi

Previous Lecture

- Classes
- React
- Imperative vs Declarative Programming
- Props
- State
- todoApp.js
- React Native...

React Native

- A framework that relies on React core
- Allows us build mobile apps using only JavaScript
  - “Learn once, write anywhere”
- Supports iOS and Android

How does React Native work?

- JavaScript is bundled
  - Transpiled and minified
- Separate threads for UI, layout and JavaScript
- Communicate asynchronously through a “bridge”
  - JS thread will request UI elements to be shown
  - JS thread can be blocked and UI will still work

Differences between RN and Web

- Base components
- Style
- No browser APIs
  - CSS animations, Canvas, SVG, etc.
  - Some have been polyfilled (fetch, timers, console, etc.)
- Navigation

React Native Components

- Not globally in scope like React web components
  - Import from `'react-native'`
- `div` → `View`
- `span` → `Text`
  - All text must be wrapped by a `<Text />` tag
- `button` → `Button`
- `ScrollView`

https://facebook.github.io/react-native/docs/components-and-apis.html

Style

- React Native uses JS objects for styling
- Object keys are based on CSS properties
- Flexbox layout
  - Default to column layout
- Lengths are in unitless numbers
- `style` prop can take an array of styles
- `StyleSheet.create()`
  - Functionally the same as creating objects for style
  - Additional optimization: only sends IDs over the bridge

Event Handling

- Unlike web, not every component has every interaction
- Only a few “touchable” components
  - `Button`
  - `TouchableOpacity`, `TouchableHighlight`, `TouchableWithoutFeedback`
  - `TouchableNativeFeedback` (Android only)
- Web handlers will receive the event as an argument, but React Native handlers often receive different arguments
  - Consult the docs

Components

- Return a node (something that can be rendered)
- Represent a discrete piece of the UI
- “All React components must act like pure functions with respect to their props.”
- Two types:
  - Stateless Functional Component (SFC) a.k.a. Pure Functional Component
  - `React.Component`

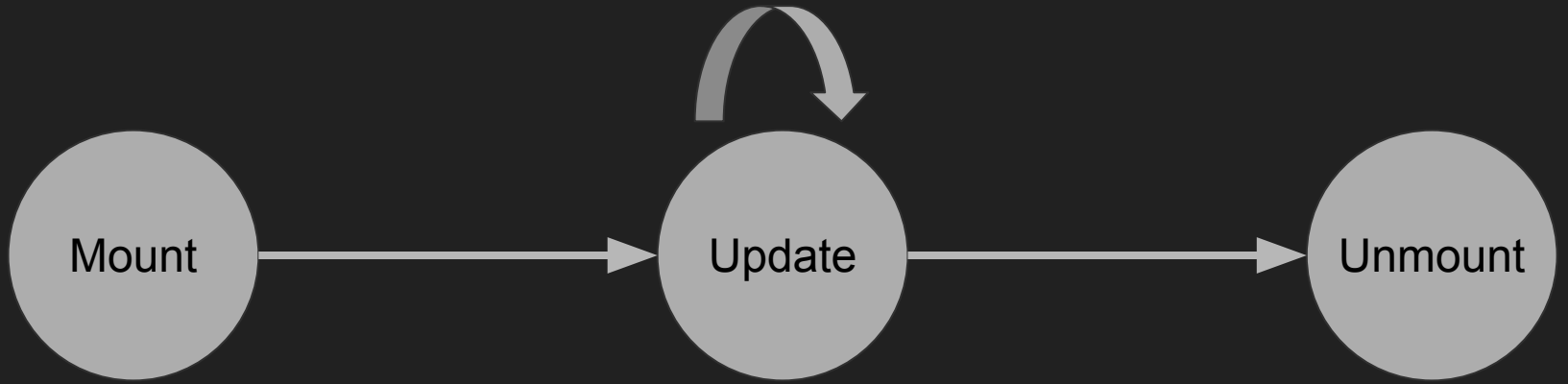
Stateless Functional Component (SFC)

- Simplest component: use when you don’t need state
- A function that takes props and returns a node
  - Should be “pure” (it should not have any side effects like setting values, updating arrays, etc.)
- Any change in props will cause the function to be re-invoked

React.Component

- An abstract class that can be extended to behave however you want
- These have additional features that SFCs don’t
  - Have instances
  - Maintain their own state
  - Have lifecycle methods (similar to hooks or event handlers) that are automatically invoked
- Rendering now becomes a function of props and class properties

Component Lifecycle



Mount

- `constructor(props)`
  - Initialize state or other class properties (bound methods, etc.)
- `render()`
  - The meat of a component
  - Return a node
- `componentDidMount()`
  - Do anything that isn’t needed for UI (async actions, timers, etc.)
  - Setting state here will cause a re-render before updating the UI

Update

- `componentWillReceiveProps(nextProps)`
  - Update any state fields that rely on props
- `shouldComponentUpdate(nextProps, nextState)`
  - Compare changed values, return true if the component should rerender
    - If returned false, the update cycle terminates
  - Almost always a premature optimization
- `render()`
- `componentDidUpdate(prevProps, prevState)`
  - Do anything that isn’t needed for UI (network requests, etc.)

Unmount

- `componentWillUnmount()`
  - Clean up
    - Remove event listeners
    - Invalidate network requests
    - Clear timeouts/intervals

Writing React Native

Expo

- “The fastest way to build an app”
- Suite of tools to accelerate the React Native development process
  - Snack - runs React Native in the browser
  - XDE - a GUI to serve, share, and publish your Expo projects
  - CLI - a command-line interface to serve, share, and publish projects
  - Client - runs your projects on your phone while developing
  - SDK - bundles and exposes cross-platform libraries and APIs

Import/Export

- Components are great for simplifying code
- We can split components into their own files
  - Helps organize project
  - Export the component from the file
- Import the component before using it in a file
- Default vs named import/export

PropTypes

- React can validate the types of component props at runtime
- Development tool that allows developers to ensure they’re passing correct props
- Helps document your components’ APIs
- Only runs in development mode

How to Read Docs

- Have a goal in mind
- See what the library/framework/API offers
- Find something that solves your problem
- Configure using the exposed API

Lecture 4:
Lists, User Input

Jordan Hayashi

Previous Lecture

- React Native
- Style
- Event Handling
- Stateless Functional Components
- Components
- Lifecycle Methods
- Expo
- Import/Export
- PropTypes

contacts.js

Lists

- In web, browsers will automatically become scrollable for content with heights taller than the window
- In mobile, we need to do that manually
  - `ScrollView`
  - `ListView` (deprecated)
  - `FlatList`, `SectionList`

