Components

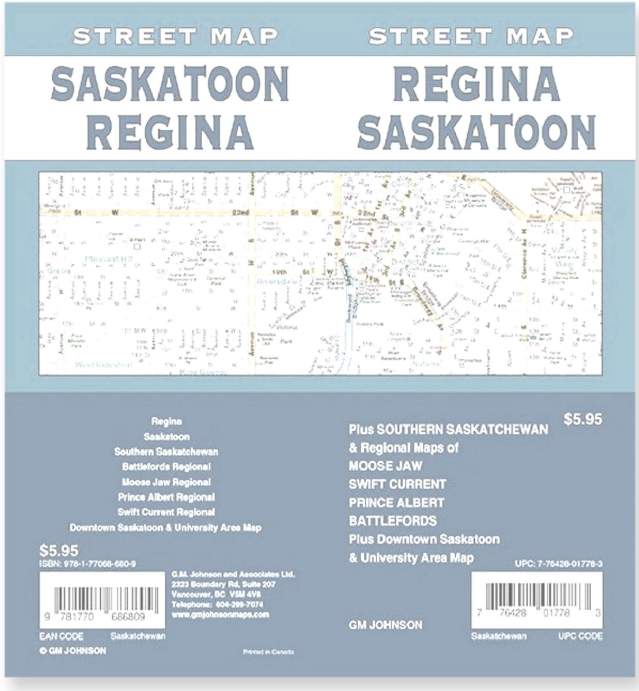
Develop Intelligence

Goals

1. List 2 options for creating a new React Native project
2. List 3 React Native components and their HTML analogs

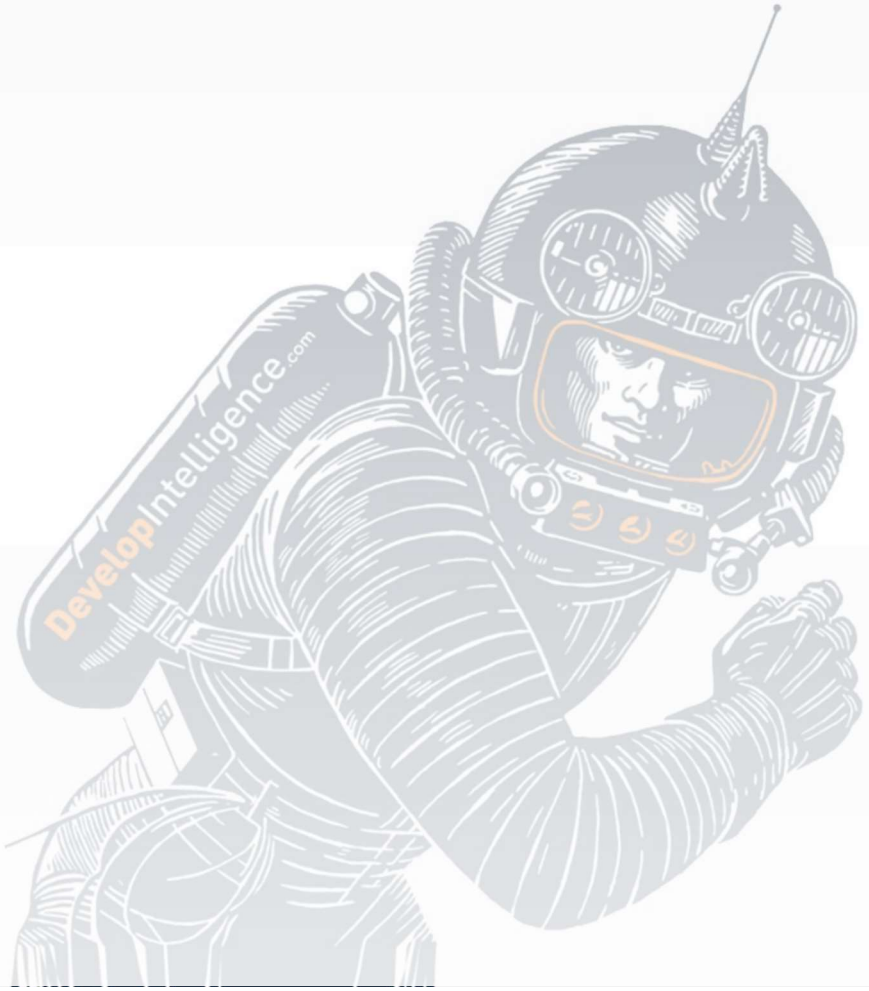
Roadmap

1. Scaffolding
2. Core Components
3. Lists
4. Pressables



Develop
Intelligence

Scaffolding



What's scaffolding?

Develop Intelligence

From wikipedia

Complicated software projects often share certain conventions on project structure and requirements. For example, they often have separate folders for source code, binaries and code tests, as well as files containing license agreements, release