WEBINAR

React Native Jumpstart

mark@objectcomputing.com

REACT NATIVE OVERVIEW ...

- Open-source framework for building mobile apps with React, JavaScript, and CSS
- Developed by Facebook
- Uses native components, not WebViews
- Can mostly focus on React Native components without learning details of iOS and Android APIs
- Can integrate native code when needed
  - Java and Kotlin for Android; Swift and Objective-C for iOS

PREREQUISITES

- This talk assumes familiarity with JavaScript, React, and CSS

... REACT NATIVE OVERVIEW

- Uses CSS Flexbox for layout
- Supports automatic reloading in simulators and real devices
- Can debug with Chrome devtools
- Supports over-the-air (OTA) app updates without a new app review
- Used by
  - Adidas, Bloomberg, Facebook, Instagram, Tesla, Pinterest, Salesforce, Skype, Uber, Walmart, Wix, & more

WHY USE REACT NATIVE?

- Want to develop native mobile apps for Android and iOS
- Want most/all of the code to work on both
- Don't want to learn Java, Kotlin, or Swift
- Want to leverage expertise in JavaScript and CSS
- Maybe already know React
- Want automatic reloading
- Don't have high performance requirements (not a video game)

COMPETITORS

all open source

apps built on WebViews lack native look-and-feel and are slower than native apps

OBJECT COMPUTING

Name	From	Native or WebView	Programming Language	Frameworks Supported	Notes
PhoneGap	Adobe	WebView	JavaScript, HTML, & CSS	custom	• contributed to Apache in 2011 and renamed Cordova • development continued outside of Apache • now a fork of Cordova • uses plugins for accessing device capabilities
Cordova	Apache	WebView	JavaScript, HTML, & CSS	custom	• uses plugins for accessing device capabilities
Ionic	Drifty	WebView	JavaScript, HTML, & CSS	Angular (working on React & Vue)	• also creates web apps and Electron-based desktop apps • has commercial support for CI/CD and themes
Xamarin	Microsoft	Native	C#	custom	• also creates desktop apps for Windows and macOS • has free and enterprise tiers
NativeScript	Progress (Telerik)	Native	JavaScript	Angular & Vue	
React Native	Facebook	Native	JavaScript & CSS	React	
Vue Native		Native	JavaScript & CSS	Vue	• builds on React Native and can use all its features and components • transpiles `.vue` files to React components
Flutter	Google	Native	Dart	custom	• as of March 2019 not ready for serious use • Dart is a hot mess

SAME IN REACT AND REACT NATIVE

- Language - JS or TS
- Components - how defined
- Props
- State management
- Lifecycle methods
- Sending HTTP requests
- Linting tools - ESLint
- Formatting tools - Prettier
- Unit testing - Jest?

DIFFERENT IN REACT AND REACT NATIVE

- Elements used
  `div` vs. `View`, `span` vs. `Text`, ...
- Platform-specific components -
  Android and iOS
- Styling approach -
  CSS vs. style objects
- Debugging approach -
  web browser vs. simulators & Expo
- Deployment -
  web server vs. app stores
- Access to device capabilities -
  ex. camera
- End-to-end testing - Cypress vs. ?
- Devtools -
  browser extensions vs. simulators

RESOURCES

- React Native - https://facebook.github.io/react-native/
- React Native Elements - https://react-native-training.github.io/react-native-elements/
  - cross-platform toolkit
  - **`Avatar`**, **`Badge`**, **`Button`**, **`ButtonGroup`**, **`Card`**, **`CheckBox`**, **`Divider`**, **`Header`**, **`Icon`**, **`Image`**, **`Input`**, **`ListItem`**, **`Overlay`**, **`Pricing`**, **`Rating`**, **`SearchBar`**, **`Slider`**, **`SocialIcon`**, **`Text`**, **`Tile`**, **`Tooltip`**
- Awesome React Native - http://www.awesome-react-native.com/
- Expo - https://expo.io/
  - set of React Native tools, libraries and services
- npmjs.org - search for “react-native”; almost 16,000 packages as of 3/24/19

EXPO (v32)

- Free tool that wraps React Native and adds many features
- https://expo.io/
- Runs on macOS, Windows, and Linux
- Supports Android 5+ and iOS 10+
  - don't use if older versions must be supported
- Each version of Expo works with a specific version of React Native
- Simplifies updating to new versions of React Native

"Expo is kind of like Rails for React Native.
Lots of things are set up for you,
so it's quicker to get started
and on the right path."

EXPO MODES

OBJECT COMPUTING

- "managed"
  - no need to install Android Studio or Xcode
  - after code changes, Expo rebuilds app, hosts in local server, and deploys to simulators/devices
  - some native APIs are not supported
  - **`expo upload`** deploys to stores
  - **`expo publish`** deploys an update to stores
- "bare"
  - can use all **native APIs** and include native code (Java, Kotlin, Swift)
  - you handle steps to build, upload, and publish using Android Studio and Xcode

devices must have "Expo Client" installed from app store

examples include Bluetooth, in-app purchases, and WebRTC

for details on how it works, see https://docs.expo.io/versions/v32.0.0/workflow/how-expo-works/

EXPO MANAGED MODE (does not apply to "bare" mode)

OBJECT COMPUTING

- Pros
  - testing on real devices using Expo Client
  - creates binaries without interacting with Android Studio or Xcode
    - in "managed" mode, not "bare" mode
  - supports over-the-air updates to apps in stores
- Cons
  - no support for native modules
    - must "eject" to use native code
  - no background code execution
  - results in larger apps
    - ~15MB for Android
    - ~20MB for iOS

but these are close to average size of mobile apps

EXPO ADDED COMPONENTS AND APIS

OBJECT COMPUTING

can use outside Expo, but need extra setup

- Components
  - **`Camera`**, **`MapView`**, **`Svg`**, **`Video`**, and more
- APIs
  - **`AppAuth`**, **`Audio`**, **`Calendar`**, **`Contacts`**, **`DeviceMotion`**, **`FaceDetector`**, **`Font`**, **`Haptics`**, **`ImagePicker`**, **`LocalAuthentication`**, **`Localization`**, **`Location`** (geolocation), **`MailComposer`**, **`MediaLibrary`**, **`Notifications`**, **`Payments`**, **`Permissions`**, **`Print`**, **`SecureStore`**, **`Sensors`**, **`SMS`**, **`Speech`**, **`SQLite`**, **`Updates`**, and more

see "API Reference" in left nav. at https://docs.expo.io/versions/latest/

Which of these support push notifications?

