國立高雄科技大學 智慧商務系
National Kaohsiung University of Science and Technology Department of Intelligent Commerce

9. React Native 基本介紹 III

廖奕雯
yiwen923@nkust.edu.tw

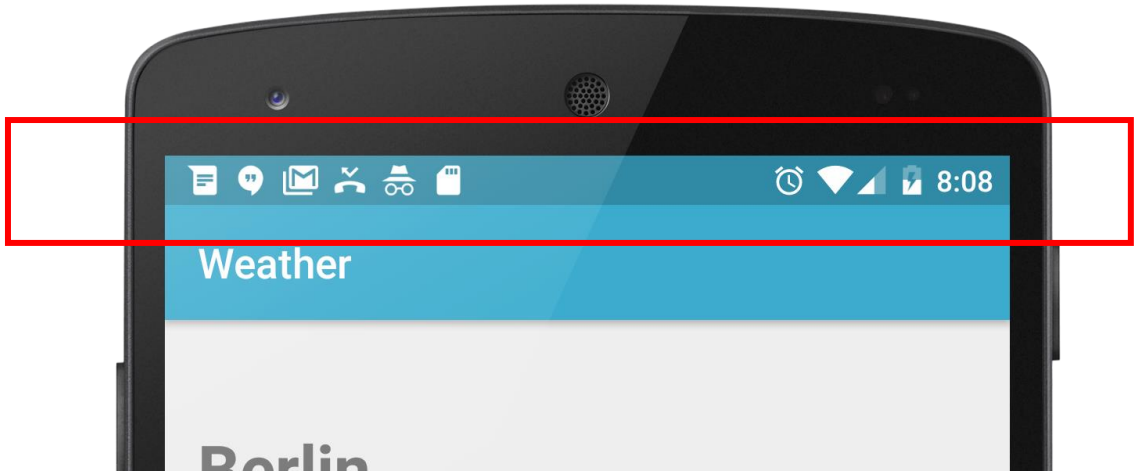
大綱

- StatusBar component
- Hook
- Fetch

StatusBar

StatusBar

- Component to control the app's status bar.
- The status bar is the zone, typically at the top of the screen, that displays the current time, Wi-Fi and cellular network information, battery level and/or other status icons.



StatusBar常見屬性

屬性	描述
currentHeight	狀態列的當前高度。 (僅限 Android)
backgroundColor	狀態列的背景色。
hidden	是否隱藏狀態列。
barStyle	設置狀態列文字的顏色。 • default :預設的樣式（ IOS 為白底黑字、Android 為黑底白字 ） • light-content : 黑底白字 • dark-content :白底黑字（ 需要 Android API>=23 ）

StatusBar 常見 functions

function	描述
setBackgroundColor	設置狀態列的背景色。僅限 Android。
setBarStyle()	設置狀態列的樣式
setHidden()	顯示 / 隱藏狀態列

StatusBar

- 使用步驟

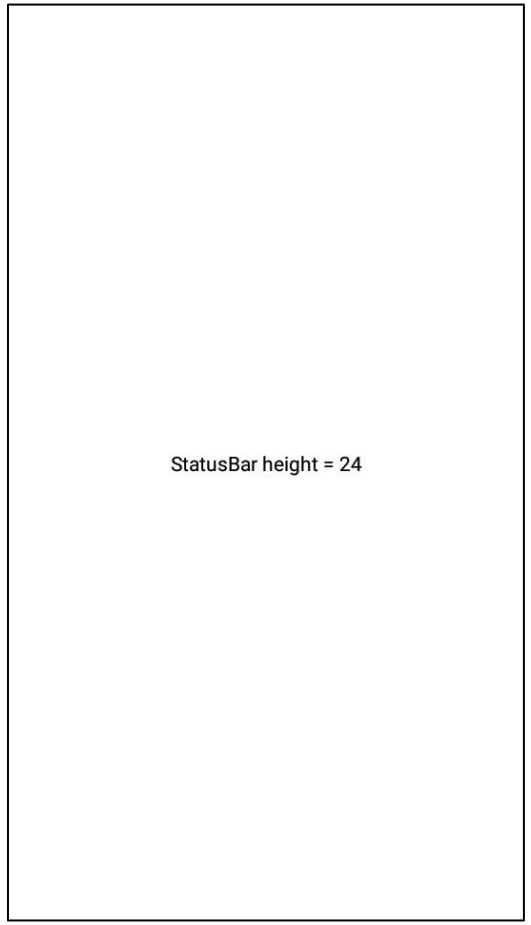
1. `import {StatusBar} from "react-native";`