notes and contact information. To simplify the creation of projects following those conventions, "scaffolding" tools can automatically generate them at the beginning of each project.

Option 1: React Native CLI

Develop Intelligence

From Reactnative.dev

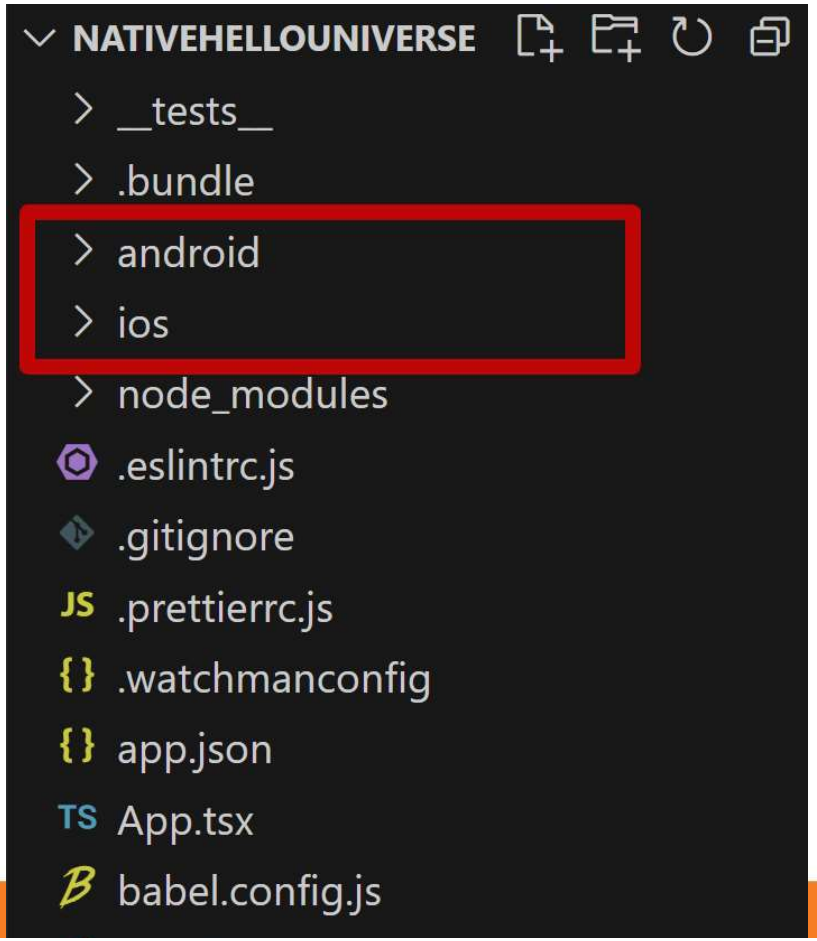
> *If you are already familiar with mobile development, you may want to use React Native CLI. It requires **Xcode** or **Android Studio** to get started.*

- Recipe:

```
1 npx react-native@latest init SomeNewProject
```

Project Structure

- Typescript by default
- Includes placeholders for working on the metal
- Notice:
  - No **/src** folder