SETUP

- There are many steps required to get started

- Step 1: install Node.js from https://nodejs.org/

OPTION #1 - React Native CLI ... skip ahead to option #2

- Browse
  https://facebook.github.io/react-native/docs/getting-started
- Click "React Native CLI Quickstart"
- Select development OS and target OS
- Follow instructions

- `npm install -g react-native-cli`
- `react-native init MyProject`
- `cd MyProject`
- `react-native start`

... OPTION #1 - React Native CLI

- To run on iOS simulator
  - open a new terminal window
  - **`react-native run-ios`**
  - takes about 5 minutes the first time!
  - to reload app press cmd-r

- To run on Android simulator
  - start simulator as shown on slide 20
  - create file
    ***`project-directory`*`/android/local.properties`**
    - add following line

    `sdk.dir=/Users/USERNAME/Library/Android/sdk`
  - **`react-native run-android`**
  - to reload app press “r” twice

OPTION #2 - Expo CLI

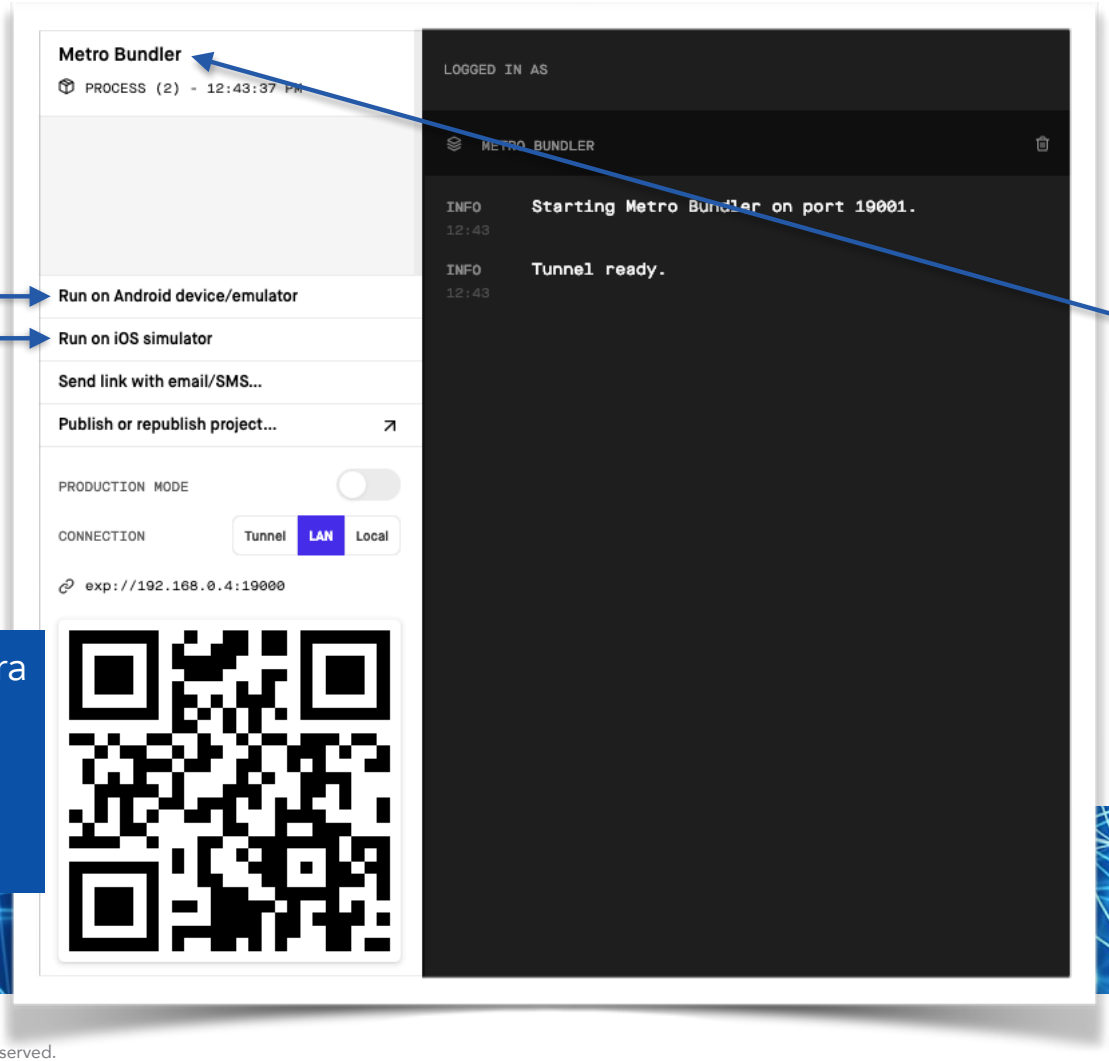
our focus

- **`npm install -g expo-cli`**
- **`expo init my-project`**
  - choose a template, “blank”, “tabs”, or “bare-minimum”
  - enter app name as it should display on devices
  - enter “Y” to install dependencies including react-native and much more
- **`cd my-project`**
- **`npm start`** or **`yarn start`** or **`expo start`**
  - opens a browser tab shown on next slide
  - can start a simulator by pressing “a” for Android or “i” for iOS

Expo projects are tied to specific versions of React and React Native. As of 4/9/2019, cannot use hooks.

expo v32.0.0 requires restart of this and simulators if a JS syntax error is saved!

Expo App Page



click these to run on a simulator that is already running

scan with a device camera to run app on device in "Expo Client"
(may need to switch connection to "Tunnel")

"Metro Bundler" is a JavaScript bundler for React Native that supports watch and live reload in Android and iOS simulators.

OBJECT COMPUTING

OS-SPECIFIC REQUIREMENTS

- Developing for Android in Windows requires Python 2
  - can't use Python 3?
- Does "Xcode for Windows" allow developing iOS apps in Windows?

SIMULATORS

- Can use Android and iOS simulators to test apps
- iOS simulator requires macOS
  - true?
  - Does “Xcode for Windows” support the iOS simulator?

INSTALL ANDROID SIMULATOR

- Browse https://developer.android.com/studio
- Press “DOWNLOAD ANDROID STUDIO”
  - based on Intellij
- Double-click downloaded installer and follow instructions

RUN ANDROID SIMULATOR

- Launch Android Studio app and follow one-time setup instructions
- Select Tools ... AVD Manager stands for "Android Virtual Device"
- Click "+ Create Virtual Device..." to add a new device simulator
- Click "Download" after an existing device simulator
- Click green triangle "play" button after a device simulator
  - takes a couple of minutes the first time
- Can quit Android Studio

INSTALL AND RUN IOS SIMULATOR

- Install Xcode
  - browse https://developer.apple.com/xcode/ and click Download button
- To run simulator
  - launch Xcode app
  - select Xcode ... Open Developer Tool ... Simulator
  - select device by selecting Hardware ... Device ... *os* ... *device-name*
  - can quit Xcode

Alternative to launching Xcode:

```
npm install -g ios-sim
ios-sim showdevicetypes
ios-sim start --devicetypeid "full-device-type"
```

Handy npm script:

```
"ios-sim": "ios-sim start --devicetypeid 'iPhone-XR, 12.1'",
```

RUN APP ON SIMULATOR

OBJECT COMPUTING

- Start one or both simulators
- In Expo web app, press
  "Run on Android device/emulator" or "Run on iOS simulator"
  - takes a couple of minutes the first time
  - installs Expo Client in simulator

INSTALL EXPO CLIENT ON DEVICES

- Install “Expo Client” app on mobile devices
- Launch “Expo Client”
- Press “Sign up for Expo”
- Enter requested info. and press “Sign Up”
  - will receive an authentication email

RUN APP ON ANDROID DEVICE USING EXPO CLIENT

- TRY THIS AND DOCUMENT!
- SHOULD BE SIMILAR TO NEXT SLIDE

RUN APP ON iOS DEVICE USING EXPO CLIENT

- Open Camera app on device
- Point at QR code in Expo web app
- Press "Open in Expo"
  - will launch Expo Client app if not already running

When "CONNECTION" is "LAN", computer and phone must be on same Wifi network.
May work better when "CONNECTION" is "Tunnel".