2. 範例:

```
08:     <StatusBar
09:         hidden={false}
10:         barStyle="dark-content"
11:         backgroundColor="green"
12:         />
```

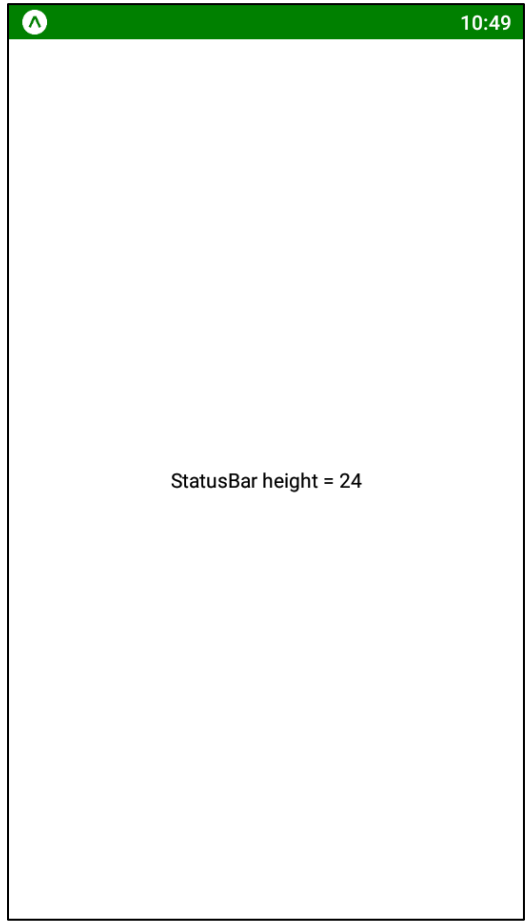
StatusBar

```
File: App-bar.js
03: export default function App() {
04:   return (
05:     <View style={styles.container}>
06:
07:       <Text>StatusBar height = {StatusBar.currentHeight}</Text>
08:       <StatusBar
09:             hidden={true}
10:             barStyle="dark-content"
11:             backgroundColor="green"
12:             />
13:     </View>
14:   );
15: }
16:
```



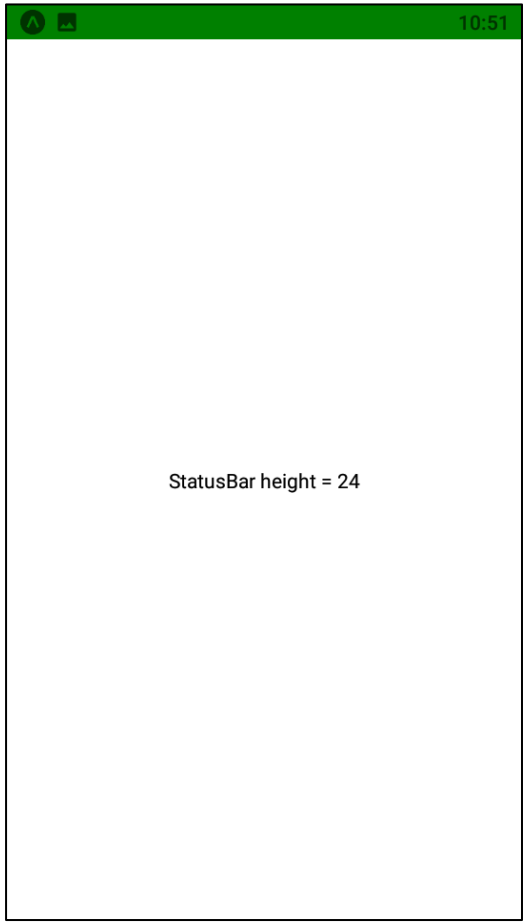
StatusBar

```
File: App-bar.js
03: export default function App() {
04:   return (
05:     <View style={styles.container}>
06:
07:       <Text>StatusBar height = {StatusBar.currentHeight}</Text>
08:       <StatusBar
09:             hidden={false}
10:             barStyle=“light-content"
11:             backgroundColor="green"
12:             />
13:     </View>
14:   );
15: }
16:
```



