Cheat Sheet: React Native Using Expo for Using FlatList and Async Storage

Download the **Cheat Sheet: React Native Using Expo for Using FlatList and Async Storage** document to access the Expo commands and codes, their descriptions, and some example of commands/codes covered in this module.

Note: If the cheat sheet doesn't download, right-click and open the link in a new tab.

Expo Commands and Codes	**Description**	**Command/Code Example**

Event Handling	The press event-handling is built into the button.	`import React, { useState } from 'react';`
		`import { View, Button, StyleSheet } from 'react-native';`
		`const App = () => {`
		`const [buttonText, setButtonText] = useState('Press Me');`
		`const handlePress = () => {`
		`setButtonText('I was pressed');`
		`setTimeout(() => setButtonText('Press Me'), 500); // Revert text after 500ms (0.5 seconds)`
		`};`
		`return (`
		`<View style={styles.container}>`
		`<Button title={buttonText} onPress={handlePress} />`
		`</View>`
		`);`
		`};`
		`const styles = StyleSheet.create({`
		`container: {`
		`flex: 1,`
		`justifyContent: 'center',`
		`alignItems: 'center',`
		`},`
		`});`
		`export default App;`

onPress	onPress - Triggered when the user presses	`import { View, TouchableOpacity, Text, StyleSheet, Alert } from 'react-native';`
		`const App = () => {`

	and releases a touchable component.	`return (`
		`<View style={styles.container}>`
		`<TouchableOpacity onPress={() => Alert.alert('You tapped the button!')}>`
		`<Text style={styles.text}>Press Me</Text>`
		`</TouchableOpacity>`
		`</View>`
		`);`
		`};`
		`const styles = StyleSheet.create({`
		`container: {`
		`flex: 1,`
		`justifyContent: 'center',`
		`alignItems: 'center'`
		`},`
		`text: {`
		`fontSize: 20,`
		`color: 'blue'`
		`},`
		`});`
		`export default App;`

onLongPress	onLongPress is invoked when the user presses the button for more than half a second by defualt	`import { View, TouchableOpacity, Text, StyleSheet, Alert } from 'react-native';`
		`const App = () => {`
		`return (`
		`<View style={styles.container}>`
		`<TouchableOpacity`
		`onLongPress={() => Alert.alert('Button Long Pressed')}`
		`delayLongPress={2000}>`
		`<Text style={styles.text}>Press Me</Text>`
		`</TouchableOpacity>`
		`</View>`
		`);`
		`};`
		`const styles = StyleSheet.create({`
		`container: {`
		`flex: 1,`
		`justifyContent: 'center',`
		`alignItems: 'center'`
		`},`
		`text: {`
		`fontSize: 20,`
		`color: 'blue'`
		`},`
		`});`
		`export default App;`

onSubmitEditing Event	The onSubmitEditing prop in React Native's TextInput component	`mport { View, TextInput, Text, StyleSheet } from 'react-native';`
		`const App = () => {`
		`const [name, setName] = useState('');`
		`const [greeting, setGreeting] = useState('');`

	triggers a function when the user submits the input, typically by pressing the "Enter" or "Done" key on the keyboard.	`const handleNameChange = (text) => {`
		`setName(text);`
		`};`
		`const handleNameSubmit = () => {`
		`` setGreeting(`Hello, ${name}!`); ``
		`}`
		`return (`
		`<View style={styles.container}>`
		`<TextInput style={styles.input} placeholder="Enter your name" value={name}`
		`onSubmitEditing={handleNameSubmit} onChangeText={handleNameChange}`
		`/>`
		`{greeting ? <Text style={styles.greeting}>{greeting}</Text> : null}`
		`</View>`
		`);`
		`};`

keyboardType Event	The keyboardType prop in React Native's TextInput component specifies the type of keyboard to display, depending on the input field's purpose, such as email, numeric, or phone number.	`<View style={styles.container}>`
		`      <TextInput`
		`        style={styles.input}`
		`        placeholder="Enter phone number"`
		`        value={phoneNumber}`
		`        onChangeText={setPhoneNumber}`
		`        keyboardType="phone-pad"  // Shows a numeric keypad`
		`      />`
		`    </View>`

secureTextEntry	The secureTextEntry prop in React Native's TextInput component is used to hide the text input, typically for password fields, by displaying dots or asterisks instead of the actual characters.	`<TextInput`
		`style={styles.input}`
		`placeholder="Password"`
		`value={formData.password}`
		`onChangeText={(value) => handleInputChange('password', value)}`
		`secureTextEntry // Hides the password input`
		`/>`
		`<Button title="Submit" onPress={handleSubmit} />`
		`</View>`

TextInput	This is used to create a textbox for users to put information.	`<TextInput style={styles.input}/>`

onChangeText	This code defines a TextInput component in React Native that allows users to input text, with the value bound to name	`<TextInput style={styles.input} placeholder="Enter your name" value={name}`
		`onChangeText={handleNameChange}`
		`/>`

	and the onChangeText prop triggering the handleNameChange function whenever the text is changed.	
Formik Package	The command installs the Formik library, which helps manage form state, validation, and submission in React and React Native applications more easily.	`npm install formik`