Native CLI: Pros, Cons

- Less friendly
- Closer to metal

Option 2: Expo

Develop Intelligence

- Friendlier to learners
- Build on top of react native
- Compatible with React Native CLI (no ejecting)
- Includes lots of tools for--
  - In-browser development
  - Project scaffolding
  - Components
  - Deployment
  - iOS builds

Expo Recipe

Develop Intelligence

- New project:

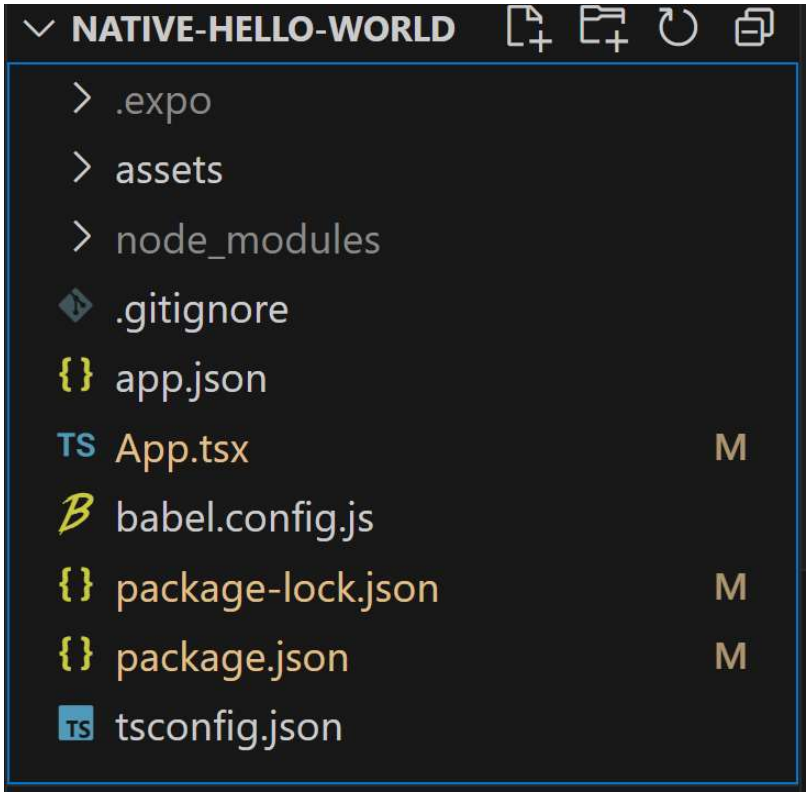
```
npx create-expo-app -t expo-template-blank-typescript
```

- Add React Native for Web:

```
npx expo install react-native-web react-dom @expo/webpack-config
```