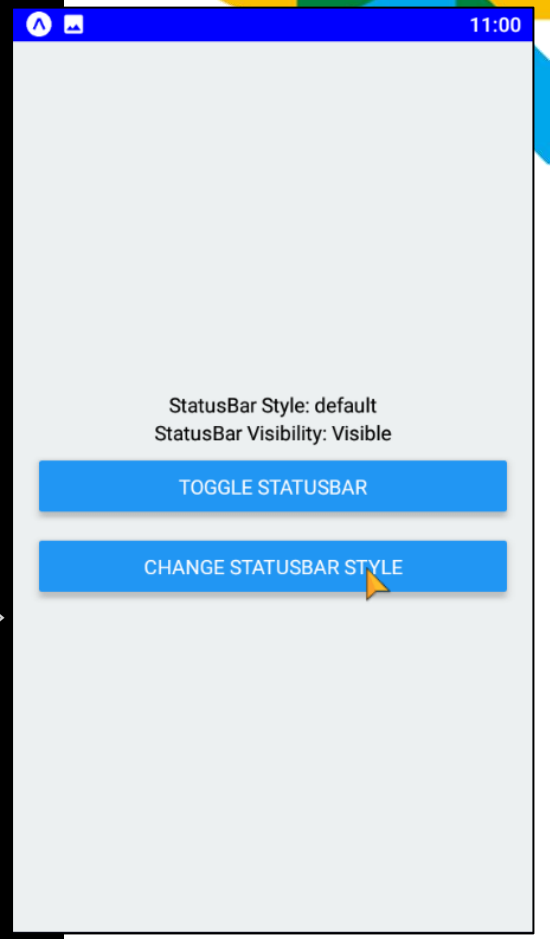
StatusBar

```
File: App-bar.js
03: export default function App() {
04:   return (
05:     <View style={styles.container}>
06:
07:       <Text>StatusBar height = {StatusBar.currentHeight}</Text>
08:       <StatusBar
09:             hidden={false}
10:             barStyle="dark-content"
11:             backgroundColor="green"
12:             />
13:     </View>
14:   );
15: }
16:
```



File: App-statusbar.js

```
06: const App = () => {
07:   const styleTypes = ['default','dark-content', 'light-content'];
08:   const [visibleStatusBar, setVisibleStatusBar] = useState(false);
09:   const [styleStatusBar, setStyleStatusBar] = useState(styleTypes[0]);
10:
11:   const changeVisibilityStatusBar = () => {
12:     setVisibleStatusBar(!visibleStatusBar);
13:   };
14:
15:   const changeStyleStatusBar = () => {
16:     const styleId = styleTypes.indexOf(styleStatusBar) + 1;
17:
18:     if(styleId === styleTypes.length){
19:       return setStyleStatusBar(styleTypes[0]);
20:     }
21:     return setStyleStatusBar(styleTypes[styleId]);
22:   };
23:
24:   return (
25:     <View style={styles.container}>
26:       <View>
27:         <Text style={styles.textStyle}>StatusBar Style: {styleStatusBar}</Text>
28:         <Text style={styles.textStyle}>StatusBar Visibility: {!visibleStatusBar ? 'Visible': 'Hidden'}</Text>
29:       </View>
30:       <StatusBar backgroundColor="blue" barStyle={styleStatusBar} />
31:       <View>
32:         <StatusBar hidden={visibleStatusBar} />
33:       </View>
34:       <View style={styles.buttonContainer}>
35:         <Button title="Toggle StatusBar" onPress={() => changeVisibilityStatusBar()} />
36:       </View>
37:       <View style={styles.buttonContainer}>
38:         <Button title="Change StatusBar Style" onPress={() => changeStyleStatusBar()} />
39:       </View>
40:     </View>
41:   );
```