PROJECT STRUCTURE

- **`index.js`** - app entry point
- **`App.js`** - top component
- **`package.json`** - describes dependencies and scripts
- **`node_modules`** directory - holds libraries installed by npm or yarn
- **`android`** directory - holds Android-specific code
- **`ios`** directory - holds iOS-specific code
- **`.flowconfig`** - configures the Flow type checker
- **`.gitignore`** - lists directories and files that should not be saved in the Git repository
- **`.watchmanconfig`** - configures the Facebook Watchman file watcher that supports hot reloading

```
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

often don't need to edit these files

INITIAL FILES TO EDIT

- **`package.json`** - dependencies and scripts
- **`app.json`** - app configuration
- **`app.js`** - top-most component

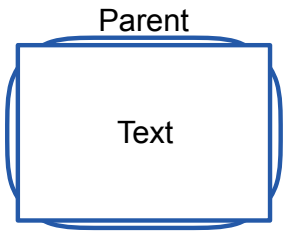
REACT NATIVE DOES NOT USE HTML

- **`div` -> `View`** main building block component
- **`span` -> `Text`**
- **`button` -> `Button`**
- **`img` -> `Image`**
- and many more differences

NOTES ON TEXT

OBJECT COMPUTING

- **`Text`** component
  - inherits **`backgroundColor`** style prop of parent
  - **`color`** style property defaults to **`'black'`**
  - can span multiple lines using **`{'\n'}`**
  - can spill outside parent if parent has non-zero **`borderRadius`** and **`Text`** has no **`margin`** or its **`backgroundColor`** is not **`transparent`**



NOTES ON IMAGE

OBJECT COMPUTING

- Example

```
<Image
  style={styles.logo}
  source={require('../assets/something.png')}
/>
```

- Supports many more props
  - including **`defaultSource`**, **`loadingIndicatorSource`**, and **`resizeMode`** (values are **`center`**, **`contain`** (default), **`cover`**, **`repeat`**, and **`stretch`**)

DEBUGGING ...

- To open debug page
  - on device, shake
  - in Android simulator press ctrl-m (cmd-m)
  - in iOS simulator press ctrl-d (cmd-d)
- `console.log` output
  - goes to terminal where development server is running
  - to see in Chrome devtools Console, open debug page, click “Debug Remote JS” to get new browser tab, and press ctrl-shift-i (cmd-option-i)

... DEBUGGING

- To inspect element hierarchy, size, and CSS properties
  - open debug page, tap “Toggle Element Inspector”, and click an element
  - can’t change input element values when this is open

PERFORMANCE

- To evaluate performance
  - open debug page and tap “Show Perf Monitor”
  - displays a white rectangle in upper-left that shows
    - process memory usage (RAM)
    - JavaScript thread memory usage (JSC)
    - # of **`View`** elements currently visible and saved in memory
    - main thread frame rate (UI)
    - JavaScript thread frame rate (JS)

WATCH AND LIVE RELOAD

- Changes to files in app can trigger reload in all simulators and in Expo Client on devices if enabled
- Live Reload vs. Hot Reloading
  - can enable only one
  - live reload loses state
  - hot reloading retains state
- To enable, open developer menu and tap either "Enable Live Reload" or "Enable Hot Reloading"

REACT-DEVTOOLS

- To install
  - `npm install -g react-devtools`
- To start
  - run app in a simulator
  - enter **`react-devtools`**
  - opens an Electron app

- To use
  - enter name of a React component in search box
  - expand to see child components
  - click a component to see its props, state, and style
  - for more see https://www.npmjs.com/package/react-devtools

can change CSS and some props and state

When the Element Inspector is open, clicking an element in the simulator causes react-devtools to highlight the element and show its props, state, and style.

PROVIDED CROSS-PLATFORM COMPONENTS

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

- Containers
  - **`View`**, **`ScrollView`**, **`FlatList`**, **`SectionList`**, **`VirtualizedList`**
- Output
  - **`Text`**, **`Image`**, **`ImageBackground`**
- Input
  - **`Button`**, **`Picker`**, **`Slider`**, **`Switch`**, **`TextInput`**, **`TouchableHighlight`**, **`TouchableOpacity`**

`View` and **`ScrollView`** automatically use flexbox for layout where **`flexDirection`** defaults to "**`column`**".

`Image` cannot have children. **`ImageBackground`** can!

`Picker` is like an HTML **`select`**.
`Slider` is like an HTML **`input`** with type **`range`**.
`Switch` is like an HTML **`input`** with type **`checkbox`**.
`TextInput` is like an HTML **`input`** with type **`text`**.

FlatList

- For efficient rendering of long lists of components
- Can be vertical (default) or horizontal
- Can have multiple columns
- How does this differ from **`ScrollView`**?
- Props are Add more from section 12.4 in book?
  - **`data`** - an array of objects that hold data for each list item
  - **`keyExtractor`** - function that takes an object and returns a key
  - **`renderItem`** - function that takes an object and its index and returns JSX

KINDS OF BUTTONS

OBJECT COMPUTING

- **`Button`** "supports a minimal level of customization"
  - no **`style`** prop is supported and only **`color`** can be set
  - most apps do not use this
- **`Touchable*`** components display feedback when tapped
  - content is specified with a single child element like **`Text`** or **`Image`**
  - to style add **`style`** prop to child

    such as **`backgroundColor`**, **`underlayColor`** (for **`TouchableOpacity`**), **`color`**, **`padding`**, and **`borderRadius`**
  - **`TouchableHighlight`** darkens background when pressed
  - **`TouchableOpacity`** reduces opacity when pressed so background is partially displayed

MORE PROVIDED CROSS-PLATFORM COMPONENTS

- **`ActivityIndicator`**
  - loading indicator
- **`KeyboardAvoidingView`**
  - moves out of way of virtual keyboard
- **`Modal`** dialog

Add a slide with code for your MyModal component

- **`RefreshControl`**
  - provides "pull to refresh" inside **`ListView`** or **`ScrollView`**
- **`StatusBar`**
  - controls app status bar
- **`WebView`**

ANDROID-SPECIFIC COMPONENTS

- `BackHandler`
- `DatePickerAndroid`
- `DrawerLayoutAndroid`
- `PermissionsAndroid`
- `ProgressBarAndroid`
- `TimePickerAndroid`
- `ToastAndroid`
- `ToolbarAndroid`
- `TouchableNativeFeedback`
- `ViewPagerAndroid`

iOS-SPECIFIC COMPONENTS

- `DatePickerIOS`
- `ImagePickerIOS`
- `InputAccessoryView`
- `MaskedViewIOS`
- `NavigatorIOS`
- `PickerIOS`
- `ProgressViewIOS`
- `PushNotificationIOS`
- `SafeAreaView`
- `SegmentedControlIOS`
- `SnapshotViewIOS`
- `TabBarIOS`

PROVIDED LIBRARIES

- **`AccessiblityInfo`**
- **`Alert`** dialog
- **`Animated`** to create animations
- **`AppState`** foreground vs. background
- **`AsyncStorage`** alternative to browser **`localStorage`**
- **`CameraRoll`** save to photo library and get specified photos
- **`Clipboard`** get and set clipboard string
- **`Dimensions`** get/set screen dimensions and listen for changes
- **`Easing`** animations
- **`Geolocation`**
- **`ImageEditor`** to crop images
- **`ImageStore`**
- **`InteractionManager`** for smooth animations
- **`Keyboard`**
- **`LayoutAnimation`**
- **`ListViewDataSource`**
- **`NetInfo`** online status
- **`PanResponder`** handles multi-touches
- **`PixelRatio`** get pixel density and font scaling factor
- **`Share`** open dialog to share text
- **`Stylesheet`** validates CSS properties and values
- **`Systrace`** debugging and testing
- **`Vibration`**

ANDROID-SPECIFIC PROVIDED LIBRARIES

- `BackHandler`
- `DatePickerAndroid`
- `TimePickerAndroid`
- `ToastAndroid`

iOS-SPECIFIC PROVIDED LIBRARIES

- **`ActionSheetIOS`**
- **`AlertIOS`**
- **`ImagePickerIOS`**
- **`Settings`**
- **`StatusBarIOS`**

PROVIDED PROPERTY SETS - listed under “APIs”

- Image Style - ex. **`opacity`**, **`overflow`**
- Layout - ex. **`flex`**, **`justifyContent`**, **`alignItems`**
- Shadow - ex. **`shadowColor`**, **`shadowRadius`**
- Text Style - ex. **`color`**, **`fontSize`**, **`fontWeight`**
- View Style - ex. **`borderColor`**, **`borderWidth`**

REFRESHER ON REACT PROPS AND STATE

props	state
passed from parent	created in component
immutable	mutable by component
parent can pass different values	component methods can change, but these can be passed to children

react-native-async-storage

- Replacement for deprecated **`Async-storage`** provided by React Native
  - has same API
- Persists data similar to browser **`localStorage`**
- Keys and values must be strings
  - but can use **`JSON.stringify`** to store objects and array