`ScrollView`

- The most basic scrolling view
- Will render all of its children before appearing
- To render an array of data, use `.map()`
  - Components in an array need a unique `key` prop

https://facebook.github.io/react-native/docs/scrollview.html

`FlatList`

- A performant scrolling view for rendering data
- “Virtualized:” only renders what’s needed at a time
  - Only the visible rows are rendered in first cycle
  - Rows are recycled, and rows that leave visibility may be unmounted
- Pass an array of `data` and a `renderItem` function as props
- Only updates if props are changed
  - Immutability is important

https://facebook.github.io/react-native/docs/flatlist.html

`SectionList`

- Like `FlatList` with additional support for sections
- Instead of `data` prop, define `sections`
  - Each section has its own `data` array
  - Each section can override the `renderItem` function with their own custom renderer
- Pass a `renderSectionHeader` function for section headers

https://facebook.github.io/react-native/docs/sectionlist.html

User Input

- Controlled vs uncontrolled components
  - Where is the source of truth for the value of an input?
- React recommends always using controlled components
- Pass `value` and `onChangeText` props

https://facebook.github.io/react-native/docs/textinput.html

Lecture 5:
User Input, Debugging

Jordan Hayashi

Previous Lecture

- `ScrollView`
- `FlatList`
- `SectionList`
- Controlled vs Uncontrolled components
- `TextInput`

User Input

- Controlled vs uncontrolled components
  - Where is the source of truth for the value of an input?
- React recommends always using controlled components
- Pass `value` and `onChangeText` props

https://facebook.github.io/react-native/docs/textinput.html

Handling multiple inputs

- `<form>` exists in HTML, but not in React Native
- With controlled components, we maintain an object with all inputs' values
- We can define a function that handles the data to submit

Validating Input

- Conditionally set state based on input value
- Validate form before submitting
- Validate form after changing single input value
  - `this.setState()` can take a callback as the second argument
  - `componentDidUpdate()`

KeyboardAvoidingView

- Native component to handle avoiding the virtual keyboard
- Good for simple/short forms
  - The view moves independent of any of its child `TextInputs`

Debugging

- React errors/warnings
- Chrome Developer Tools (devtools)
- React Native Inspector
- react-devtools

React Errors and Warnings

- Errors show up as full page alerts
  - Trigger with `console.error()`
- Warnings are yellow banners
  - Trigger with console.warn()
  - Does not appear in production mode

Chrome Devtools

- Google Chrome has amazing developer tools (debugger)
- We can run the JavaScript inside a Chrome tab
  - Separate threads for native and JavaScript
  - Communicate asynchronously through bridge
  - No reason that the JavaScript needs to be run on device

React Native Inspector

- Analogous to the Chrome element inspector
- Allows you to inspect data associated with elements, such as margin, padding, size, etc.
- Does not allow you to live-edit elements

`react-devtools`

- “Inspect the React component hierarchy, including component props and state.”
- Install with `` `npm install -g react-devtools` ``
- Run with `` `react-devtools` ``
- Allows us to make live-edits to style, props, etc.

https://github.com/facebook/react-devtools

External Libraries

- Libraries are code written outside the context of your project that you can bring into your project
- Since React Native is just JavaScript, you can add any JavaScript library
- Install using `` `npm install <library>` ``
  - Use the --save flag for npm@"<5"
  - Use the -g flag to install things globally
- Import into your project
  - `import React from 'react'`

Lecture 6: Navigation

Brent Vatne & Eric Vicenti

Previous Lecture

- User input with `TextInput`
- Simple input validation
- `KeyboardAvoidingView`
- Debugging
  - Errors and warnings
  - Chrome Developer Tools
  - React Native Inspector with react-devtools
- Installing external libraries with `npm`

What is navigation?

- Navigation is a broad term that covers topics related to how you move between screens in your app
- Web navigation is oriented around URLs
- Mobile apps do not use URLs for navigating *within* the app*
- Navigation APIs completely different on iOS and Android
  - Several React Native libraries provide a platform agnostic alternative
  - We will talk about one of them today, React Navigation

* Linking into a mobile app with a URL is known as deep linking: https://v2.reactnavigation.org/docs/deep-linking.html

React Navigation and alternatives

- Two distinct approaches
  1. Implement mainly in JavaScript with React
  2. Implement mostly in native, expose an interface to JavaScript for existing native navigation APIs
- React Navigation takes approach #1

Read more at https://v2.reactnavigation.org/docs/pitch.html and https://v2.reactnavigation.org/docs/alternatives.html

Install React Navigation

- `npm install react-navigation@2.0.0-beta.5 --save`
- This will install the latest *pre-release* version at the time of writing. Typically you would just write `npm install react-navigation --save` to use the latest *stable* version.
- If you refer back to this in the future, keep in mind that this material is all specific to the 2.x series of releases.

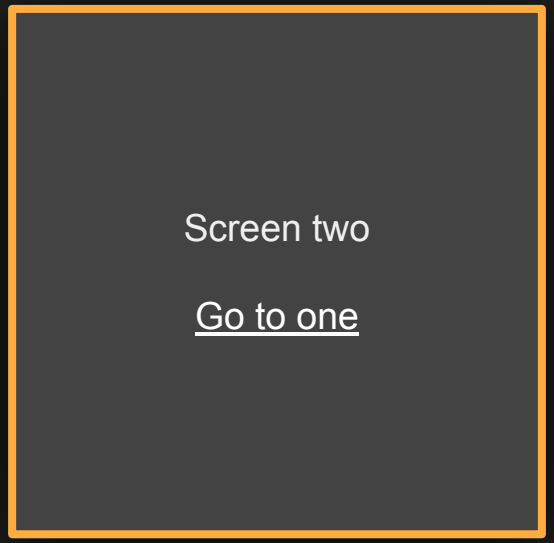
Navigators, routes, and screen components

- A navigator is a component that implements a navigation pattern (eg: tabs)
- Each navigator must have one or more routes.
  - A navigator is a parent of a route.
  - A route is a child of a navigator.
- Each route must have a name and a screen component.
  - The name is usually unique across the app
  - The screen component is a React component that is rendered when the route is active.
  - The screen component can also be another navigator.

Switch Navigator

- Display one screen at a time
- Inactive screens are unmounted
- The only action a user can take to switch from one route to another

Switch Navigator



Creating a navigator

```
import { createSwitchNavigator } from 'react-navigation';

const AppNavigator = createSwitchNavigator({
  "RouteNameOne": ScreenComponentOne,
  "RouteNameTwo": ScreenComponentTwo,
});
```

Rendering a navigator

```
const AppNavigator = createSwitchNavigator({
  "RouteNameOne": ScreenComponentOne,
  "RouteNameTwo": ScreenComponentTwo,
});

export default class App extends React.Component {
  render() {
    return <AppNavigator />
  }
}
```

- `createSwitchNavigator` is a function that returns a React component
- We render the component in our root App component. Usually we only *explicitly* render one navigator per app because navigators are composable.

Higher order components

- `createSwitchNavigator` is a Higher Order Component: it is a function that returns a React component.
- “A higher-order component (HOC) is an advanced technique in React ***for reusing component logic***.”
- This is similar to higher order functions, which are functions that either take functions as arguments or return a function as a result.