Formik	Formik is used as a form handling library that simplifies form creation in React Native by managing the form's state, validation, and submission. It provides methods like handleChange, handleBlur, and handleSubmit to handle form interactions and updates, as well as capturing user input for fields like name, email, and password.	`const App = () => {`
		`return (`
		`<Formik`
		`initialValues={{ name: '', email: '', password: '' }}`
		`validationSchema={validationSchema}`
		`onSubmit={values => console.log(values)}`
		`>`
		`{({ handleChange, handleBlur, handleSubmit, values }) => (`
		`<View style={styles.form}>`
		`<TextInput`
		`placeholder="Name"`
		`onChangeText={handleChange('name')}`
		`onBlur={handleBlur('name')}`
		`value={values.name}`
		`style={styles.input}`
		`/>`
		`<TextInput`
		`placeholder="Email"`
		`onChangeText={handleChange('email')}`
		`onBlur={handleBlur('email')}`
		`value={values.email}`
		`style={styles.input}`
		`/>`
		`<TextInput`
		`placeholder="Password"`
		`onChangeText={handleChange('password')}`
		`onBlur={handleBlur('password')}`
		`value={values.password}`
		`secureTextEntry`
		`style={styles.input}`
		`/>`
		`<Button onPress={handleSubmit} title="Submit" />`
		`</View>`
		`)}`
		`</Formik>`

		`);`
		`};`

yup Package	The command npm install yup installs the yup library, which is used for schema-based validation of JavaScript objects, commonly used to validate form inputs in React and React Native applications.	`npm install yup`

yup	yup is a schema validation library used to define and validate the structure of form fields. It creates a validation schema that ensures the name is required, the email is both required and in a valid format, and the password is required with a minimum length of 6 characters.	`const validationSchema = Yup.object().shape({`
		`name: Yup.string().required('Name is required'),`
		`email: Yup.string().email('Invalid email').required('Email is required'),`
		`password: Yup.string()`
		`.min(6, 'Password must be at least 6 characters')`
		`.required('Password is required'),`
		`});`

Flat List for Multiple Column Support	FlatLists also provide Multiple-column support which is useful in displaying the items in multiple columns.	`const App = () => {`
		`return (`
		`<View style={styles.container}>`
		`<FlatList data={proverbs}`
		`renderItem={({ item }) => (`
		`<View style={styles.itemContainer}>`
		`<Text style={styles.itemText}>{item}</Text> </View>`
		`)}`
		`keyExtractor={(item, index) => index.toString()}`
		`numColumns={2} // Set the number of columns`
		`ListHeaderComponent={() => <Text style={styles.header}>Words of Wisdom</Text>}`
		`/>`
		`</View>`
		`);`
		`};`

renderSectionHeader and renderSectionFooter prop	Section Headers allows you to set a header for every section using the renderSectionHeader prop. Section Footers allows you to set a footer for every section using the renderSectionFooter prop.	`const App = () => {`
		`const initialVisibleItems = DATA.map(section => section.data.slice(0, 3));`
		`const [visibleItems, setVisibleItems] = useState(initialVisibleItems);`
		`const showMore = (sectionIndex) => {`
		`//Imkplement method to handle showing more data`
		`};`
		`return (`
		`<View style={styles.container}>`
		`<SectionList`
		`sections={DATA.map((section, index) => ({`
		`...section, data: visibleItems[index], }))}`
		`keyExtractor={(item, index) => item + index}`
		`renderItem={({ item }) => (`
		`<View style={styles.item}>`
		`<Text style={styles.title}>{item}</Text>`
		`</View>`
		`)}`
		`renderSectionHeader={({ section: { title } }) => (`
		`<View>`
		`<Text style={styles.sectionHeader}>{title}</Text>`
		`</View>`
		`)}`
		`renderSectionFooter={({ section, index }) => r`
		`enderShowMoreButton(index)}`
		`ItemSeparatorComponent={() => <View style={styles.separator} />}`
		`/>`
		`</View>`
		`);`
		`}`

onEndReached, onEndReachedThreshold	onEndReachedThreshold in React Native's FlatList sets the distance from the bottom of the list at which the onEndReached event will be triggered. onEndReached is an event handler in FlatList that triggers a function when the user scrolls to the end of the list, often used for loading more data for infinite scroll.	`` const INITIAL_DATA = Array.from({ length: 10 }, (_, i) => `Item ${i + 1}`); ``
		`const App = () => {`
		`const [data, setData] = useState(INITIAL_DATA);`
		`const [loading, setLoading] = useState(false);`
		`const [page, setPage] = useState(1);`
		`const [hasMore, setHasMore] = useState(true);`
		`const fetchMoreData = () => {`
		`if (loading) return; // Stop fetching if already loading setLoading(true);`
		`/*add asynchronous call to request data when response received, refresh data using`
		`state and setLoading(false); */`
		`if (newData.length === 0)`
		`setHasMore(false);`
		`};`
		`useEffect(() => {`
		`fetchMoreData();`
		`}, []);`
		`const handleLoadMore = () => {`
		`if (!loading && hasMore) { //Called only if it is not already loading`
		`fetchMoreData();`
		`}`

		`};`
		`return (`
		`<View style={styles.container}>`
		`<FlatList data={data} renderItem={({ item }) => (`
		`<View style={styles.item}>`
		`<Text>{item}</Text>`
		`</View>`
		`)}`
		`keyExtractor={(item, index) => item + index}`
		`onEndReached={handleLoadMore} // Trigger load more when scrolled to end`
		`onEndReachedThreshold={0.5} // Threshold to trigger load more (0.5 means 50%`
		`of the list is visible)`
		`ListFooterComponent={loading ? <ActivityIndicator size="small" /> : null} // Show`
		`loader at the bottom`
		`/>`
		`</View>`
		`);`
		`};`

Async Storage package	The command npm install installs the Async Storage library in React Native, allowing the app to store and persist key-value data locally on the device.	`npm install @react-native-async-storage/async-storage`
AsyncStorage.setItem(	Adding the username to the storage.	`await AsyncStorage.setItem('@username','John' );`
AsyncStorage.getItem	This is used to retrieve the username from the storage.	`const username = await AsyncStorage.getItem('@username');`
AsyncStorage.removeItem()	This is used to delete the username from the storage.	`await AsyncStorage.removeItem('@username')`