```
import AsyncStorage from '@react-native-community/async-storage';

try {
  await AsyncStorage.setItem(key, value, optionalCallback);
} catch (e) {
  // handle error
}

// To do this after a call to setState updates state ...
this.setState(newState, () => {
  // code above goes here
});

try {
  const value = await AsyncStorage.getItem(key, optionalCallback);
  // use value
} catch (e) {
  // handle error
}
```

MORE `react-native-async-storage` METHODS

- `AsyncStorage.mergeItem(key, value, callback)`
- `AsyncStorage.removeItem(key, callback)`
- `AsyncStorage.clear(callback)`
- `AsyncStorage.getAllKeys(callback)`
- `AsyncStorage.flushGetRequests()`
- `AsyncStorage.multiGet(keyArray, callback)`
- `AsyncStorage.multiSet(pairArray, callback)`
- `AsyncStorage.multiMerge(pairArray, callback)`
- `AsyncStorage.multiRemove(keyArray, callback)`

use when value is stringified JSON

All these methods return a **`Promise`**, so can use **`await`**.
Optional callback functions are passed an error if any and results if any.

SPLASH SCREEN

- Displayed during app startup
- To customize, replace **`assets/splash.png`** with new file that has same aspect ratio

NAVIGATION

- Like routing in a React app
  - one difference is that React Native pages do not have URLs
- Three main kinds
  - tab-based - at top or bottom; create a component for each tab
  - stack-based - go forward and backward by pushing pages onto and popping pages off of a stack
  - drawer-based - side menu of options like a hamburger menu
- Can combine approaches within an app
  - ex. top navigation can be tab-based and some tabs can use stack-based for sub-pages

top app component can be a navigation component

NAVIGATION OPTIONS

- None provided by React Native
- **React Navigation**
  - cross-platform library at https://facebook.github.io/react-native/docs/navigation#react-navigation
  - only supports React Native, not React
  - easy to use
- **React Native Navigation**
  - cross-platform library at https://github.com/wix/react-native-navigation
  - not as easy, but provides native look and feel

REACT NAVIGATION ...

TRY THIS!

OBJECT COMPUTING

- **`npm install react-navigation`**
- Example

```
import {createBottomTabNavigator, createStackNavigator} from 'react-navigation';
const options = {
  navigationOptions: {
    headerStyle: {
      backgroundColor: someColor
    },
    headerTintColor: someColor
  }
};
const StackNav = createStackNavigator({
  Name1: {screen: Component1},
  Name2: {screen: Component2}
}, options);
const Tabs = createBottomTabNavigator({
  Tab1Text: {screen: StackNav},
  Tab2Text: {screen: SomeComponent}
});
export default Tabs;
```

import **`Tabs`** in another component and render it

```
<Tabs screenProps={someObj} />
```

`someObj` is passed to all screens; can include data and callback functions

```
import {createDrawerNavigator} from 'react-navigation';
const DrawerNav = createDrawerNavigation(screens);
```

... REACT NAVIGATION

OBJECT COMPUTING

- To navigate to a different screen in a navigation
  - **`this.props.navigation.navigate(screenName);`**
- Every screen component
  - is passed the props **`screenProps`** and **`navigation`** verify use of screenProps!
    - **`navigation`** object has **`state.params prop`** and **`navigate`** method verify
  - can define a static **`navigationOptions`** property
    - can be an object or a function that takes the props and returns an object
    - configures several things like **`title`** and **`headerTitleStyle`** (object containing CSS properties)

What animation options are available for screen transitions?

ANIMATED API ...

- Provided by React Native
- Four steps to use
  - import
  - create an animatable value
  - use a style to associate the animatable value with a component
  - write and call a function to trigger

... ANIMATED API ...

OBJECT COMPUTING

- Animatable builtin components include
  - **`View`**, **`ScrollView`**, **`Text`**, and **`Image`**
- Can create custom components that are animatable
  - ex. to animate marginTop like a CSS transition

```
import {Animated} from 'react-native';
const animate = () => {
  Animated.timing(
    new Animated.Value(30); // starting value
    {duration: 1000, toValue: 300} // ending value
  ).start();
};
```

could use current value

could make this an argument to **`animate`** function

```
// In some component ...
  render() {
    return (
      <View style={...}>
        ...
        <Button title="..." onPress={animate} />
        ...
      </View>
    );
  }
```

Are there any CSS properties that cannot be animated?

... ANIMATED API ...

- Can animate expanding a **`TextInput`** when it gains focus
  - using its **`onFocus`** and **`onBlur`** event handling props
- Can create continuous animations with **`Animated.loop`** function
  - one use is activity indicators
  - resets at end of each animation
  - can cancel

... ANIMATED API ...

OBJECT COMPUTING

- What does interpolation do?
  - `Animated.interpolate({inputRange: originalValues, outputRange: goalValue});`
- Easing functions
  - control progression of changes from input to output values