Read more at https://reactjs.org/docs/higher-order-components.html

Navigating to another route

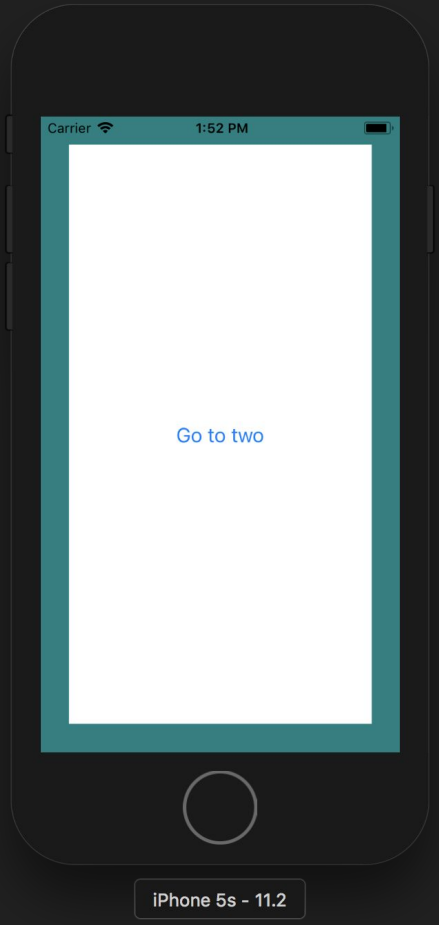
```
class ScreenComponentOne extends React.Component {
 render() {
   return (
     <Button
       title="Go to two"
       onPress={() => this.props.navigation.navigate('RouteNameTwo')}
     />
   );
 }
}
```

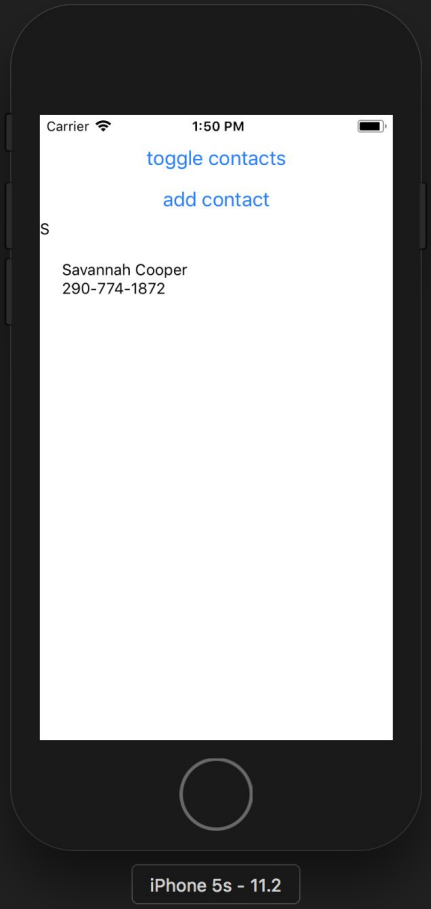
The `navigation` prop

- **`navigate(..)`**
- `goBack(..)`
- `setParams(..)`
- `getParam(..)`
- `dispatch(..)`
- `isFocused(..)`
- `addListener(..)`
- `state`

* The navigation prop is passed in to the screen component for each route.

Full reference: https://v2.reactnavigation.org/docs/navigation-prop.html


Carrier
1:52 PM
Go to two
iPhone 5s - 11.2


Carrier
1:50 PM
toggle contacts
add contact
S
Savannah Cooper
290-774-1872
iPhone 5s - 11.2

screenProps

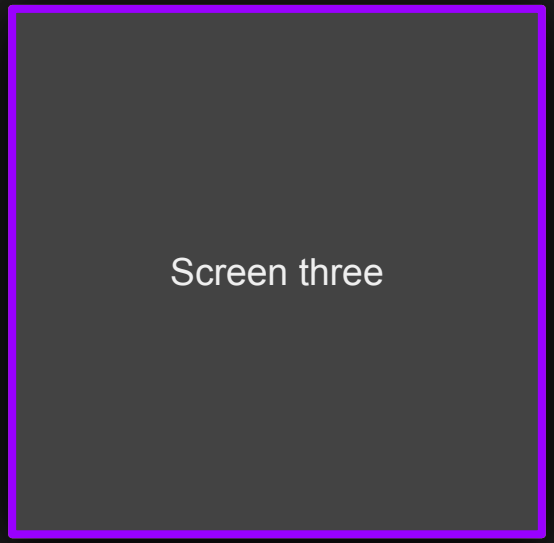
```
export default class App extends React.Component {
  render() {
    return <AppNavigator screenProps={/* object here */} />
  }
}
```

- Made available to every screen component in the navigator.
- Perfectly fine for very small applications and prototyping but inefficient for most meaningful applications - **every route in your app will re-render when `screenProps` changes**. Use a state management library or the React Context API instead.

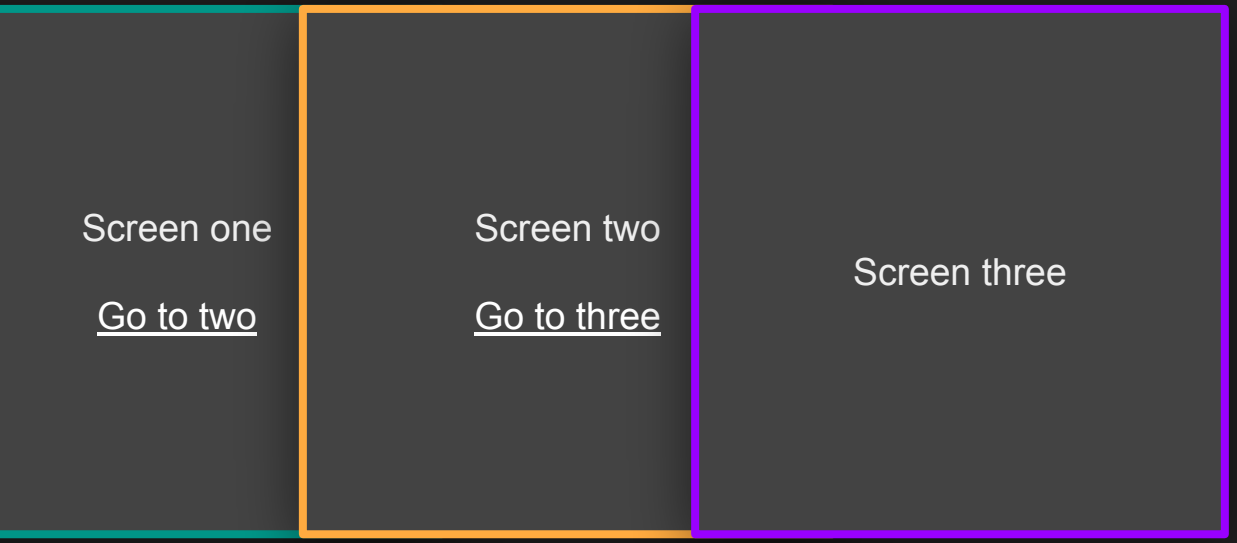
Stack Navigator

- Display one screen at a time
- The state of inactive screens is **maintained** and they remain mounted
- **Platform-specific layout, animations, and gestures**
  - Screens are stacked on top of each other
  - iOS: screens slide in from right to left, can be dismissed with left to right gesture. Modal screens slide in from bottom to top, can be dismissed with top to bottom gesture.
  - Android: screens fade in on top of each other, no dismiss gesture. Hardware back button dismisses the active screen.
- Users can push and pop items from the stack, replace the current item, and various other