練習一下

- 下載程式碼，嘗試修改一下 statusbar的設定，例如顏色...等。
  - App-bar.js
  - App-statusbar.js

useEffect hook

useEffect hook

- 當我們希望元件在渲染後，接著執行某個任務時，可以使用 useEffect hook 來達到功能。
- effect 指的是 **副作用（side-effect）** 的意思，在 React 中會把畫面渲染後和 React 本身無關而需要執行的動作稱做「副作用」，這些動作像是「發送 API 請求資料」、「手動更改 DOM 畫面」等等。
- 副作用（side-effect）又簡稱為 effect，所以就使用 useEffect 這個詞。因為 useEffect 內帶入的函式主要就是要用來處理這些副作用，因此這些帶入 useEffect 內的函式也會被稱作 effect。

useEffect 基本用法

- 基本語法:
  - 匯入 library

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

  - useEffect() function

```
  useEffect(() => {
    // 需要執行的指令
  });
```

useEffect 範例

```
01: import './App.css';
02: import React from 'react';
03: import { useEffect } from 'react';
04:
05: function MyComponent(){
06:   console.log('creating function component');
07:
08:   useEffect(() => {
09:     console.log('useEffect excuted');
10:   });
11:
12:   return(
13:     <>
14:       {console.log('rendering')}
15:       <h1>Hello Effect</h1>
16:     </>
17:   );
18: }
19:
20: function App() {
21:   return (
22:     <div className="App">
23:       <MyComponent/>
24:     </div>
25:   );
26: }
27:
28: export default App;
```

console.log 順序

```
creating function component
rendering
useEffect excuted
```

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

useEffect 範例

App.js > App

```
1  import React, { useEffect } from 'react';
2  import { StyleSheet, Text, View, StatusBar } from 'react-native';
3
4  function MyComponent() {
5    console.log('creating function component');
6    useEffect(() => {
7      console.log('useEffect executed');
8    });
9
10   return (
11     <>
12       {console.log('rendering')}
13       <View style={styles.container}>
14         <Text style={styles.text}>Hello Effect</Text>
15         <StatusBar style="auto" />
16       </View>
17     </>
18   );
19 }
```

useEffect 範例

```
21  export default function App() {
22    return (
23      <View style={styles.appContainer}>
24        <MyComponent />
25      </View>
26    );
27  }
28
29  const styles = StyleSheet.create({
30    appContainer: {
31      flex: 1,
32      backgroundColor: '#f0f0f0', // Light backgr
33    },
34    container: {
35      flex: 1,
36      alignItems: 'center',
37      justifyContent: 'center',
38      backgroundColor: '#fff',
39    },
40    text: {
41      fontSize: 20,
42      color: 'black',
43    },
44  });
```

useEffect hook

- useEffect 有兩個參數

```
useEffect(() => {
  // 需要執行的指令
},[]);
```

  - 第一個是 Effect function，
  - 第二個是 depandancy array。根據不同 depandancy array決定何時要執行 Effect function。
    - dependency array 若為空，只會在第一次 rendering 時執行 useEffect。
    - dependency array 可以寫入要參照的變數
      - 當變數的值改變時，便會執行 useEffect。
      - Component rendering後，只有變數值改變，才會呼叫 useEffect。

Fetch

Fetch

- 很多行動應用都需要從遠端獲取資料或資源。
- 你可能需要給某個 REST API 發起 POST 請求以提交使用者資料，又或者可能需要從某個伺服器上獲取一些靜態內容。

- React Native 提供了和 web 標準一致的Fetch API，用於滿足使用者訪問網路的需求。

Making requests

- 要從任意網址獲取內容的話，只需簡單地將網址作為參數傳遞給 fetch 方法即可（fetch 這個詞本身也就是獲取的意思）：

```
fetch('https://mywebsite.com/mydata.json');
```

