ZTM

REACT NATIVE CHEAT SHEET

MO BINNI

V1.01

HEEELLLOOOOO!

I'm Andrei Neagoie, Founder and Lead Instructor of the Zero To Mastery Academy.

After working as a Senior Software Developer over the years, I now dedicate 100% of my time to teaching others in-demand skills, help them break into the tech industry, and advance their careers.

In only a few years, **over 1,000,000 students** around the world have taken Zero To Mastery courses and many of them are now working at top tier companies like Apple, Google, Amazon, Tesla, IBM, Facebook, and Shopify, just to name a few.

This cheat sheet, created by our React Native Instructor (Mo Binni) provides you with the key React Native information and concepts that you need to know and remember.

If you want to not only learn React Native but also get the exact steps to build your own projects and get hired as a Developer, then check out our Career Paths.

Happy Learning!
Andrei

[illegible]

Founder & Lead Instructor, Zero To Mastery
Andrei Neagoie

React Native Cheat Sheet

React Native is a popular framework for building native mobile applications using JavaScript and React. This cheat sheet covers the essentials to help you get started and serves as your a React Native quick reference guide!

Table of Contents

1. Setup

Install Expo CLI

Expo CLI provides a convenient way to start React Native projects.

```
npm install -g expo-cli
```

Create a New Project

```
expo init MyApp
cd MyApp
npm start
```

2. Core Components

React Native provides a set of core components for building user interfaces.

Basic Components

- `View`: The fundamental component for building UI, similar to a `div` in HTML.
- `Text`: Used for displaying text.
- `Image`: Displays images.
- `ScrollView`: A scrollable container for views.
- `TextInput`: For text input fields.
- `TouchableOpacity`: For wrapping components to make them touch-responsive.

Example

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

const App = () => (
  <View>
    <Text>Hello, World!</Text>
    <Image
      source={{ uri: 'https://example.com/image.png' }}
      style={{ width: 100, height: 100 }}
    />
  </View>
);
```

```
export default App;
```

3. Styling

React Native uses JavaScript objects for styling, similar to inline styles in HTML but with a CSS-like syntax.

Example

```
const styles = {
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
};

<View style={styles.container}>
  <Text style={{ color: 'blue', fontSize: 20 }}>Styled Text</Text>
</View>
```

- **Units**: All units are density-independent pixels (dp).
- **Property Names**: Use camelCase (e.g., `backgroundColor`).

4. Flexbox Layout

React Native uses Flexbox for layout, providing a powerful way to arrange components.

Common Flexbox Properties

- `flexDirection`: Determines the primary axis (`'row'` or `'column'`).
- `justifyContent`: Aligns children along the primary axis.
- `alignItems`: Aligns children along the secondary axis.
- `flex`: Determines how a component grows or shrinks.

Example

```
<View style={{ flex: 1, flexDirection: 'row' }}>
  <View style={{ flex: 1, backgroundColor: 'red' }} /><View style={{ flex: 2,
backgroundColor: 'green' }} /><View style={{ flex: 3, backgroundColor: 'blue'
}} />
</View>
```

5. State and Props

Props

Props are inputs to components, allowing data to be passed from parent to child.

```
const Greeting = (props) => (
  <Text>Hello, {props.name}!</Text>
);

<Greeting name="Alice" />
```

State

State allows components to create and manage their own data.

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

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

  return (
    <View>
      <Text>You clicked {count} times</Text>
      <Button onPress={() => setCount(count + 1)} title="Click me" />
    </View>
  );
};
```

6. Handling Events

Events are handled using callback functions passed as props.

Example

```
<Button
  onPress={() => {
    alert('Button pressed!');
  }}
  title="Press Me"
/>
```

Common Event Handlers

- `onPress`: Triggered when a component is pressed.
- `onChangeText`: Used with `TextInput` to handle text changes.
- `onSubmitEditing`: Triggered when the user submits the text input.