Stack Navigator



Stack Navigator



Creating a StackNavigator

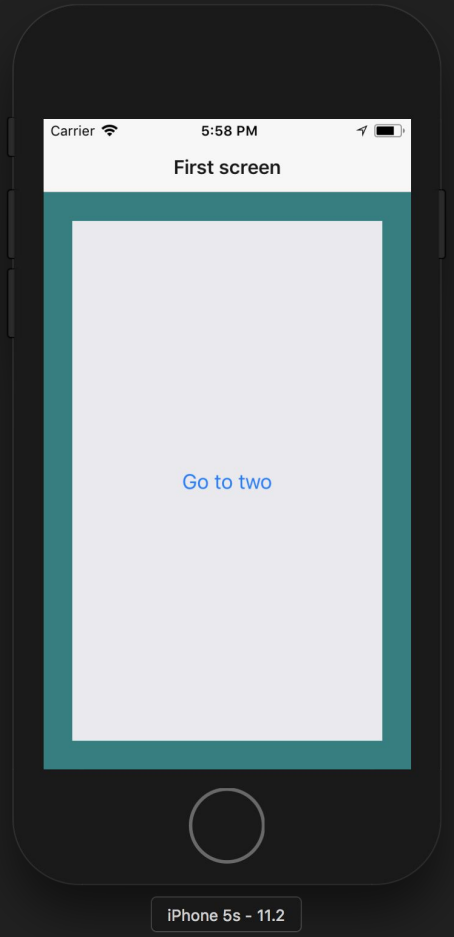
```
import { createStackNavigator } from 'react-navigation';

const AppNavigator = createStackNavigator({
 "RouteNameOne": ScreenComponentOne,
 "RouteNameTwo": ScreenComponentTwo,
});
```

Navigating to another route

```
class ScreenComponentOne extends React.Component {
 render() {
   return (
     <Button
       title="Go to two"
       onPress={() => this.props.navigation.navigate('RouteNameTwo')}
     />
   );
 }
}
```

Returning to the previously active route

```
class ScreenComponentThree extends React.Component {
 render() {
   return (
     <Button
       title="Go back"
       onPress={() => this.props.navigation.goBack()}
     />
   );
 }
}
```


Carrier
5:58 PM
First screen
Go to two
iPhone 5s - 11.2

Configuring `navigationOptions`

- `headerTitle`
- `headerStyle`
- `headerTintColor`
- `headerLeft`
- `headerRight`

Full list: https://v2.reactnavigation.org/docs/stack-navigator.html#navigationoptions...

Using params to pass state between routes

- `navigate` with params

```
this.props.navigation.navigate('RouteName', {
  paramName: 'value-of-param'
});
```

- `setParams` to update params for the route

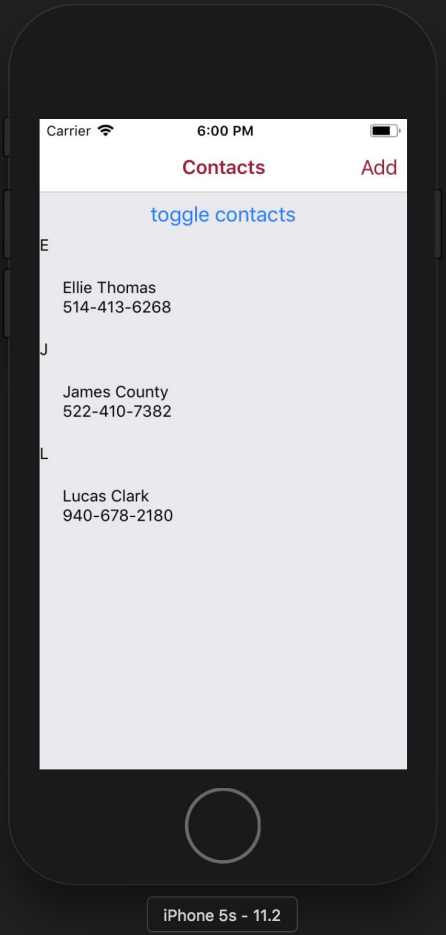
```
this.props.navigation.setParams({
 paramName: 'new-value-of-param',
});
```

- `getParam` to read a param

```
this.props.navigation.getParam('paramName', 'default-value');
```

```
// It's time for us to take a short break
this.props.navigation.navigate('BreakTime');
```

```
// Break time is over
this.props.navigation.goBack();
```


Carrier
6:00 PM
Contacts
Add
toggle contacts
E
Ellie Thomas
514-413-6268
J
James County
522-410-7382
L
Lucas Clark
940-678-2180
iPhone 5s - 11.2

Add button to header with `navigationOptions`

- `headerLeft`
- `headerRight`

Full list: https://v2.reactnavigation.org/docs/stack-navigator.html#navigationoptions...

```
// Jump to a screen, identified by route name
navigate('MyRouteName', { paramName: 'param-value' });

// “Push” a new screen, even if it already is in the stack
push('MyRouteName');
```

Stack specific navigation actions

- `push(..)`
- `pop(..)`
- `popToTop(..)`
- `replace(..)`

More information: https://v2.reactnavigation.org/docs/navigation-prop.html#...

Composing navigators

- Navigators can be composed when one type of navigation visually appears to be inside another navigator
- A navigator can be the Screen Component of another navigator
- The app should only contain one top-level navigator
- You can `navigate()` to any route in the app
- `goBack()` works for the whole app, supports Android back button

Composing navigators

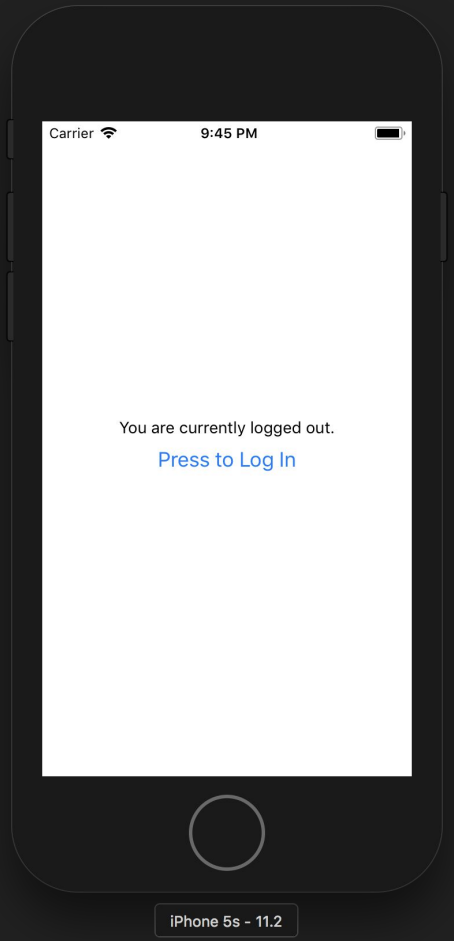
```
const MyStackNavigator = createStackNavigator({
  "Home": HomeScreen,
  "AddContact": AddContactScreen,
});
const AppNavigator = createSwitchNavigator({
  "Login": LoginScreen,
  "Main": MyStackNavigator,
});
```