Project Structure

- Typescript-- with the appropriate template
- Easy to set up web target



Feature: Expo Go

Develop Intelligence

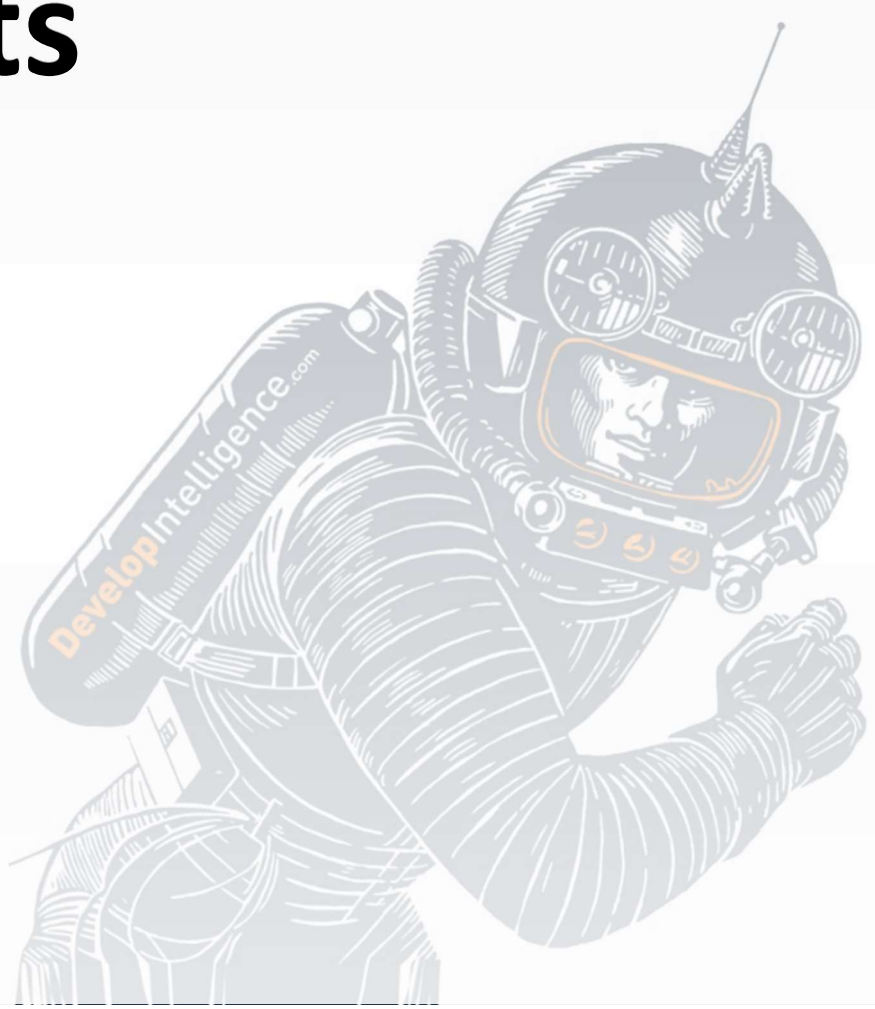
- Mobile client
- Quick native prototyping
  - No USB cable
  - Just scan the QR code

```
PS C:\Users\jake_\workspace\native-hello-world> npm run start

> native-hello-world@1.0.0 start
> expo start

Starting project at C:\Users\jake_\workspace\native-hello-world
Starting Metro Bundler
```

```
› Metro waiting on exp://192.168.0.40:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Using Expo Go
› Press s | switch to development build
```

Develop
Intelligence

Core Components



React v React Native

Develop Intelligence

- Same
  - JSX
  - Hooks-- `useState`, `useEffect`, etc.
  - Browser-like APIs-- e.g. `setTimeout`, `fetch`
  - Props & events
- Different
  - Native components instead of HTML
  - Styles instead of CSS
- **BUT** the DX feels familiar