7. Lists

FlatList

The optimal way to render scrolling lists of data, it has various properties that you can adjust and adapt to allow for more efficient rendering.

```
import { FlatList, Text } from 'react-native';

<FlatList
  data={[{ key: 'Alice' }, { key: 'Bob' }]}
  renderItem={({ item }) => <Text>{item.key}</Text>}
/>
```

SectionList

For rendering sections with headers.

```
import { SectionList, Text } from 'react-native';

<SectionList
  sections={[
    { title: 'A', data: ['Alice', 'Alan'] },
    { title: 'B', data: ['Bob', 'Bill'] },
  ]}
  renderItem={({ item }) => <Text>{item}</Text>}
  renderSectionHeader={({ section }) => <Text>{section.title}</Text>}
/>
```

8. Navigation

React Navigation is the standard library for handling navigation in React Native apps. Make sure to utilize native stack navigator for the full native experience and performance benefits.

Installation

```
npm install @react-navigation/native
npm install @react-navigation/native-stack
expo install react-native-screens react-native-safe-area-context
```

Example

```
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

function HomeScreen({ navigation }) {
  return (
    <Button
      title="Go to Details"
      onPress={() => navigation.navigate('Details')}
    />
  );
}

function DetailsScreen() {
  return (
    <View>
      <Text>Details Screen</Text>
    </View>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

9. Networking

Use the `fetch` API or libraries like `axios` for network requests.

Example with Fetch

```
import React, { useEffect, useState } from 'react';
import { FlatList, Text } from 'react-native';

const FetchExample = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch('https://example.com/data')
      .then((response) => response.json())
      .then((json) => setData(json.items))
      .catch((error) => console.error(error));
  }, []);

  return (
    <FlatList
      data={data}
      renderItem={({ item }) => <Text>{item.name}</Text>}
      keyExtractor={(item) => item.id.toString()}
    />
  );
};
```

10. Platform-Specific Code

Use the `Platform` module to conditionally render code based on the platform.

Example

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

const instructions = Platform.select({
  ios: 'Press Cmd+R to reload',
  android: 'Double tap R on your keyboard to reload',
});

<Text>{instructions}</Text>
```

- **`Platform.OS`**: Returns `'ios'`, `'android'`, or `'web'`.

11. Animations

React Native provides the `Animated` API for creating animations.

Example

```
import React, { useRef, useEffect } from 'react';
import { Animated, Text } from 'react-native';

const FadeInView = (props) => {
  const fadeAnim = useRef(new Animated.Value(0)).current; // Initial opacity
value

  useEffect(() => {
    Animated.timing(fadeAnim, {
      toValue: 1, // Fade to opacity 1
      duration: 1000,
      useNativeDriver: true,
    }).start();
  }, [fadeAnim]);

  return (
    <Animated.View style={{ opacity: fadeAnim }}>
      {props.children}
    </Animated.View>
  );
};

<FadeInView>
  <Text>Fading in</Text>
</FadeInView>;
```

---

12. Hooks

React Native supports React Hooks for state and lifecycle management.

`useState`

```
import React, { useState } from 'react';

const [value, setValue] = useState(initialValue);
```

`useEffect`

```
import React, { useEffect } from 'react';

useEffect(() => {
  // Perform side effects here
  return () => {
    // Cleanup if necessary
  };
}, [dependencies]);
```

---

13. Third-Party Libraries

Install additional libraries using `npm` or `yarn`.

Example: Using `axios`

```
npm install axios
```

```
import axios from 'axios';

axios
  .get('https://example.com/data')
  .then((response) => {
    // Handle success
  })
  .catch((error) => {
    // Handle error
  });
```

---

14. Debugging

- **Logging**: Use `console.log()` for debugging.
- **Developer Menu**: Shake your device or press `Ctrl + M` (`Cmd + M` on Mac) to open the developer menu.
- **Debugging Tools**: Use React Native Debugger or Chrome DevTools.