Do not render a navigator inside a screen

```
class MyScreen extends React.Component {
  render() {
    return <MyStackNavigator />;
  }
}
```

Instead, set as a screen within the `AppNavigator`

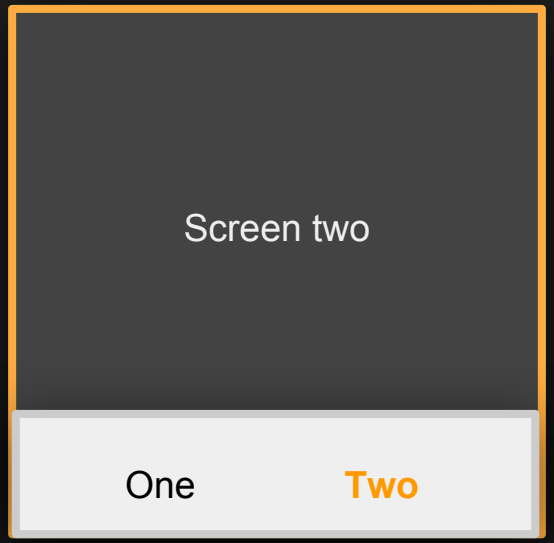
```
const AppNavigator = createSwitchNavigator({
  "Main": MyStackNavigator,
});
```


Carrier
9:45 PM
You are currently logged out.
Press to Log In
iPhone 5s - 11.2

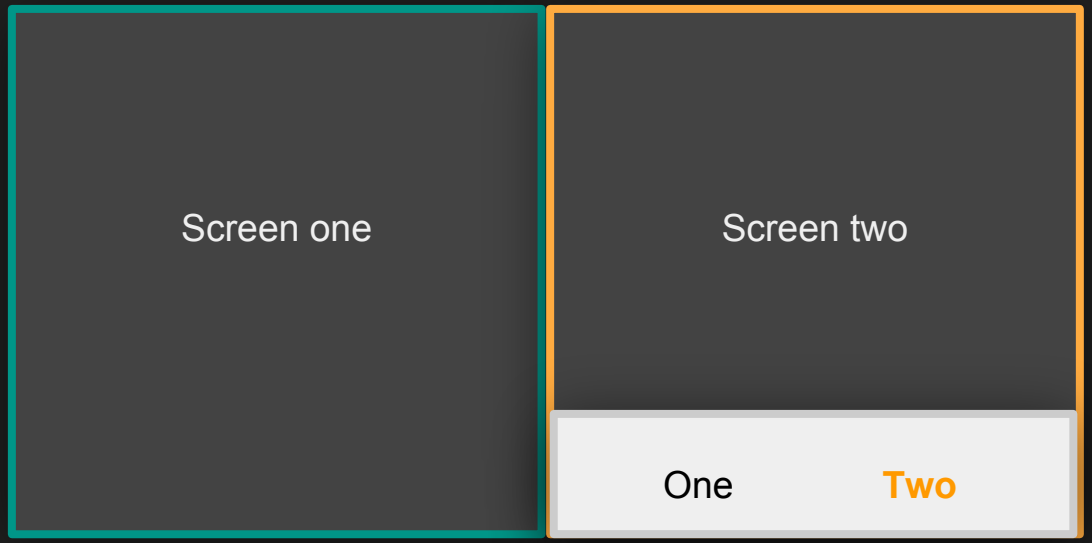
Tab navigators

- Display one screen at a time
- The state of inactive screens is maintained
- Platform-specific layout, animations, and gestures
  - `createMaterialTopTabNavigator`
  - `createMaterialBottomTabNavigator`
  - `createBottomTabNavigator`
- The `navigate()` action is used to switch to different tabs
- `goBack()` can be called to go back to the first tab
  - The tab navigator `goBack` behavior is configurable

Tab navigators

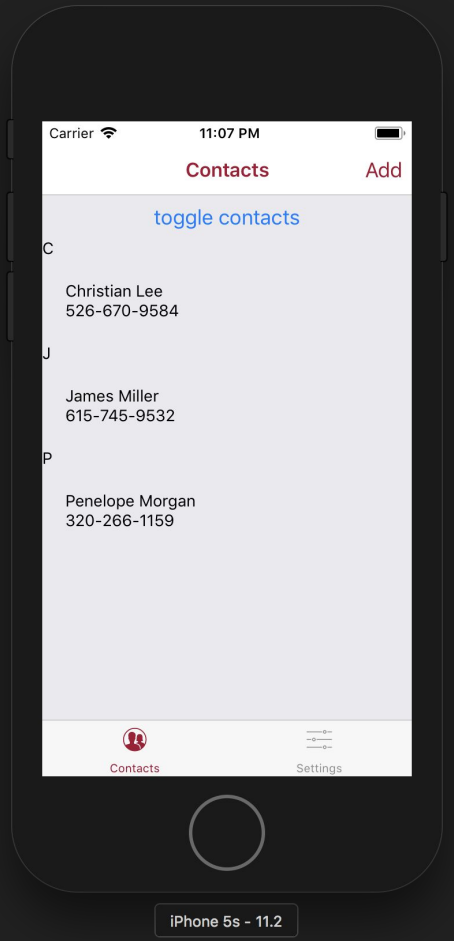


Tab navigators



Creating a tab navigator

```
const AppNavigator = createBottomTabNavigator({
  "TabOne": ScreenComponentOne,
  "TabTwo": ScreenComponentTwo,
});

export default class App extends React.Component {
  render() {
    return <AppNavigator />
  }
}
```


Carrier
11:07 PM
Contacts
Add
toggle contacts
C
Christian Lee
526-670-9584
J
James Miller
615-745-9532
P
Penelope Morgan
320-266-1159
Contacts
Settings
iPhone 5s - 11.2

Configure tab bar settings

```
const MainTabs = createBottomTabNavigator(
 {
   ...
 },
 {
   tabBarOptions: {
     activeTintColor: "#a41034"
   }
 }
);
```

Full reference for `tabBarOptions`: https://v2.reactnavigation.org/docs/tab-navigator.html#...

