GETTING STARTED

Installing EXPO-CLI:

`sudo npm install -g expo-cli`

- Managed workflow - Expo takes care of a lot of the complexity
- Bare workflow, the iOS and Android folders are available to work in, etc.

Creating a project:

`expo init <appName>`
`cd <appFolder>`
`sudo npm install`

Start a server to see development:

`npm start`

- Type `i` for **iOS** simulator (ctl+d, cmd+d -> developer tools)
- Type `a` for **Android** emulator (ctl+m -> developer tools)
- Type `j` for **Chrome developer tools** debugging (sources, pause in upper right, select pause on caught exceptions

Debugging:

- Lots of issues debugging in VSCode due to Mac security issues. Must fix.

Publishing:

- `expo publish`
  - Username: EvanMarie
  - Password: #=?c9S.N!8%}iW'
- App.json has the details about the app. This is where a description can be added / edited.

How React Native Works:

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
 return (
   <View style={styles.container}>
     <Text>Open up App.js to start working on your app!</Text>
     <StatusBar style="auto" />
   </View>
 );
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: '#fff',
   alignItems: 'center',
   justifyContent: 'center',
 },
});
```

- This is a basic React Native component
- These are components of React Native:
  - **View** = similar to div
  - **Text** = used to display text on screen
- React Native uses function components by default rather than class components
- Returns JSX expressions
  - Here, we have a View, like a div
  - It contains styles
  - Within View, we have a Text component for displaying text on screen

- **styles.container**
  - **StyleSheet.create** - property container holds all the styles for that one style component

- These components will be mapped to various components native to the operating systems when compiled

- Other core components include: **Image**, **Button**, **Touchables**, and **Alert**

CORE COMPONENTS and APIs

- ReactNative.dev
- **SafeAreaView** from React Native - insures that your content is placed within the safe areas for viewing on a device. In can be used instead of View.
- **Text** - Text cannot be placed just anywhere as with web apps. It must always be within a Text component.
  - **numberOfLines** - often used with ellipsizeMode, when set, if the text is longer, it will get truncated.

    - **onPress** - set to a function, but only use very simple function inline
    - Read docs for WAY more.
- Image -
    - To get items centered on both axes: `alignItems: 'center'` and `justifyContent: 'center'`,
    - With the Image component, you can render images bundled with the app or images from online
    - **Picsum** - for random images
    - **Dimensions** can either be specified in styles or inline as a part of an object
    - **loadingIndicatorSource** - the image that will be displayed while a network image is being downloaded
    - **blurRadius** - blurs a portion of the image
- **Touchables** - unlike text, images and other components do not have the onPress attribute. Touchables are how we make components interactive in this case. There are 3 different types, and the choice between comes down to what kind of interaction it should have with the user.
    - TouchableWithoutFeedback -
        - onPress
        - onLongPress
    - TouchableOpacity -
        - Changes the opacity of the image for a split second after being tapped
    - TouchableHighlight -
        - Creates an effect not that different from TouchableOpacity
    - TouchableNativeFeedback - Android only - creates a cool effect on divs or images with a dark background.
- Buttons - see code for examples.
- **Platform Module for Native Specifications**:
    - See examples in code
- **Pixels on mobile devices**:
    - Density Independent Pixels (dp or dip) is a unit of measurement used in Android development to specify sizes and positions in a way that is consistent across devices with different pixel densities. It allows for scalable and consistent user interfaces on devices with varying screen densities.
    - Mobile devices come in different screen sizes and resolutions. The pixel density (measured in pixels per inch or PPI) refers to the number of pixels per inch on a device's screen. Devices with

higher pixel densities have more pixels packed into each inch, resulting in sharper and more detailed displays.
    - However, if you were to specify sizes using absolute pixel values (e.g., setting a width to 100 pixels), it would lead to inconsistent sizes on devices with different pixel densities. For example, an element that appears small on a high-density device might appear larger on a low-density device.
    - This is where Density Independent Pixels come into play. dp is a unit of measurement that accounts for the screen density and ensures that sizes are scaled consistently across devices.
    - The conversion formula for converting dp to pixels varies based on the device's screen density:
    - pixels = dp * (density / 160)
    - In this formula, `dp` is the value in Density Independent Pixels, `density` represents the device's screen density (dots per inch or dpi), and `160` is the baseline density of a medium-density screen (160 dpi).
    - When specifying sizes in dp, the Android framework automatically performs the necessary scaling calculations to adapt the sizes to the device's screen density. This helps to maintain consistent visual proportions and layouts across different devices.
    - For example, if you set a width of `100dp` for an element, it will occupy approximately the same physical width on different devices, regardless of their pixel densities. The Android system will handle the conversion from dp to pixels behind the scenes.
    - Using dp allows for better cross-device compatibility and helps ensure that user interfaces appear consistent across various screen densities.
    - It's important to note that in React Native, the equivalent unit to dp is called Density Independent Pixels (dp) in Android and Points (pt) in iOS. However, in React Native, the common term used is `dp` for both platforms.
    - By specifying sizes in dp (or pt in iOS) rather than absolute pixel values, you can create responsive layouts that adapt well to devices with different screen densities.
    - I hope this explanation clarifies the concept of Density Independent Pixels and how they relate to specifying sizes on mobile devices. Let me know if you have any further questions!

- Using Dimensions from React Native, we can get the dimensions of the device:
  ```
  import { Dimensions } from "react-native";
  console.log("Dimensions: ", Dimensions.get("screen"));
  ```

(in terminal) `Dimensions:  {"fontScale": 1, "height": 932, "scale": 3, "width": 430}`

- To allow for both landscape and portrait mode on mobile devices, go to app.json, and set orientation as default.
    - To detect the orientation of a user's mobile device, use [hooks from react-native-community]
        - `sudo npm install @react-native-community/hooks`
        - `import { useDeviceOrientation } from "@react-native-community/hooks"`
            - While react native will not generally get the dimensions once the user turns the mobile device, this hook will update the orientation as it changes.
            - This does not really work anymore, because the library no longer has useDimensions(), which retrieves the dimensions of the screen whenever the device's orientation changed.
            -

FLEXBOX in REACT NATIVE

- With a View Component:
    - flex: 1 -> View will take 100% of the screen
    - flex: 0.5 -> View will take ½ of the screen

EXPO CONSTANTS:

- npm install expo-constants
    - Gives a lot of information about the current platform of a user's device
    - EX. in the iphone 14 simulator, console.logging Constants:
        - Constants {"appOwnership": "expo", "debugMode": false, "deviceName": "iPhone 14 Pro Max", "executionEnvironment": "storeClient", "experienceUrl": "exp://10.0.0.159:19000", "expoConfig": {"__flipperHack": "React Native packager is running", "_internal": {"dynamicConfigPath": null, "isDebug": false, "packageJsonPath":
        - And so on, for pages

```
<SafeAreaView style={styles.screen}>
const styles = StyleSheet.create({
 screen: {
   paddingTop: Constants.statusBarHeight,
 },
});
```

EXPO SWIPING:

- `expo install react-native-gesture-handler`
    - When we use expo, it makes sure the library is compatible with the version of expo being used.
- Swipeable
    - import Swipeable from “react-native-gesture-handler/Swipeable”
    -

APP BUILDING

- Create a folder called “app” in the root directory so that if anything goes wrong with the React Native portion of the project, all the files are in one place and can be recreated easily.
- UI Toolkits:
    - https://react-native-elements.github.io/react-native-elements/
    - https://callstack.github.io/react-native-paper/
    - https://nativebase.io/
    -