OBJECT
COMPUTING

WEBINAR

React Native Cross-Platform Components

mark@objectcomputing.com

PROVIDED CROSS-PLATFORM COMPONENTS

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

OBJECT COMPUTING

- 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`**.

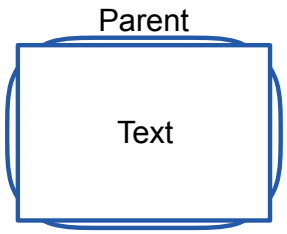
REACT NATIVE ELEMENTS

- Collection of cross-platform React Native components that supplement provided components
- https://react-native-training.github.io/react-native-elements/
- **`Avatar`**, **`Badge`**, **`Button`**, **`ButtonGroup`**, **`Card`**, **`CheckBox`**, **`Divider`**, **`Header`**, **`Icon`**, **`Image`**, **`Input`**, **`ListItem`**, **`Overlay`**, **`Pricing`**, **`Rating`**, **`SearchBar`**, **`Slider`**, **`SocialIcon`**, **`Text`**, **`Tile`**, **`Tooltip`**

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`**)

`ScrollView`

- When height is less that content height, content can be scrolled
- When content includes **`TextInput`** components,
  add **`ScrollView`** prop **`keyboardShouldPersistTaps="handled"`**
- Allows taps on components outside the **`ScrollView`**
  while on-screen keyboard is displayed

`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`**

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`**

PROVIDED PROPERTY SETS - listed under “APIs”

OBJECT COMPUTING

- 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

OBJECT COMPUTING

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

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

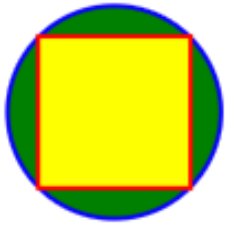
THIRD-PARTY COMPONENTS

- 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`**

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

OBJECT COMPUTING

- 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

OBJECT COMPUTING

- 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 ...

OBJECT COMPUTING

- 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

- 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**

```
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);
```

Is this a valid alternative?

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

PANRESPONDER API ...

- 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`**