Configure tab icons

```
MainStack.navigationOptions = {
 tabBarIcon: ({ focused, tintColor }) => (
   <Ionicons
     name={`ios-contacts${focused ? "" : "-outline"}`}
     size={25}
     color={tintColor}
   />
 )
};
```

Full reference of options: https://v2.reactnavigation.org/docs/tab-navigator.html#...

Use common icon packs

```
# Install it in your shell
npm install --save react-native-vector-icons
```

```
// Import a supported icon set in your code
import Ionicons from "react-native-vector-icons/Ionicons";
```

```
// Use it as a React component
<Ionicons name="md-checkmark" size={25} color="#000" />
```

See other icon sets that are included: https://expo.github.io/vector-icons/

React Navigation Resources

- React Navigation Documentation
- React Navigation API Reference
- NavigationPlayground example source code

Lecture 7: Data

Jordan Hayashi

Previous Lecture

- `react-navigation`
- `SwitchNavigator`
- `navigation` prop
- `StackNavigator`
- Configuring navigators
- `TabNavigator`
- Composing navigators

Data

- Not all apps are self-contained
- Any app that wants to rely on information not computed within the app needs to get it from somewhere
  - Communicate with other resources using an API

API

- “Application Programming Interface”
- A defined set of ways with which a resource can be interacted
  - React components have APIs; you interact by passing props
  - A class has an API; you interact by invoking methods
  - A web service has an API; you interact by making network requests
- Providers often get to decide on the API, but sometimes it’s decided for them
- Consumers have to read docs to know how to use an API

https://randomuser.me/documentation

Making Network Requests

- `fetch()` is polyfilled
  - It's not natively part of JavaScript, but it is implemented to match the usage of the browser `fetch()`
- `fetch()` expects an URL and optionally some config
- `fetch()` returns a `Promise`, which is fulfilled with a `Response` object

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

https://developer.mozilla.org/en-US/docs/Web/API/Response

Promises

- Allows writing asynchronous, non-blocking code
- Allows chaining callbacks and/or error handlers
  - `.then()` - executed after the previous Promise block returns
  - `.catch()` - executed if the previous Promise block errors

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Async/Await

- Allows writing async code as if it were synchronous
  - Still non-blocking
- A function can be marked as `async`, and it will return a `Promise`
- Within an `async` function, you can `await` the value of another async function or `Promise`
- Use `try/catch` to handle errors

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await

Transforming Data

- Sometimes the shape of the data returned by an API isn’t ideal
  - Where should we do this “transformation?”
- Doing it early gives us an abstraction barrier and is more efficient

Authentication

- A process to determine if a user is who they say they are
- Generally done using a name and password
- But how do we send the name and password in the request?

HTTP Methods

- GET
  - The default in browsers and in `fetch()`
  - Add parameters in the url by appending a `?` and chaining `key=value` pairs separated by &
- POST
  - Submit data (e.g. a form) to an endpoint
  - Parameters are included in the request body
  - If POSTing JSON, must have `content-type: application/json` header and body must be JSON string

HTTP Response Codes

- Every network response has a “code” associated with it
  - 200: OK
  - 400: Bad Request
  - 403: Forbidden
  - 404: Not Found
  - 500: Internal Server Error
  - 418: I’m a teapot

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Guest Lecture for CSCI E-39b
April 2, 2018

Charlie Cheever <ccheever@expo.io> @ccheever


Expo

Maps and Location

- MapView
- Location
- Geocoding

Text or Delete

- Contacts

A Compass

- Magnetometer
- DeviceMotion

A Multimedia Board

- Video
- Audio
- Fonts

A Photo Editor

- ImagePicker
- Simple Camera
- Camera View
- GestureHandler

Other APIs

- Push Notifications
- Calendar
- Gyroscope, Accelerometer, Pedometer
- OpenGL View
- Facebook Login, Google Login
- Ads (AdMob/Google, Facebook)
- Fingerprint Scanner
- SecureStore
- SQLite
- Analytics (Amplitude, Segment)
- Sentry (stacktraces)

See https://docs.expo.io/

Notes

https://github.com/ccheever/harvard-guest-lecture

Lecture 9: Redux

Jordan Hayashi

Previous Lectures

- APIs
- Making Network Requests
- Promises, Async/Await
- Data Transformations
- Authentication
- HTTP Methods
- HTTP Response Codes
- Expo Components

Scaling Complexity

- Our apps have been relatively simple, but we’re already starting to see bugs related to app complexity
  - Forgetting to pass a prop
  - Directly managing deeply nested state
  - Duplicated information in state
  - Not updating all dependent props
  - Components with large number of props
  - Uncertainty where a piece of data is managed

Scaling Complexity: Facebook

- Facebook found the MVC architecture too complex for their scale
- The complexity manifested itself into bugs
- Facebook rearchitected into one-way data flow

https://youtu.be/nYkdrAPrdcw?t=10m22s

Flux

- “An application architecture for React utilizing a unidirectional data flow”
  - The views react to changes in some number of “stores”
  - The only thing that can update data in a store is a “dispatcher”
  - The only way to trigger the dispatcher is by invoking “actions”
  - Actions are triggered from the views
- Many implementations
  - https://github.com/facebook/flux
  - https://github.com/reactjs/redux
    - Whether redux is an implementation of Flux is an opinion that can be argued either way

Redux

- A data management library inspired by Flux
- Single source of truth for data
- State can only be updated by an action that triggers a recomputation
- Updates are made using pure functions
- Action → Reducer → Update Store

https://redux.js.org

simpleRedux/

Reducer

- Takes the previous state and an update and applies the update
- Should be a pure function
  - Result is deterministic and determined exclusively by arguments
  - No side effects
- Should be immutable
  - Return a new object
- What is responsible for invoking the reducer?

Store

- Responsible for maintaining state
- Exposes getter via getState()
- Can only be updated by using dispatch()
- Can add listeners that get invoked when state changes

Actions

- An action is a piece of data that contains the information required to make a state update
  - Usually objects with a type key
  - https://github.com/redux-utilities/flux-standard-action
- Functions that create actions are called action creators
- Actions must be dispatched in order to affect the state

`simpleRedux` → `redux`

- Our redux implementation has a very similar API
  - Missing a way to notify that state has updated
- How do we get the info from the store to our components?
  - `store.getState()`
- How do we update the store?
  - `store.dispatch()`
- How do we get the application to update when the store changes?