---

15. Useful Commands

Start the Development Server

```
npm start
```

Run on Android Emulator

```
npm run android
```

Run on iOS Simulator

```
npm run ios
```

Build APK (Android)

```
expo build:android
```

Build IPA (iOS)

```
expo build:ios
```

---

16. TypeScript in React Native

TypeScript adds optional static typing to JavaScript, which can help catch errors early and improve code quality.

Setup

To add TypeScript to a React Native project:

```
npx react-native init MyApp --template react-native-template-typescript
```

If you have an existing project:

```
npm install --save-dev typescript @types/react @types/react-native
```

Example

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

interface Props {
  name: string;
}

const Greeting: React.FC<Props> = ({ name }) => (
  <View>
    <Text>Hello, {name}!</Text>
  </View>
);

export default Greeting;
```

17. Testing

Testing ensures your app works as expected and reduces bugs.

Jest

React Native comes with Jest, a JavaScript testing framework.

Installation

```
npm install --save-dev jest @testing-library/react-native react-test-renderer
```

Example Test

```
import React from 'react';
import renderer from 'react-test-renderer';
import App from '../App';

test('renders correctly', () => {
  const tree = renderer.create(<App />).toJSON();
  expect(tree).toMatchSnapshot();
});
```

Testing Library

For testing components with user interactions.

Example

```
import React from 'react';
import { render, fireEvent } from '@testing-library/react-native';
import MyButton from '../MyButton';

test('Button presses correctly', () => {
  const onPressMock = jest.fn();
  const { getByText } = render(
    <MyButton onPress={onPressMock} title="Press me" />
  );

  fireEvent.press(getByText('Press me'));
  expect(onPressMock).toHaveBeenCalled();
});
```

18. Accessibility

Making your app accessible ensures it can be used by as many people as possible.

Accessibility Props

- `accessible`: Marks a component as accessible.
- `accessibilityLabel`: Provides a readable label for screen readers.
- `accessibilityHint`: Provides additional information about a component.

Example

```
<TouchableOpacity
  accessible={true}
  accessibilityLabel="Play Button"
  accessibilityHint="Plays the current track"
  onPress={handlePlay}
>
  <Text>Play</Text>
</TouchableOpacity>
```

19. Permissions

To access certain device features, you may need to request permissions.

Using `PermissionsAndroid` (Android)

Example

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

async function requestCameraPermission() {
  try {
    const granted = await PermissionsAndroid.request(
      PermissionsAndroid.PERMISSIONS.CAMERA,
      {
        title: 'Camera Permission',
        message: 'This app needs access to your camera',
        buttonNeutral: 'Ask Me Later',
        buttonNegative: 'Cancel',
        buttonPositive: 'OK',
      }
    );
    if (granted === PermissionsAndroid.RESULTS.GRANTED) {
      console.log('You can use the camera');
    } else {
      console.log('Camera permission denied');
    }
  } catch (err) {
    console.warn(err);
  }
```

Using `react-native-permissions` (Cross-platform)

Installation

```
npm install react-native-permission
```

Example

```
import { check, request, PERMISSIONS, RESULTS } from 'react-native-permission
s';

async function checkCameraPermission() {
  const result = await check(PERMISSIONS.ANDROID.CAMERA);
  switch (result) {
    case RESULTS.GRANTED:
      console.log('Camera permission granted');
      break;
    case RESULTS.DENIED:
      const requestResult = await request(PERMISSIONS.ANDROID.CAMERA);
      if (requestResult === RESULTS.GRANTED) {
        console.log('Camera permission granted');
      }
      break;
    // Handle other cases
  }
}
```

20. Device APIs

React Native provides access to various device APIs.