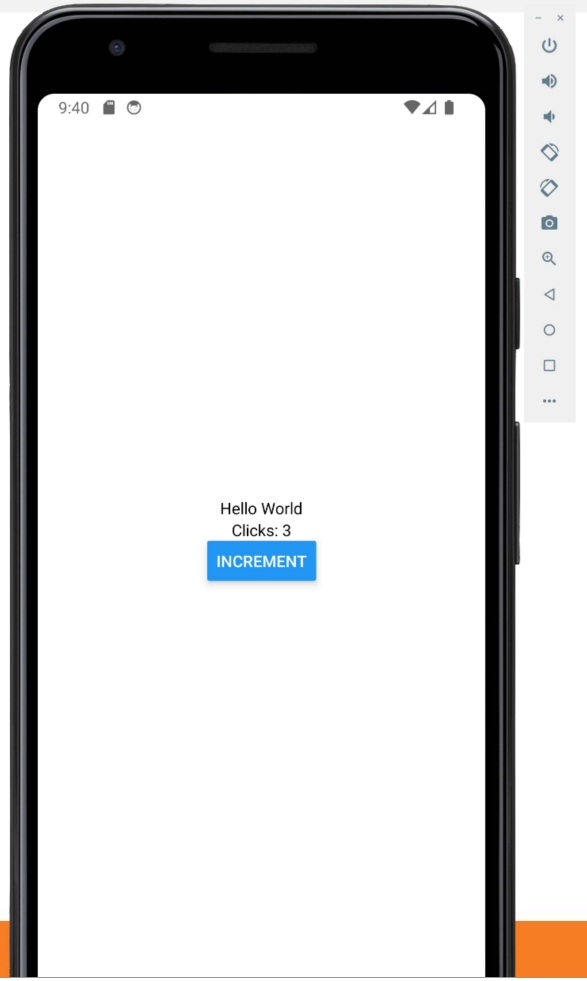
Native Components

- From the docs

> *In Android development, you write views in Kotlin or Java; in iOS development, you use Swift or Objective-C. With React Native, you can invoke these views with JavaScript using React components. At **runtime,** React Native creates the corresponding Android and iOS views for those components. Because React Native components are backed by the same views as Android and iOS, React Native apps look, feel, and perform like any other apps. We call these platform-backed components **Native Components**.*

Example: Hello World

- Stupid application
- Functionality
  - Says 'hello world'
  - Button increments click count



Hello World: Code

Develop Intelligence

```
1  import { useState } from 'react';
2  import { StyleSheet, Text, View, Button } from 'react-native';
3
4  export default function App() {
5    const [clicks, setClicks] = useState(0);
6    return (
7      <View style={styles.container}>
8        <Text>Hello World</Text>
9        <Text>Clicks: {clicks}</Text>
10       <Button title='Increment' onPress={()=>setClicks(clicks+1)}/>
11     </View>
12   );
13 }
```

Big 5 Core Components

Develop Intelligence

Native Control	HTML Equivalent
View	`<div>` (Non-scrolling)
ScrollView	`<div>`
Text	`<p>`
Image	`<img>`
TextInput	`<input type='text'>`

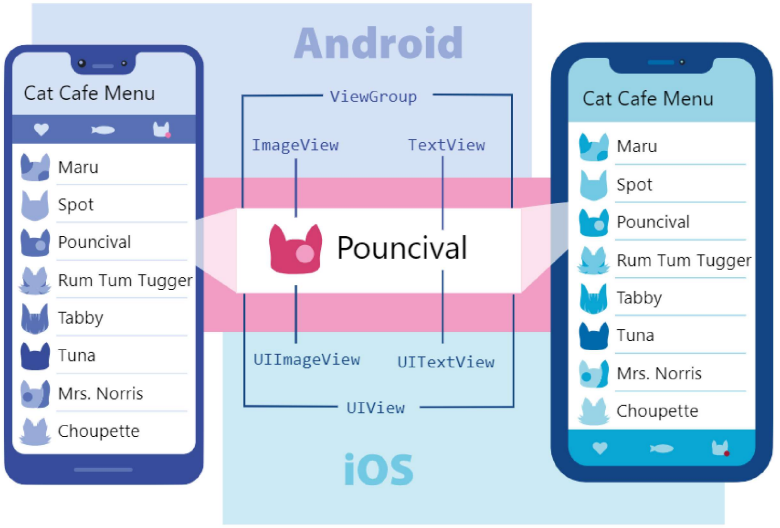
Memorize these right now!

Control: View

- Directly maps to `UIView`, `android.view`, and `<div>`
- Works well with flexbox layouts

Views are Everywhere

- **Layout-only views** are removed at runtime



Example

Develop Intelligence

```
export default  App = () =>
    <View>
      <Text style={{fontSize:96}}>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Quisque varius bibendum sodales. In hac habitasse platea
        dictumst. Fusce ac urna sollicitudin, condimentum dolor
        non, placerat lorem. Nulla facilisi.
      </Text>
      <Text style={{fontSize:96}}>
        Nam ut quam id justo molestie rutrum in vitae dolor. Aenean
        in semper enim. Sed auctor est sed purus accumsan, at suscipit
        justo ornare. Proin lobortis, lacus volutpat congue fermentum,
        ante ante pellentesque justo, vitae tempor odio ex eget erat.
      </Text>
    </View>;
```