Review: HOCs

- Higher-Order Components take components as arguments or return components
- We could create a HOC that does the following:
  - Check for state updates and pass new props when that happens
  - Automatically bind our action creators with `store.dispatch()`
- We'd also need to subscribe to store updates
- https://github.com/reactjs/react-redux

Lecture 10: Async Redux, Tools

Jordan Hayashi

Previous Lecture

- Scaling Complexity
- Flux
- Redux
- `simpleRedux/`
- Reducers
- Store
- Actions
- `react-redux`

Review: `react-redux`

- React bindings for redux
  - `<Provider />`
  - `connect()`
- `Provider` gives children access to our redux store
- `connect()` helps us subscribe to any subset of our store and bind our action creators

Async simpleRedux/

Supporting Async Requests

- Where do we want to add this support? How do we change our API?
  - Reducers
  - Store
  - Actions
  - Action creators
- We need to change more than just the action creators
- `Store.dispatch()` needs to accept other types
- Our addition is unideal, since we had to change our redux implementation

Redux Middleware

- This allows us to extend redux without having to touch the implementation
- Any function with this prototype can be middleware
  - `({getState, dispatch}) => next => action => void`
- We can reimplement our feature as middleware
- https://github.com/gaearon/redux-thunk
  - “A thunk is a function that wraps an expression to delay its evaluation”

Persisting State

- Our app can now be a pure function of the redux store
- If we can persist the store, we can reload the app into the current state
- React Native provides `AsyncStorage`
  - “Use an abstraction on top of `AsyncStorage` instead of using it directly for anything more than light usage since it operates globally.”

redux-persist

- Abstracts out the storage of the store into `AsyncStorage`
- Gives us `persistStore`, `persistReducer`, `PersistGate`
  - Automatically stores the state at every change
  - Automatically rehydrates the store when the app is re-opened
  - Will display loading screen while waiting for store to rehydrate
- https://github.com/rt2zz/redux-persist

Container vs Presentational Components

- As an application grows in size and complexity, not all components need to be aware of application state
- Container components are aware of redux state
- Presentational components are only aware of their props

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Do I need Redux?

- Redux helps apps scale, but does add complexity
- Sometimes, the complexity overhead isn’t worth it
- Do as much as you can with local component state, then add redux if you hit pain points
  - Forgetting to pass a prop
  - Directly managing deeply nested state
  - Duplicated information in state
  - Not updating all dependent props
  - Components with large number of props
  - Uncertainty where a piece of data is managed

JavaScript Tools

- NPM
- Babel
- `@std/esm`
- Chrome devtools
- React/Redux devtools
- ESLint
- Prettier
- Flow/TypeScript

JavaScript Tools

- NPM
- Babel
- `@std/esm`
- Chrome devtools
- React/Redux devtools
- ESLint
- Prettier
- Flow/TypeScript

ESLint

- “A fully pluggable tool for identifying and reporting on patterns in JavaScript”
- Allows us to enforce code style rules and statically analyze our code to ensure it complies with the rules
  - Ensure style consistency across a codebase

https://github.com/eslint/eslint

ESLint: Setup

- Install
  - Per project: `` `npm install --save-dev eslint` ``
  - Globally: `` `npm install -g eslint` ``
- Create your own config
  - Per project: `` `./node_modules/.bin/eslint --init` ``
  - Globally: `` `eslint init` ``
- Or extend an existing config
  - https://github.com/airbnb/javascript
  - https://github.com/kensho/eslint-config-kensho

ESLint: Running

- Run on a file/directory
  - Per project: `` `./node_modules/.bin/eslint <path>` ``
  - Globally: `` `eslint <path>` ``
- Lint whole project by adding as an NPM script
- Most text editors have an integration

Prettier

- “Prettier is an opinionated code formatter”
- Prettier will rewrite your files to adhere to a specified code style
- It can integrate with ESLint
  - Specify an eslint config and pass `--fix` to `eslint` to have prettier auto-fix improper styling

https://github.com/prettier/prettier

Lecture 11: Performance

Jordan Hayashi

Previous Lecture

- Async `simpleRedux/`
- Redux Middleware
- `redux-persist`
- Container vs Presentational Components
- ESLint
- Prettier

Performance

- How quickly and efficiently something works
- Performance optimization is the process of making something work as efficiently as possible
- Performance optimization is a very wide field
  - Today we'll discuss optimizing on the JavaScript side of things
  - Mostly high-level, with examples

Trade-Offs

- Performance optimization usually comes at a complexity cost
  - In most cases, optimization is not worth the cost in complexity and maintainability
- Don’t over-optimize until a bottleneck is found
- How do we measure for bottlenecks?

Measuring Performance

- Be mindful of the environment setting of your application
- React Native Perf Monitor
  - Shows you the refresh rate on both the UI and JS threads
  - Anything below 60 means frames are being dropped
- Chrome Performance Profiler
  - Shows you a flame chart of all of your components
  - Only available in development mode

Common Inefficiencies

- Rerendering too often
- Unnecessarily changing props
- Unnecessary logic in mount/update

Rerendering Too Often

- Components will automatically rerender when they receive new props
  - Sometimes, a prop that isn't needed for the UI will change and cause an unnecessary rerender
- If you use redux, only subscribe to the part of state that is necessary
- `keys` in arrays/lists
- `shouldComponentUpdate()` and `React.PureComponent`
  - A `PureComponent` has a predefined `shouldComponentUpdate()` that does a shallow diff of props

Unnecessarily Changing Props

- Unnecessarily changing a value that is passed to a child could cause a rerender of the entire subtree
- If you have any object (or array, function, etc.) literals in your `render()` method, a new object will be created at each render
  - Use constants, methods, or properties on the class instance

Unnecessary Logic in Mount/Update

- Adding properties to class instance instead of methods on the class
  - Properties are created at each mount whereas methods are one time ever

Reminder: Trade-Offs

- Performance optimization usually comes at a complexity cost
  - In most cases, optimization is not worth the cost in complexity and maintainability
- Don’t over-optimize until a bottleneck is found

Animations

- Let’s add a progress bar to our Project 1 timer
- Animations require both the JS and UI threads
  - Sending messages over the bridge 10s of times per second is expensive
  - Blocking either thread impacts the UX
- We could implement the animation in native
  - Requires knowing Obj-C/Swift and Java
- What if we could declare the animation in JS and have it execute on the native thread?

Animated

- Allows us to declare a computation in JS and compute it on the native thread
  - JS thread no longer needs to compute anything
  - JS thread can be blocked and the animation will still run
- Cannot use native driver for layout props

https://facebook.github.io/react-native/docs/animated.html

Lecture 12: Deploying, Testing

Jordan Hayashi

Previous Lecture

- Performance Trade-Offs
- React Native Perf Monitor
- Chrome Performance Profiler
- Common Inefficiencies
  - Rerendering too often
  - Unnecessarily changing props
  - Unnecessary logic
- `Animated`

Deploying

- Deploy to the appropriate store by building the app and uploading it to the store
- Set the correct metadata in app.json
  - https://docs.expo.io/versions/latest/workflow/configuration
- Build the app using `exp` (command-line alternative to the XDE)
  - Install with `npm install --global exp`
  - Build with `exp build:ios` or `exp build:android`
  - Expo will upload the build to s3
  - Run `exp build:status` and paste the url in a browser to download

Deploying, cont.

- Upload to the appropriate store
  - https://docs.expo.io/versions/latest/distribution/building-standalone-apps
  - https://docs.expo.io/versions/latest/distribution/app-stores
- Deploy new JS by republishing from the XDE or exp
  - Rebuild app and resubmit to store to change app metadata

Testing

- The term “testing” generally refers to automated testing
- As an application grows in size, manual testing gets more difficult
  - More complexity means more points of failure
- Adding a test suite ensures that you catch bugs before they get shipped
- How do we know which parts of our app to test?