Geolocation

```
import Geolocation from '@react-native-community/geolocation';

Geolocation.getCurrentPosition(
  (position) => {
    console.log(position);
  },
  (error) => {
    console.log(error);
  },
  { enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 }
);
```

Camera Roll

```
import { CameraRoll } from '@react-native-community/cameraroll';

CameraRoll.getPhotos({
  first: 20,
  assetType: 'Photos',
})
  .then((r) => {
    this.setState({ photos: r.edges });
  })
  .catch((err) => {
    console.log(err);
  });
```

Clipboard

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

Clipboard.setString('Hello World');

Clipboard.getString().then((content) => {
  console.log(content);
});
```

21. Offline Storage

AsyncStorage

AsyncStorage is a simple, unencrypted, asynchronous, persistent, key-value storage system.

Installation

```
npm install @react-native-async-storage/async-storage
```

Example

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

const storeData = async (value) => {
  try {
    await AsyncStorage.setItem('@storage_Key', value);
  } catch (e) {
    // saving error
  }
};

const getData = async () => {
  try {
    const value = await AsyncStorage.getItem('@storage_Key');
    if (value !== null) {
      // value previously stored
    }
  } catch (e) {
    // error reading value
  }
};
```

22. Performance Optimization

Optimizing performance ensures your app runs smoothly.

Use `PureComponent` or `React.memo`

Prevents unnecessary re-renders.

```
import React from 'react';

const MyComponent = React.memo(({ prop1, prop2 }) => {
  // Component code
});
```

Avoid Anonymous Functions in `render`

Pass functions defined outside `render` to child components.

Use `useCallback` and `useMemo`

Optimize expensive functions and computations.

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

---

23. Internationalization (i18n)

Support multiple languages in your app.

Using `react-native-i18n`

Installation

```
npm install react-native-i18n
```

Example

```
import I18n from 'react-native-i18n';

I18n.fallbacks = true;
I18n.translations = {
  en: { welcome: 'Welcome' },
  fr: { welcome: 'Bienvenue' },
};

<Text>{I18n.t('welcome')}</Text>
```

Using `react-native-localize` and `i18n-js`

A modern approach to internationalization.

Installation

```
npm install react-native-localize i18n-js
```

Example

```
import * as RNLocalize from 'react-native-localize';
import i18n from 'i18n-js';
```

```
const locales = RNLocalize.getLocales();

if (Array.isArray(locales)) {
  i18n.locale = locales[0].languageTag;
}

i18n.fallbacks = true;
i18n.translations = {
  en: { welcome: 'Welcome' },
  fr: { welcome: 'Bienvenue' },
};

<Text>{i18n.t('welcome')}</Text>
```

24. Error Handling

Handling errors gracefully improves user experience.

Error Boundaries

Catch JavaScript errors in components.

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

class ErrorBoundary extends React.Component {
  state = { hasError: false };

  componentDidCatch(error, info) {
    // Log error
    this.setState({ hasError: true });
  }

  render() {
    if (this.state.hasError) {
      return <Text>Something went wrong.</Text>;
    }
    return this.props.children;
  }
}

<ErrorBoundary>
```

```
  <MyComponent />
</ErrorBoundary>;
```

Try-Catch Blocks

Use try-catch blocks in async functions.

```
try {
  const response = await fetchData();
} catch (error) {
  console.error(error);
}
```

25. Best Practices

Following best practices helps maintain code quality.

Directory Structure

Organize your files logically.

```
- src/
  - components/
  - screens/
  - navigation/
  - services/
  - assets/
```

Linting and Formatting

Use ESLint and Prettier.

```
npm install --save-dev eslint prettier eslint-config-prettier eslint-plugin-p
rettier
```

Write Reusable Components

Create components that can be reused throughout your app.

Comment and Document Your Code

Use comments and documentation to explain complex logic.

Use PropTypes or TypeScript

Ensure component props are correctly used.

```
import PropTypes from 'prop-types';

MyComponent.propTypes = {
  name: PropTypes.string.isRequired,
};
```

This React Native cheat sheet provides a comprehensive reference to React Native concepts, including testing, accessibility, permissions, device APIs, offline storage, performance optimization, internationalization, error handling, and best practices. For more detailed information, refer to the official React Native documentation and resources.