Component: ScrollView

- Scrolling container
- Horizontal / vertical
- Paging via `pagingEnabled`
- Need a bounded height (or width)
  - Set `height` style directly
  - OR place inside a `View` that's bounded via flexbox
- Use for: Small number of things

Example

Develop Intelligence

```
export default App =() =>
    <ScrollView>
      <Text style={{fontSize:96}}>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Quisque varius bibendum sodales. In hac habitasse platea
        dictumst. Fusce ac urna sollicitudin, condimentum dolor
        non, placerat lorem. Nulla facilisi.
      </Text>
      <Text style={{fontSize:96}}>
        Nam ut quam id justo molestie rutrum in vitae dolor. Aenean
        in semper enim. Sed auctor est sed purus accumsan, at suscipit
        justo ornare. Proin lobortis, lacus volutpat congue fermentum,
        ante ante pellentesque justo, vitae tempor odio ex eget erat.
      </Text>
    </ScrollView>;
```

Control: Text

- Works mostly as expected
- Nestable
- Touchable
- Uses text layout, not flexbox

Example

Develop Intelligence

```
export default App =() => {
  const [title,setTitle] = useState('Lorem Ipsum');
  return <View>
          <Text
            style={{fontWeight:'bold',fontSize:36}}
            onPress={()=>setTitle('Sic dolor')}
          >
            {title}
          </Text>
            <Text>
              Subtitle something something...
              {'\n'}
              {'\n'}
              <Text numberOfLines={5}>The quick brown fox </Text>
            </Text>
        </View>;
```

Control: Image

- Works pretty much as expected
- `require` tells the bundler to package a static resource
- Example:

```
export const Images = () =>
    <View style={styles.container}>
      {/* Bundled  */}
      <Image
        source={require('../assets/fancy-logo.png')}
      />
      {/* Online  */}
      <Image
        source={{uri: 'https://example.com/img/small_logo.png'}}
      />
```

Control: TextInput

- Options for
  - Auto-correct
  - Placeholder text
  - Keyboard type

Example

Develop Intelligence

```
export default App =() =>
    <View>
      <Text style={{fontSize:32}}>Description</Text>
      <TextInput
        multiline={true}
        placeholder='Enter description here'
        style={{
          borderWidth:1,
          borderStyle:'solid',
          height:400,
          margin:4,
          padding:4,
        }}
      />
    </View>;
```

Develop
Intelligence

Lists

FlatList

Develop Intelligence

- Alternative to dynamically adding elements via `array.map()`
- Flatlist adds
  - Columns
  - Lazy rendering
  - Header & footer
  - ***And there's more!***
- Things I always forget
  - `renderItem` includes metadata-- **item** is the item
  - `keyExtractor` gets the key

Example: Motivation

Develop Intelligence

```
const coreComponents = ['View', 'Text', 'Image', 'ScrollView', 'TextInput'];

export default App =() => {
  return <Text style={{fontSize:128}}>
            {coreComponents.map(cc=><Text>{cc}{'\n'}</Text>)}
          </Text>;
}
```