```
const animatedValue = new Animated.Value(value);
const interpolatedValue = animatedValue.interpolate({
  inputRange: [0, 1],
  outputRange: ['0deg', '360deg']
});

// Show this only when loading some resource.
<AnimatedImage
  source={____}
  style={{
    transform: [{rotate: interpolatedValue}]
  }}
/>
```

... ANIMATED API ...

- To run animations of multiple components in parallel
  - **`Animated.parallel(values)`** returns object with **`start`** method that must called to start animations
  - values is an array of values returned by **`Animated.timing`** calls
- To run multiple animations in series
  - **`Animated.sequence(values)`** returns object with **`start`** method that must called to start animations
- To run multiple animations whose start times are staggered
  - **`Animated.stagger(values)`** returns object with **`start`** method that must called to start animations

specifies start of each by some # of milliseconds more than previous

... ANIMATED API ...

- To reset an animated value so it can be used again
  - `animatedValue.setValue(someValue)`
- To run code when animations complete
  - pass a function to `start` method

... ANIMATED API ...

- Animations are performed in JavaScript thread by default
  - What does this mean since the app is compiled to a native app?
  - Where is the JS?
- Some animations can run in a native UI thread
  - perhaps only animations of non-layout properties
  - to do this, add `useNativeDriver: true` to object passed to `Animated.timing`

... ANIMATED API

- To create a custom animatable component
  - `const AnimatableName = Animated.createAnimatableComponent(ComponentName);`
  - for example, the component could be **`TouchableHighlight`**
  - **TRY THIS!**

WEB VIEW

- Cross-platform rendering of HTML
- https://github.com/react-native-community/react-native-webview
- Replacement for provided **`WebView`** which will be removed
- Not supported by Expo unless ejected

THIRD-PARTY COMPONENTS

- NativeBase at https://nativebase.io
- Current list of components
  - **`Anatomy`**, **`Accordion`**, **`ActionSheet`**, **`Badge`**, **`Button`**, **`Card`**, **`CheckBox`**, **`DatePicker`**, **`DeckSwiper`**, **`Fab`** (Floating Action Button), **`Footer`**, **`FooterTab`**, **`Form`**, **`Header`**, **`Icon`**, **`Layout`**, **`List`**, **`Picker`**, **`Radio`**, search bar (special kind of **`Header`**), **`Segment`**, **`Spinner`**, swipeable list (special kind of **`List`**), **`SwipeRow`**, **`Tab`**, **`Tabs`**, **`Thumbnail`**, **`Toast`**, **`Typography`**, **`Drawer`**, **`Ref`**

STYLING ...

- Supports a subset of CSS
  - properties supported vary by component
  - some are specific to Android or iOS
- Units (ex. px) are not specified
  - and depend on element and property
- **`Stylesheet.create`**
  - validates CSS properties and values

```
const styles = StyleSheet.create({
  container: {
    backgroundColor: 'cornflowerblue',
    flex: 1, // fills screen vertically
    justifyContent: 'flex-start',
    alignItems: 'center'
 },
  error: {
    color: 'red',
    fontSize: 36,
    fontWeight: 'bold'
  },
});
```

definition of **`styles`** object typically appears after component definition

typical to use **`styles.container`** for top element in each component

```
<ScrollView contentContainerStyle={styles.container}>

<Text style={styles.error}>
  Authentication failed!
</Text>
```

... STYLING

- Inline styling
  - `style={{prop1: value1, prop2: value2, ...}}`
- Combining multiple style objects
  - `style={[styleObject1, styleObject2, ...]}`
  - when multiple style objects define same property, last one in wins
  - can also combine outside **`style`** prop

```
const newStyles = StyleSheet.flatten([styleObject1, styleObject2, ...]);
```

GOTCHA!
The prop name is "**`style`**", not "**`styles`**".
If you use "**`styles`**" it will be silently ignored.

test styling in both Android and iOS to verify consistent support

STYLING IMPORTS

- Components can import style objects
- Allows them to be shared between components

FLEXBOX ...

- Layout for all components with children is done with flexbox
  - supported by Yoga library (https://yogalayout.com/)
  - turned on by default
  - some differences from web usage
  - **`flex-direction`** default is **`column`** instead of **`row`**
  - **`flex-grow`** is just **`flex`**

FLEXBOX ...

- Review of properties that are the same as in CSS
  - **`justifyContent`** - **`flex-start`** (default), **`center`**, **`flex-end`**, **`space-around`**, **`space-between`**
  - **`alignItems`** - **`stretch`** (default), **`center`**, **`flex-start`**, **`flex-end`** applies to children
  - **`alignSelf`** - **`auto`** (default; uses **`alignItems`** value) and all **`alignItems`** values applies only to current element
  - **`flexWrap`** - **`nowrap`** (default; elements that don't fit can be clipped) and **`wrap`**

POSITION

- Only supported values for CSS **`position`** property are **`relative`** (default) and **`absolute`**
- Typically use flexbox and default **`position`** to position components rather than **`position: 'absolute'`**

BORDERS

- Defaults
  - `borderColor` -> `'black'`
  - `borderStyle` -> `'solid'`
    - as of 4/7/19, other values are currently rendered as solid on both Android and iOS
  - `borderWidth` -> `0`
- So can get solid black border by just setting `borderWidth`

MARGIN AND PADDING

- No shortcut properties like in CSS
  - ex. cannot use **`margin: '10 20 30 40'`**
  - but React Native adds the style properties **`marginHorizontal`**, **`marginVertical`**, **`paddingHorizontal`**, and **`paddingVertical`**

FONT STYLE PROPERTIES

OBJECT COMPUTING

- **`fontFamily`**
  - only accepts a single font name, not multiple like in CSS
  - can specify a different **`fontFamily`** for each platform using **`Platform.OS`** (and a ternary) or **`Platform.select`**
- **`fontSize`**
  - defaults to **`14`**
- **`fontStyle`**
  - values are **`'normal'`** (default) and **`'italic'`**
- **`fontWeight`**
  - values are **`'normal'`** (same as **`400`**; default), **`'bold'`** (same as **`700`**), and strings containing the numbers **`100`** to **`900`** in increments of **`100`** (ex. **`'900'`**)

TEXT STYLE PROPERTIES ...

- **`textAlign`** What does `'auto'` do?
  - values are `'auto'` (default), `'center'`, `'left'`, `'right'`, and `'justify'` (only iOS)
- **`textDecorationLine`**
  - values are `'none'` (default), `'underline'`, `'line-through'`, and `'underline line-through'`
- **`textDecorationColor`** - a color
- **`textDecorationStyle`**
  - values are `'solid'` (default), `'dashed'`, `'dotted'`, and `'double'`

... TEXT STYLE PROPERTIES

OBJECT COMPUTING

- **`textShadowColor`** - a color
- **`textShadowOffset`**
  - value is an object with **`height`** and **`width`** properties
  - can be negative for shadows on top and left
- **`textShadowRadius`** - a number

To add a drop shadow to a component,
use Elevation styles on Android (not a great effect)
or ShadowPropTypesIOS styles on iOS (ignored on Android).

TRANSFORMS

OBJECT COMPUTING

- **`transform`** style property
  - value is an array of objects applied sequentially with no delay between
  - ex. **`transform: [{rotate: '90deg'}, {scale: 0.7}]`** one operation per object
- Supported operations
  - **`rotate`** (same as **`rotateZ`**), **`rotateX`**, **`rotateY`**, **`rotateZ`** values can be in degrees or radians; origin is original center of component
  - **`scale`**, **`scaleX`**, **`scaleY`** one use is to render thumbnails
  - **`translateX`**, **`translateY`** x values increase to right; y values increase going down; components can be translated out of view
  - **`skewX`**, **`skewY`**
  - **`perspective`** for 3D effects

`backfaceVisibility` style property can be used to create a "card effect" and has values **`'visible'`** and **`'hidden'`**.

PROVIDED FONTS

- Listed at https://github.com/react-native-training/react-native-fonts
- Android provides far fewer than iOS
- Android supports some generic font names, but iOS does not
  - `monospace`, `sans-serif`, `serif`

CUSTOM FONTS

- To use
  - create **`assets/fonts`** directory if it doesn't exist
  - download font files into this directory
  - follow steps on next two slides based on project type

IMPORTANT
Open font files and verify that the file names matches the font name exactly. In macOS, double-clicking a font file opens it in the **Font Book** app and displays its name in the title bar.
If the name differs, rename the file.

CUSTOM FONTS IN EXPO PROJECTS

- Do this in **`App.js`**
- Specify custom fonts in **`fontFamily`** style properties
- Fonts loaded this way do not currently support **`fontStyle`** or **`fontWeight`** properties, so load font variations separately