Making requests

- Fetch 的第二個參數，可以用來客製化 HTTP request 一些參數。
- 你可以指定 header 參數，或是指定使用 POST 方法，又或是提交資料等等：

JSON

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    firstParam: 'yourValue',
    secondParam: 'yourOtherValue'
  })
});
```

Form

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded'
  },
  body: 'key1=value1&key2=value2'
});
```

Fetch 基本格式

- Fetch 的基本格式如下

```
1: fetch(url, options)
2:     .then(response => response.json())
3:     .then(data => console.log(data))
4:     .catch(e => console.log("error",e))
```

處理伺服器的回應資料

- 網路請求天然是一種非同步作業。
- Fetch 方法會返回一個Promise，這種模式可以簡化非同步風格的代碼。

```
function getMoviesFromApiAsync() {
  return fetch(
    'https://facebook.github.io/react-native/movies.json'
  )
    .then((response) => response.json())
    .then((responseJson) => {
      return responseJson.movies;
    })
    .catch((error) => {
      console.error(error);
    });
}
```

Fetch 使用方法

1. 建立一個 fetch object

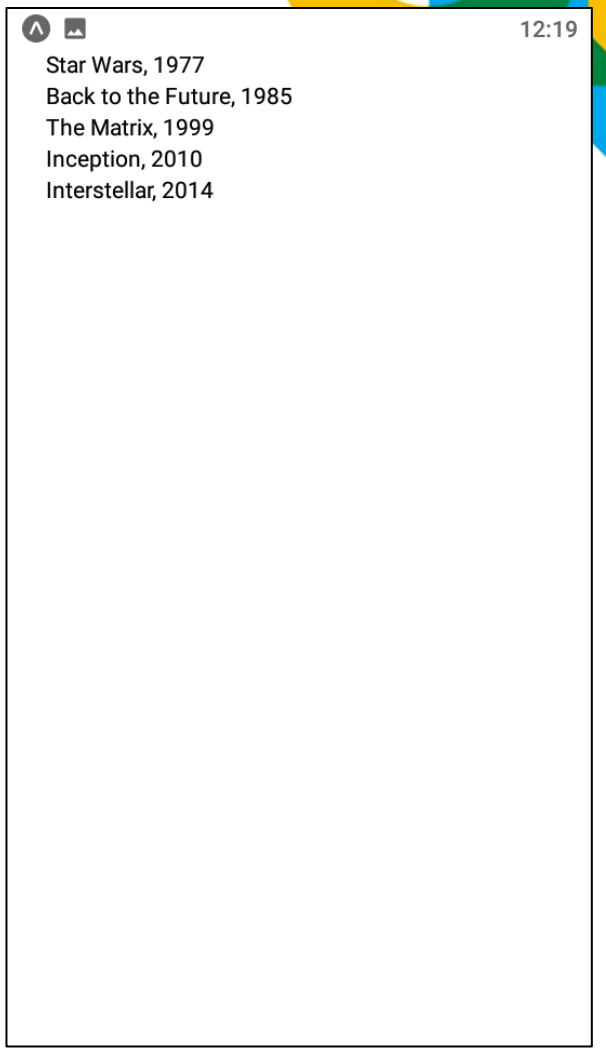
```
const getData=()=>{
    fetch('https://jsonplaceholder.typicode.com/users')
    .then(response=>response.json())
    .then(data=>console.log(data))
  }
```

2. 建立一個 useEffect

```
useEffect(() => {
    getData();
  },[])
```

File: App-fetch.js

```
01: import React, { useEffect, useState } from 'react';
02: import { ActivityIndicator, FlatList, Text, View } from 'react-native';
03:
04: export default App = () => {
05:   const [isLoading, setLoading] = useState(true);
06:   const [data, setData] = useState([]);
07:
08:   useEffect(() => {
09:     fetch('https://reactnative.dev/movies.json')
10:       .then((response) => response.json())
11:       .then((json) => setData(json.movies))
12:       .catch((error) => console.error(error))
13:       .finally(() => setLoading(false));
14:   }, []);
15:
16:   return (
17:     <View style={{ flex: 1, padding: 24 }}>
18:       {isLoading ? <ActivityIndicator/> : (
19:         <FlatList
20:           data={data}
21:           keyExtractor={({ id }, index) => id}
22:           renderItem={({ item }) => (
23:             <Text>{item.title}, {item.releaseYear}</Text>
24:           )}
25:         />
26:       )}
27:     </View>
28:   );
29: };
```