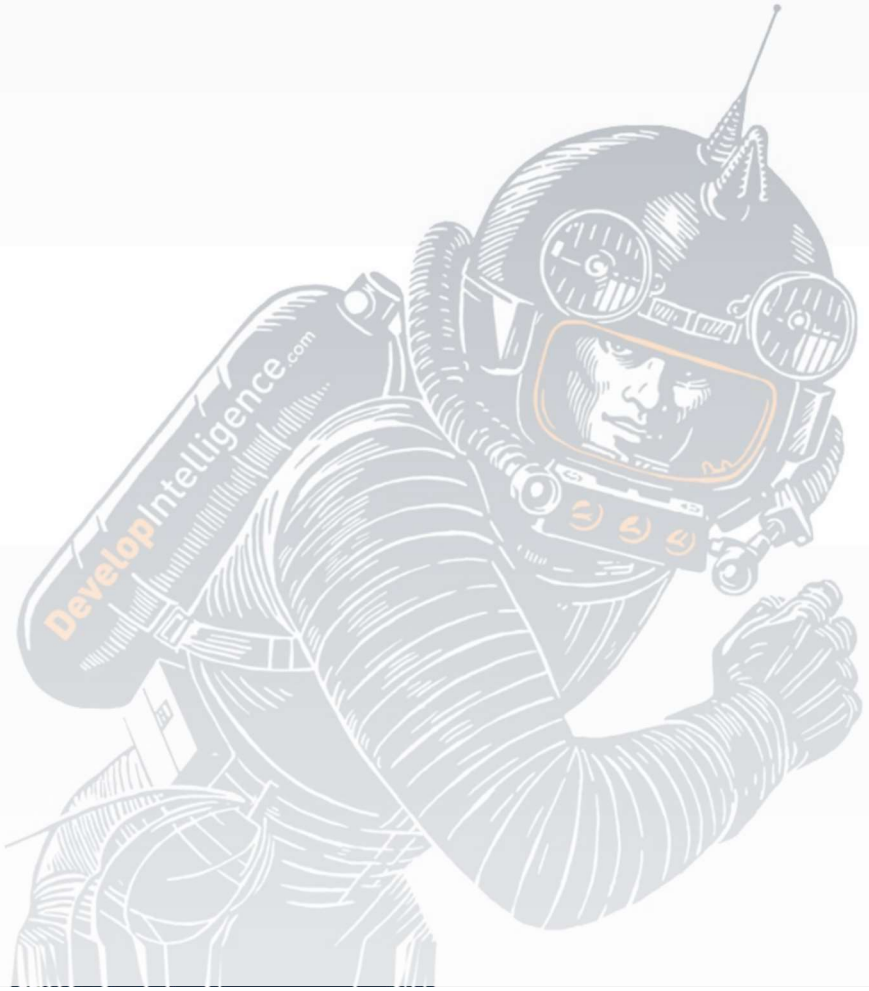
Example (Fixed)

Develop Intelligence

```
const coreComponents = ['View', 'Text', 'Image', 'ScrollView', 'TextInput'];

export default App =() => {
  return  <View>
            <FlatList
                style={{flex:1}}
                renderItem = {
                  ({item})=><Text style={{fontSize:128}}>{item}</Text>
                }
                data={coreComponents}
                keyExtractor={i => i.index}
              />
          </View>;
}
```

Develop
Intelligence

Pressables



Control: Button

- Self-closing
- **Doesn't** have a `style` property
  - Only `color`

```
<Button
  title='Increment'
  onPress={()=>setClicks(clicks+1)}
  color={'purple'}
  style={{fontSize:400}} //Doesn't work
/>
```

Control: Pressable

- Stylable alternative to `Button`
- No built-in animation
- No built-in style

```
<Pressable style={styles.button} onPress={onPress}>
  <Text style={styles.text}>{title}</Text>
</Pressable>
```

Touchables

- More usable alternatives
  - TouchableOpacity
  - TouchableHighlight

Example

Develop Intelligence

```
export default App =() => {
  const [components, setComponents] = useState(coreComponents);
  return  <FlatList
              renderItem = {({item})=>
                  <TouchableOpacity
                    onPress={()=>setComponents(components.filter(c=>c!==item)
                    style={{  backgroundColor:'lightgrey',
                              margin:3,
                              padding:5,
                              borderRadius:5,
                    }}>
                    <Text style={{fontSize:36}}>{item}</Text>
                  </TouchableOpacity>
              }
              data={components}
          />;
}
```

Develop
Intelligence

Review

1. List 2 options for creating a new React Native project
2. List 3 React Native components and their HTML analogs