CHEAT SHEETS HERO

React Native Cheatsheet

A quick reference guide for React Native development, covering core components, APIs, styling, navigation, and state management.

Core Components & APIs

Basic Components

`View`	The most fundamental component for building UI. Equivalent to a `div` in web development. Used for layout and styling.
`Text`	Used to display text. Must be used for any text content.
`Image`	Displays images. Supports local and remote images.
`TextInput`	Allows users to input text. Used for forms and user input.
`ScrollView`	A scrollable container. Used for content that exceeds the screen size.
`FlatList`	Efficiently renders large lists of data. Optimized for performance.

Core APIs

`Alert`	Displays an alert dialog with an optional title, message, and buttons.
`Dimensions`	Provides access to the screen dimensions (width and height).
`Platform`	Detects the platform the app is running on (e.g., 'ios', 'android').
`AsyncStorage`	A simple, unencrypted, persistent, key-value storage system.
`Linking`	Provides an interface to interact with incoming and outgoing app links.
`Keyboard`	Provides events and methods for handling the virtual keyboard.

Platform Specific Code

Use `Platform.OS === 'ios'` or `Platform.OS === 'android'` to run platform-specific code.

Example:

```
import { Platform, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    paddingTop: Platform.OS === 'ios' ? 20 : 0,
  },
});
```

Styling & Layout

Styling with StyleSheet

Use `StyleSheet.create` to define styles for your components.

Example:

```
import { StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  text: {
    fontSize: 20,
  },
});
```

Common Styles

`flex: 1`	Makes a component take up all available space.
`flexDirection`	Controls the direction of items in a flex container (`row`, `column`, `row-reverse`, `column-reverse`). Default is `column`.
`alignItems`	Defines how flex items are aligned along the cross axis (`flex-start`, `center`, `flex-end`, `stretch`).
`justifyContent`	Defines how flex items are aligned along the main axis (`flex-start`, `center`, `flex-end`, `space-around`, `space-between`, `space-evenly`).
`padding`, `margin`	Defines spacing around elements. Can specify `paddingTop`, `paddingBottom`, `paddingLeft`, `paddingRight`.
`backgroundColor`	Sets the background color of a component.

Flexbox Layout

React Native uses Flexbox for layout. Understand the concepts of main axis and cross axis for effective UI design.

- `flexDirection`: Determines the direction of the main axis.
- `justifyContent`: Aligns items along the main axis.
- `alignItems`: Aligns items along the cross axis.

Navigation

React Navigation

React Navigation is a popular library for handling navigation in React Native apps. Install it using:

```
yarn add @react-navigation/native @react-navigation/stack
yarn add react-native-reanimated react-native-gesture-handler react-native-screens react-native-safe-area-context @react-native-community/masked-view
```

For Expo managed projects:

```
expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view
```

Stack Navigator

Creating a Stack Navigator

```
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

function MyStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Details" component={DetailsScreen} />
    </Stack.Navigator>
  );
}
```

Navigating between screens

```
navigation.navigate('Details', { itemId: 86 });
```

Passing parameters

```
<Button
  title="Go to Details"
  onPress={() => {
    navigation.navigate('Details', { itemId: 86, otherParam: 'anything you want here' });
  }}
/>
```

Tab Navigator

Creating a Tab Navigator

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Tab = createBottomTabNavigator();

function MyTabs() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Home" component={HomeScreen} />
      <Tab.Screen name="Settings" component={SettingsScreen} />
    </Tab.Navigator>
  );
}
```

State Management

useState Hook

The `useState` hook is the most basic way to manage state in a functional component.

Example:

```
import React, { useState } from 'react';
import { Button, Text, View } from 'react-native';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>Count: {count}</Text>
      <Button title="Increment" onPress={() => setCount(count + 1)} />
    </View>
  );
}
```

useReducer Hook

The `useReducer` hook is useful for managing more complex state logic.

Example:

```
import React, { useReducer } from 'react';
import { Button, Text, View } from 'react-
native';

const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
};

function Counter() {
  const [state, dispatch] =
useReducer(reducer, { count: 0 });

  return (
    <View>
      <Text>Count: {state.count}</Text>
      <Button title="Increment" onPress={() =>
dispatch({ type: 'increment' })} />
      <Button title="Decrement" onPress={() =>
dispatch({ type: 'decrement' })} />
    </View>
  );
}
```

Context API

The Context API allows you to share state between components without having to pass props manually at every level.

Example:

```
import React, { createContext, useContext,
useState } from 'react';
import { Button, Text, View } from 'react-
native';

const CountContext = createContext();

function CounterProvider({ children }) {
  const [count, setCount] = useState(0);
  return (
    <CountContext.Provider value={{ count,
setCount }}>
      {children}
    </CountContext.Provider>
  );
}

function useCount() {
  return useContext(CountContext);
}

function Counter() {
  const { count, setCount } = useCount();

  return (
    <View>
      <Text>Count: {count}</Text>
      <Button title="Increment" onPress={() =>
setCount(count + 1)} />
    </View>
  );
}

function App() {
  return (
    <CounterProvider>
      <Counter />
    </CounterProvider>
  );
}
```