```
import {Font} from 'expo';
import React, {Component} from 'react';

export default class App extends Component {
  state = {fontLoaded: false};

  async componentDidMount() {
    await Font.loadAsync({
      'font name': require('./assets/fonts/font-file')
    });
    this.setState({fontLoaded: true});
  }

  render() {
    return this.state.fontLoaded ? <MyTopComponent /> : null;
  }
}
```

can't render components that use a font until it has been loaded

CUSTOM FONTS IN REACT NATIVE CLI PROJECTS

- Add to **`package.json`**

```
"rnpm": {
  "assets": ["./assets/fonts/"]
},
```

- Run **`react-native link`**
- Specify custom fonts in **`fontFamily`** style properties

DYNAMIC STYLES

- Since styles are defined in JavaScript code they can be dynamic
  - ex. can define light and dark theme style objects and allow users to switch between their use

STYLE FILES

- Styles can be defined in separate file from component definition
  - often named **`styles.js`** inside component-specific directory
  - export styles object that is imported where needed
  - allows sharing styles between components

most prefer to only put styles in a separate file when they are shared by multiple components

PLATFORM DIFFERENCES

- Guidelines differ between Android and iOS
- Consider these to provide the most native experience
- Resources
  - https://medium.muz.li/differences-between-designing-native-ios-apps-and-native-android-apps-e71256dfa1ca
  - https://www.ready4s.com/blog/android-vs-ios-comparing-ui-design
  - https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7
  - https://medium.com/@chunchuanlin/android-vs-ios-compare-20-ui-components-patterns-part-1-ad33c2418b45

PLATFORM-SPECIFIC CODE

OBJECT COMPUTING

- Use **`Platform`** module to take platform-specific actions
- **`Platform.OS`**
  - holds string **`'android'`** or **`'ios'`**
- **`Platform.select(obj)`**
  - ***`obj`*** has properties **`android`** and **`ios`**
  - returns value of property that matches current platform
  - often used in CSS property lists
    - values are objects that are used with spread operator

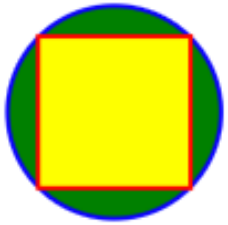
```
const company =
  Platform.OS === 'android' ?
  'Google' : 'Apple';
```

```
const styles = StyleSheet.create({
  container: {
    color: 'white',
    ...Platform.select({
      android: {
        backgroundColor: ''
      },
      ios: {
        backgroundColor: ''
      }
    })
  }
});
```

SVG ...

- Can use react-native-svg on npm
  - https://github.com/react-native-community/react-native-svg#common-props
- Installing
  - when using Expo, skip this because it is already installed
  - **`npm install react-native-svg`**
  - **`react-native link react-native-svg`**
- Example
  - see next slide

... SVG



```
import {Svg} from 'expo';
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';

const {Circle, Rect} = Svg;

const SIZE = 100;
const HALF_SIZE = SIZE / 2;
const OFFSET = SIZE * 0.15;
const SIZE7 = SIZE * 0.7;

const SvgDemo = () => (
  <View style={styles.container}>
    <Svg
      height="100%"
      width="100%"
      viewBox={`0 0 ${SIZE} ${SIZE}`}>
      <Circle
        cx={HALF_SIZE}
        cy={HALF_SIZE}
        r={HALF_SIZE - 1}
        stroke="blue"
        strokeWidth={2}
        fill="green"
      />
      <Rect
        x={OFFSET}
        y={OFFSET}
        width={SIZE7}
        height={SIZE7}
        stroke="red"
        strokeWidth={2}
        fill="yellow"
      />
    </Svg>
  </View>
);

const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    justifyContent: 'center',
    height: SIZE,
    width: SIZE
  }
});

export default SvgDemo;
```

STATUS BAR

- Modify native status bar by including **`StatusBar`** element
- Useful props
  - **`barStyle`** - set to “light-content”, “dark-content”, or “default”
  - **`hidden`** - add with no value to hide entire status bar
  - some props only work on Android

CAMERA

- Doesn't work in simulators, but does in Expo Client on mobile devices

BIOMETRICS

- Can use touch id or face id to authenticate
- Expo API **`LocalAuthentication`** supports this
- Do not need to eject to use
- iOS Face ID
  - must describe why in **`app.json`**
  - not supported in Expo Client, but works in standalone apps created by Expo

```
{
  ...
  "infoPlist": {
    "NSFaceIDUsageDescription": "access Running Calculator"
  },
  ...
}
```

ALERT

- To display an alert dialog

```
Alert.alert(title, message, buttons, options);
```

  - ***buttons*** is an array of objects with properties **`text`**, **`style`**, and **`onPress`**
  - ***options*** is an object that can contain **`cancelable: true`**

APPLICATION STATE

OBJECT COMPUTING

- States are **`active`**, **`inactive`**, and **`background`**

```
AppState.addEventListener('change', newAppState => { ... });
```

- Can use to repeat authentication or refetch data whenever app becomes active again
- When performing polling, could use to disable when app state becomes non-active and resume when active

CLIPBOARD

- To change and retrieve system clipboard contents