Test Pyramid

- Methodology for determining test scale/granularity
- Unit tests
  - Test an individual unit of code (function/class/method)
- Integration/Service tests
  - Test the integration of multiple pieces of code working together, independent of the UI
- UI/End-to-end tests
  - Test a feature thoroughly including the UI, network calls, etc.

Unit Tests

- Test an individual unit of code (function/class/method)
- Very granular and easy to tell what is breaking
- The most basic test is a function that notifies you when behavior is unexpected
- Testing frameworks give you additional benefits
  - Run all tests instead of failing on first error
  - Pretty output
  - Automatically watch for changes
  - Mocked functions

Jest

- “Delightful JavaScript Testing”
- A testing framework by Facebook
- Install with `npm install --save-dev jest`
- Run with `npx jest` or by adding script to `package.json`
  - Will automatically find and run any files that end in `.test.js` (or any other regex expression that you specify)

Jest: Testing Redux Actions

- We can replace our functions with Jest's `expect()`, `toBe()`, and `toEqual()`
- We can use snapshots to compare the output of a function to the previous output of the function
  - We get notified if the output changes
  - We don't need to rewrite tests if the change was intended
- Which should we use?
  - Use `toBe()` for primitives
  - Use `toEqual()` for objects that shouldn't change
  - Use snapshots for objects that may change

Jest: Testing Async Redux Actions

- Async functions add multiple difficulties
  - We have to wait for the result before testing against the result
  - Our tests may rely on imported libs
  - Our tests may rely on external services
- If we return a `Promise`, Jest will wait for it to resolve
  - Jest also supports `async/await`
- Jest supports mocking functions
- Dependency injection
  - Pass functions on which other functions rely as arguments
  - Allows us to mock functions that rely on external services

Integration Tests

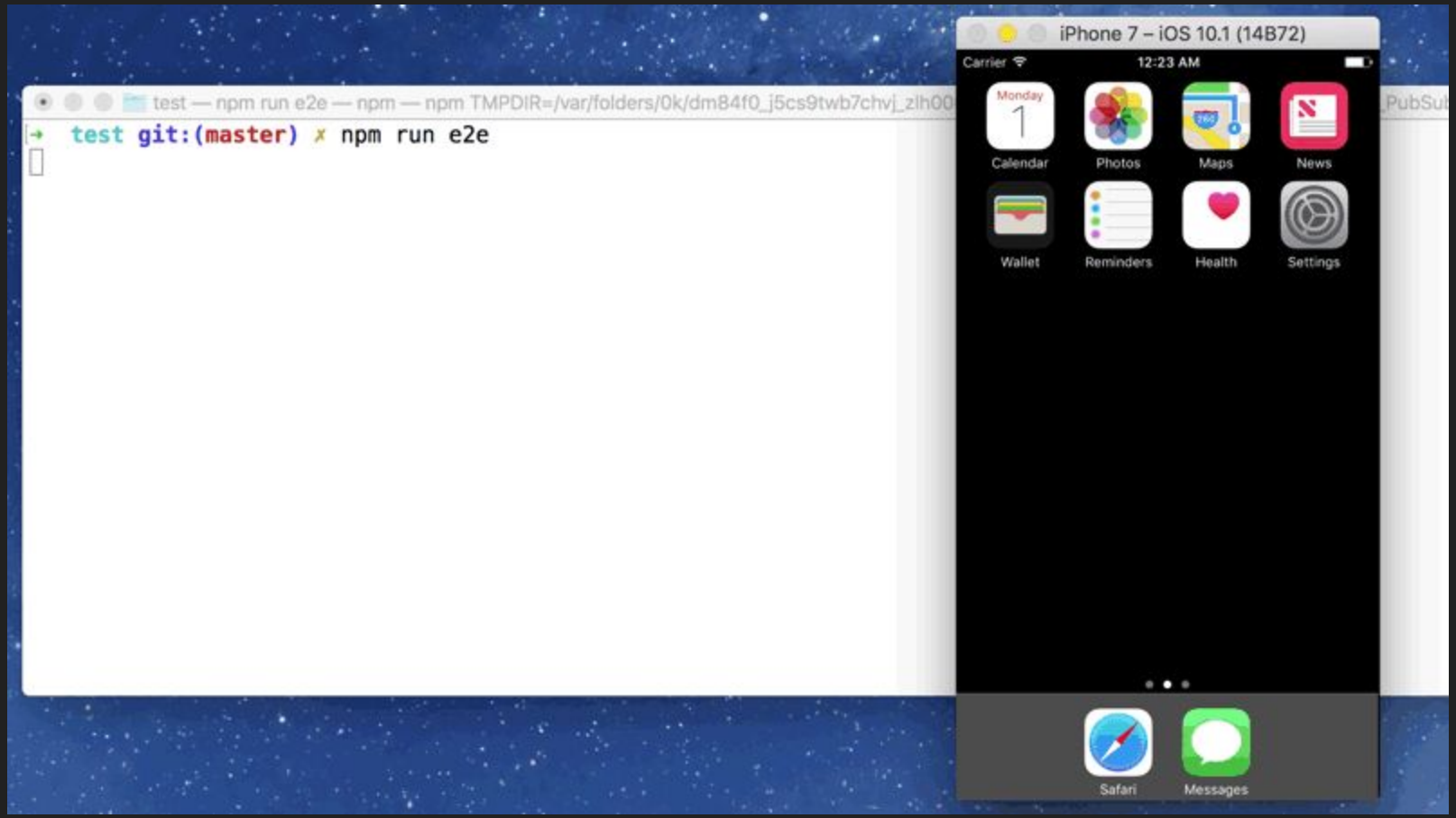
- We can use Jest's snapshot testing to test components
- `react-test-renderer` allows us to render components outside the context of an app
  - https://reactjs.org/docs/test-renderer.html
- jest-expo has all the configuration you need
  - https://github.com/expo/jest-expo

Code Coverage

- Metric for tracking how well-tested an application is
  - Statements: How many statements in the program have been executed?
  - Branches: How many of the possible code paths have been executed?
  - Functions: How many of the defined function been executed?
  - Lines: How many of the lines have been executed?
- Get coverage report by passing `--coverage` to `jest`

End-To-End Tests

- There is currently no easy way to run automated e2e tests in react native
- There is an awesome work-in-progress by Wix called Detox
  - https://github.com/wix/detox
  - https://github.com/expo/with-detox-tests
  - Lacks Android support


test — npm run e2e — npm — npm TMPDIR=/var/folders/0k/dm84f0_j5cs9twb7chvj_zlh00
→ test git:(master) ✗ npm run e2e
iPhone 7 – iOS 10.1 (14B72)
Carrier
12:23 AM
Monday
1
Calendar
Photos
Maps
News
Wallet
Reminders
Health
Settings
Safari
Messages

Thanks!