練習一下

- 資料 API
  - https://jsonplaceholder.typicode.com/users
- 參考資料
  - https://www.waldo.com/blog/react-native-fetch


```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
      "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
      }
```

Reference

- https://reactnative.dev/docs/0.69/statusbar
- https://zh-hans.reactjs.org/docs/hooks-effect.html
- https://ithelp.ithome.com.tw/articles/10224270
- https://reactnative.dev/docs/0.69/network
- https://www.waldo.com/blog/react-native-fetch

Async Storage

Async Storage

- An asynchronous, unencrypted, persistent, key-value storage API.
- 一個簡單、非同步、持久化的 Key-Value 儲存系统，它對於 App 來說是全域性的。可用來代替 LocalStorage。

- Async Storage can only store string data, so in order to store object data you need to serialize it first.
- For data that can be serialized to JSON you can use JSON.stringify() when saving the data and JSON.parse() when loading the data.

Async Storage

- 安裝套件
- npx expo install @react-native-async-storage/async-storage

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

- AsyncStorage 的資料只要是由以下兩個方法來做處理
  - setItem 儲存資料(AsyncStorage.setItem('key',value))
  - getItem 取得資料(AsyncStorage.getItem('key'))

儲存資料

Storing string value

資料的 key　　資料的值

```
1: const storeData = async (value) => {
2:   try {
3:     await AsyncStorage.setItem('@storage_Key', value)
4:   } catch (e) {
5:     // saving error
6:   }
7: }
```

Storing object value

```
2: const storeData = async (value) => {
3:     try {
4:       const jsonValue = JSON.stringify(value)
5:       await AsyncStorage.setItem('@storage_Key', jsonValue)
6:     } catch (e) {
7:       // saving error
8:     }
9: }
```

資料的 key　　資料的值

讀取資料

Reading string value

```
02: const getData = async () => {
03:     try {
04:       const value = await AsyncStorage.getItem('@storage_Key')
05:       if(value !== null) {
06:         // value previously stored
07:       }
08:     } catch(e) {
09:       // error reading value
10:     }
11: }
```

Reading object value

```
04: const getData = async () => {
05:     try {
06:       const jsonValue = await AsyncStorage.getItem('@storage_Key')
07:       return jsonValue != null ? JSON.parse(jsonValue) : null;
08:     } catch(e) {
09:       // error reading value
10:     }
11:   }
```

```
File: App-store.js
06: export default function App() {
07:
08:   const [token, setToken] = useState("data");
09:
10:   const storeData = async (value) => {
11:     try {
12:       await AsyncStorage.setItem('my-key', value);
13:     } catch (e) {
14:       // saving error
15:     }
16:   };
17:
18:   const getData = async () => {
19:     try {
20:       const value = await AsyncStorage.getItem('my-key');
21:       if (value !== null) {
22:         setToken(value) ;
23:       }
24:     } catch (e) {
25:       // error reading value
26:     }
27:   };
```

Known storage limits

- AsyncStorage for Android uses SQLite for storage backend. While it has its own size limits, Android system also have two known limits: total storage size and per-entry size limit.

1. Total storage size is capped at 6 MB by default. You can increase this size by specifying a new size using feature flag.
2. Per-entry is limited by a size of a WindowCursor, a buffer used to read data from SQLite. Currently it's size is around 2 MB. This means that the single item read at one time cannot be larger than 2 MB. There's no supported workaround from AsyncStorage. We suggest keeping your data lower than that, by chopping it down into many entries, instead of one massive entry. This is where multiGet and multiSet APIs can shine.

Reference

- https://react-native-async-storage.github.io/async-storage/docs/usage/

Q&A