```
import {Clipboard} from 'react-native';
Clipboard.setString(value);
const value = await Clipboard.getString();
```

  - Does setString return a Promise?
  - Are there any other methods?

SCREEN DIMENSIONS

OBJECT COMPUTING

- To get screen dimensions

```
import {Dimensions} from 'react-native';
const {height, width} = Dimensions.get("window");
```

  - Can any other values be passed to the **`get`** method?

GEOLOCATION ...

- Enabled by default in iOS
- Enable for Android by adding line to **`AndroidManifest.xml`**

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- Can use with react-native-maps to display location

```
navigator.geolocation.getCurrentPosition(successCb, errorCb);
```

  - **`successCb`** is passed an object with a coords property whose value is an object with properties **`accuracy`**, **`altitude`**, **`altitudeAccuracy`**, **`heading`**, **`latitude`**, **`longitude`**, and **`speed`**

... GEOLOCATION

- **`navigator.geolocation.watchPosition`**
  - similar to **`getCurrentPosition`**,
    but calls **`successCb`** again every time device location changes
  - returns a watch id used to cancel it
- **`navigator.geolocation.clearWatch(`*`watchId`*`)`**
  - cancels a specific **`watchPosition`**
- **`navigator.geolocation.stopObserving()`**
  - cancels all **`watchPositions`**

KEYBOARD API

- Used to access native keyboard when behavior of **`TextInput`** isn't enough
- Can control when device keyboard is displayed and hidden

```
import {Keyboard} from 'react-native';
Keyboard.addListener(eventName, callback);
Keyboard.removeListener(eventName);
Keyboard.dismiss();
```

- Events are
  - **`keyboardWillShow`**, **`keyboardDidShow`**, **`keyboardWillHide`**, **`keyboardDidHide`**, **`keyboardWillChangeFrameListener`**, and **`keyboardDidChangeFrameListener`**

What does this mean?

NETINFO API ...

- Determines if online or offline
- Enabled by default in iOS
- Enable for Android by adding line to **`AndroidManifest.xml`**

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- Uses
  - show an offline indicator
  - show cached data when offline
  - save inputs when offline and process when online again

... NETINFO API

OBJECT COMPUTING

- On iOS values are none, Wifi, cell, and unknown
- On Android there are many more values
  - **NONE**, **BLUETOOTH**, **DUMMY**, **ETHERNET**, **MOBILE**, **MOBILE_*kind***, **VPN**, **WIFI**, **WIMAX**, and **UNKNOWN**
  - where kind is **DUN**, **HIPRI**, **MMS**, or **SUPL**

Is this a valid alternative?

```
const netType = await NetInfo.fetch();
```

```
import {NetInfo} from 'react-native';
NetInfo.fetch().done(callback);
// callback is passed one of the values above
const expensive = await NetInfo.isConnectionExpensive();
const connected = await NetInfo.isConnected();
const listener = connection => { ... };
NetInfo.addEventListener('change', listener);
NetInfo.removeEventListener('change', listener);
```

PANRESPONDER API ...

OBJECT COMPUTING

- Used to detect single and multiple touch events
- Uses
  - implement swipeable cards
  - allow users to rearrange items

```
import {PanResponder} from 'react-native';
const panResponder = PanResponder.create({
  on___: (event, gestureState) => { ... },
  ...
});
```

can be **`StartShouldSetPanResponder`**, **`MoveShouldSetPanResponder`**, or **`PanResponder`*`Action`*** where ***`Action`*** is **`Reject`**, **`Grant`**, **`Start`**, **`End`**, **`Move`**, **`TerminationRequest`**, **`Release`**, or **`Terminate`**

Try to use these to imeplement a draggable View (see example in book)

... PANRESPONDER API

- Properties in event objects passed to **`on*`** methods
  - **`changedTouches`** - array of objects with these properties
    - **`identifier`**, **`locationX`**, **`locationY`**, **`pageX`**, **`pageY`**, **`target`**, **`timestamp`**, **`touches`**
    - **`target`** is a node id
    - **`touches`** is an array of all touches, not just the changed ones
  - **`gestureState`** - object with these properties
    - **`stateID`**
    - **`moveX`**, **`moveY`** - screen coordinates of touch
    - **`x0`**, **`y0`** - screen coordinates of responder
    - **`dx`**, **`dy`** - since touch started
    - **`vx`**, **`vy`** - current velocity
    - **`numberOfActiveTouches`**

PLATFORM-SPECIFIC CODE ...

OBJECT COMPUTING

- One way to implement platform-specific code is to use platform file extensions
  - **`___.android.js`** and **`____.ios.js`**
  - in other files, import just file name with no extension
  - for example, the **`.android.js`** file could use **`DatePickerAndroid`** and the **`.ios.js`** file could use **`DatePickerIOS`**

... PLATFORM-SPECIFIC CODE

OBJECT COMPUTING

- Another way to implement platform-specific code is to use **`Platform.OS`** or **`Platform.select`**
  - **`Platform.OS`** will be the string **`'android'`** or **`'ios'`**
  - can use **`Platform.select`** with spread operator in style objects

```
Platform.select({
  android() {
    // Android-specific code goes here.
  },
  ios() {
    // iOS-specific code goes here.
  }
});
```

ANDROID-SPECIFIC COMPONENTS

- The next several slides describe some components that can only be used in Android

DrawerLayoutAndroid

- Why not use platform-independent drawer navigation?
- Add more detail or delete this slide

ToolbarAndroid

- Can display a title, subtitle, navigation icon, and action buttons
- Add more detail or delete this slide

ViewPagerAndroid

- To support swiping left and right to change views
- Add more detail or delete this slide

DatePickerAndroid

- Is there a platform-independent component that wraps the use of DatePickerAndroid and DatePickerIOS (maybe in Expo)?
- Can use MomentJS library to work with values
- Add more detail or delete this slide

TimePickerAndroid

- Add more detail or delete this slide

ToastAndroid

- Displays a message popup
  that goes away after specified amount of time
- Can be at bottom or top of screen

```
<ToastAndroid.show(message, duration, position)
```

- **duration** can be **SHORT** (~2 seconds) or **LONG** (~4 seconds)

iOS-SPECIFIC COMPONENTS

- The next several slides describe some components that can only be used in iOS

DatePickerIOS

```
<DatePickerIOS
  date={initialDate}
  mode="mode"
  onDateChange={date => { ... }}
/>
```

`date` is a JavaScript `Date` object

- `mode` values are `'date'`, `'time'`, and `'datetime'` (default)
- Other props are
  - `maximumDate`
  - `minimumDate`
  - `minimumInterval` - interval at which minutes can be selected (defaults to 1)
- Can use MomentJS library to work with values

PickerIOS

- For picking an item from a list
- How is this better than the platform-independent **`Picker`** component?
- If better, add details from book

ProgressViewIOS

- To show percent completion of an activity such as fetching data from a REST service
  - How would you know the percentage, from Content-Length header?
- Other props
  - `progressImage`
  - `trackImage`
  - `progressTintColor`
  - `progressViewStyle`
  - `trackTintColor`

```
<ProgressViewIOS
  progress={percentage}
/>
```

ActionSheetIOS ...

- To show action sheets and share sheets
- Action sheets are specified with an array of buttons that each have an associated function that is called when pressed
  - setting **cancelButtonIndex** to 0 places it at bottom

```
ActionSheetIOS.showActionSheetWithOptions(
  buttonTexts,
  cancelButtonIndex: 0,
  successCb);
```

... ActionSheetIOS

- **showShareSheetWithOptions** options
  - **url**
  - **message**
  - **title**
  - **subject**
  - **excludeActivityTypes** - an array; What are the possible values?
  - **successCb** is passed
    - a boolean to indicate if successful
      (Why doesn't it just call errorCb if not successful?)
    - a method (What is this?)

```
ActionSheetIOS.showShareSheetWithOptions(
  option, errorCb, successCb);
```

SegmentedControlIOS

- For horizontal tab bars
- Doesn't on its own hide and show other components
- Why not use platform-independent navigation tabs?

TabBarIOS

- Add description
- Doesn't on its own hide and show other components

EXPO SNACK ...

- For experimenting with React Native in a browser
- No need to install anything
- Like Codepen and JSFiddle for React Native
- https://blog.expo.io/sketch-a-playground-for-react-native-16b2401f44a2

... EXPO SNACK

- To use
  - browse https://snack.expo.io/ and log in
  - run “Expo Client” app on mobile device
  - click “Projects” and note Device ID
  - enter Device ID in web UI
  - click “QR Code” tab
  - scan with device camera
  - app will appear on mobile device
  - modify app code in browser; no need to save
  - will hot reload on device
- To export
  - press Save button
  - click Export
  - will download zip file containing code

HTTP SUPPORT

- React Native supports the Fetch API
- Doesn't allow http requests; must use https
  - unless configured to allow use of http

SWAPI

- Star Wars REST API good for building demo apps
- https://swapi.co

DEPLOYING TO APP STORES

- Many steps are required
- Summarized at
  https://apiko.com/blog/deploying-react-native-apps-to-app-store-and-play-market/
- For Android need **`.apk`** files (Android Package) can download directly from any website
- For iOS need **`.ipa`** files (iOS App Store Package) must install from iOS App Store
- Developer accounts are not free
  - iOS - need an Apple Developer account which is $99 per year
  - Android - need a ? which is a $25 one-time registration fee

BUILDING SIGNED ANDROID APK

- See https://facebook.github.io/react-native/docs/signed-apk-android
- Steps
  - generate signing key
  - edit **`gradle.properties`**
  - add signing config to gradle config
  - build
    - **`cd android`**
    - **`./gradlew assembleRelease`**
  - test
    - **`react-native run-android --variant=release`**

VISUAL STUDIO APP CENTER ...

- "Continuously build, test, release, and monitor apps for every platform"
- Supports many kinds of apps
  - Android, iOS, React Native, Xamarin, Cordova, Windows, macOS, tvOS
- https://appcenter.ms/
  - "Connect to GitHub, Bitbucket, or Azure DevOps"
  - "build your app in the cloud on every commit"
  - "automatically run unit tests, release to testers and stores, or test your UI on real devices"

may not work with Expo projects!

... VISUAL STUDIO APP CENTER ...

- Free features
  - 240 build minutes per month
  - 30 day free trial for running tests
  - unlimited app distribution
  - analytics
  - crash reporting

APP CENTER SETUP

https://docs.microsoft.com/en-us/appcenter/sdk/getting-started/react-native

- Browse `http://appcenter.ms`
- Click “Add new app”
- Enter name and description
- Select iOS or Android for OS
- Select “React Native” for Platform
- Click “Settings” in left nav.
- Click vertical “...” in upper-right and select “Copy app secret"

- `npm install appcenter appcenter-analytics \ appcenter-crashes --save-exact`
- For iOS, one-time
  `sudo gem install -n /usr/local/bin cocoapods`
- `react-native link`
- Enter the secret when prompted or modify secret in file path that is output
- May need to add “setup” section to package.json
- Commit all code to a GitHub repo

KEYSTORE GENERATION

Is anything on this slide really needed? It seems like commands on the next slide generate this file again.

- `keytool -genkeypair -v -keystore my-release-key.keystore \ -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000`
- Answer a bunch of questions
  - remember the password you entered
- Generates **`my-release-key.keystore`**

ADDING KEYSTORE TO JDK ...

OBJECT COMPUTING

- Determine location by entering **`/usr/libexec/java_home`**
- cd to that directory
- **`sudo keytool -genkey -v -keystore my-release-key.keystore \`**
  `-alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000`
- Copy **`my-release-key.keystore`** to **`android/app`** directory of application
- Edit **`android/gradle.properties`**

```
MYAPP_RELEASE_STORE_FILE=my-release-key.keystore
MYAPP_RELEASE_KEY_ALIAS=my-key-alias
MYAPP_RELEASE_STORE_PASSWORD=*****
MYAPP_RELEASE_KEY_PASSWORD=*****
```

password from previous slide

... ADDING KEYSTORE TO JDK

OBJECT COMPUTING

- Edit **`android/app/build.gradle`**

```
android {
    ...
    signingConfigs {                    new section
        release {
            if (project.hasProperty('MYAPP_RELEASE_STORE_FILE')) {
                storeFile file(MYAPP_RELEASE_STORE_FILE)
                storePassword MYAPP_RELEASE_STORE_PASSWORD
                keyAlias MYAPP_RELEASE_KEY_ALIAS
                keyPassword MYAPP_RELEASE_KEY_PASSWORD
            }
        }
    }
    buildTypes {                        existing section
        release {
            ...
            signingConfig signingConfigs.release        add this
        }
    }
}
```

BUILDING AND TESTING ANDROID .APK

- `cd android`
- `./gradlew assembleRelease` I get an error in this step!
- `react-native run-android --variant=release`

APP CENTER BUILD

- Click “Build” in browser left nav.
- Once for each project
  - select a repository service such as GitHub
  - select a repository
  - click a branch name such as “master”
  - click “Configure build” and choose options
  - for signed builds
    - click wrench icon in upper-right
    - turn on “Sign builds”
- press “Save & Build”
- Subsequent builds
  - click a branch name
  - click “Build now”
    - will see build output in browser

APP CENTER DOWNLOAD/RUN/TEST

- Click “Download” and click “Download build”
  - downloads a **`build.zip`** file
- Unzip to get **`app-release.apk`**
- Install on an Android phone

APPLE DEVELOPER ACCOUNT

- Steps to create
  - browse https://developer.apple.com/enroll/
  - See notes/AppleDeveloperNotes.md

WRAP UP

- Add this!

LEARN MORE ABOUT OCI EVENTS AND TRAINING

Events:

- objectcomputing.com/events

Training:

- objectcomputing.com/training
- grailstraining.com
- micronauttraining.com

Or email info@ocitraining.com to schedule a custom training program for your team online, on site, or in our state-of-the-art, Midwest training lab.

OBJECT COMPUTING

CONNECT WITH US

1+ (314) 579-0066

@objectcomputing

objectcomputing.com