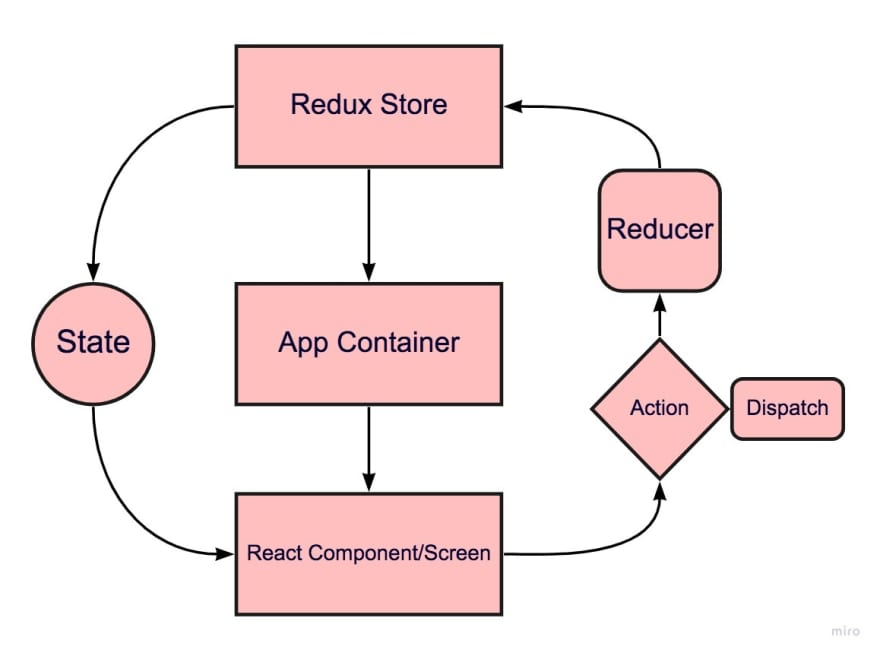
**Reference:**

**https://dev.to/theadultnoble\_6/how-to-use-redux-toolkit-with-react-native-2dm5**

**Redux:** Redux is a predictable state management library for JavaScript applications. It is widely used in React applications, but it can also be used with other libraries and frameworks.

Redux is a Javascript global state management library designed to act as a central store for managing application state. Redux helps you build apps that behave consistently across all environments by tracking all changes made to the app state.



Redux is based on three principles:

1. Single source of truth: The state of an entire application is stored in a single object tree within a single store.
2. Read-only state: The state tree is read-only, and the only way to change it is by dispatching an action.
3. Changes are made with pure functions: The logic that changes the state is written in pure functions called reducers.

Redux consists of three parts:

1. store: It is a single JavaScript object that holds the state of the application.
2. actions: Actions are plain JavaScript objects that describe the changes to the state
3. reducers: Reducers are pure functions that take the current state and an action as input and return the next state

Here are some advantages of using Redux:

1. **Centralized state management**: Redux provides a centralized location for storing the state of an application, which makes it easier to manage and debug.
2. **Predictable state updates**: Since the state of an application can only be updated through actions and reducers, the state changes become predictable and easy to reason about.
3. **Time-travel debugging**: Redux allows you to trace the state of an application at any point in time, which makes debugging easier.

Some scenarios in which Redux is appropriate include:

* When you have a large or complex application that requires centralized state management
* When you need to maintain a predictable state of the application
* When you need to debug complex state changes

Redux uses a combination of actions, reducers and the store to manage the application state.

* **The State**

Very lightly, the application state is all the information that the application uses and/or modifies. Having a centralized state and a predictable way of modifying it can enable applications to be coherent with the information that they show to the users. This is particularly useful for large single page applications.

* **Actions and reducers**

Actions and reducers together modify the state. Precisely, actions determine what is being modified and where. Reducers specify how it is being modified. By analyzing the layout at the beginning of the blog post, we will need three actions: INCREMENT, DECREMENT, and CHANGE\_BY\_AMOUNT. Actions are objects with a type and a payload attribute. The type is the action's identifier, and the payload is all the information that the reducer will need to modify the state. The first two actions will only decrement or increment the counter amount by 1, so we only need to define two objects with the attribute type, and the reducer will handle the rest. The third action must have a payload to specify how much we want to increment or decrement the counter. Declare your actions in a separate file called Actions:

### The Store

The next step is to create the store. The store is the object where the state is saved. It is common practice to create the store and export it in a separate file.

## When to Use Redux?

The app you built is ideally too basic to use a state manager like Redux. However, this tutorial aims to introduce Redux Toolkit in the most basic way possible.

So when should you use Redux?

* When there's a considerable amount of data changing over time
* When you need to track state changes
* When dealing with deeply nested components, and passing state becomes problematic
* When multiple components require access to the same piece of state

# Creating a global storage for react navigation using redux toolkit api

Run the following command to install redux toolkit api:

npm install react-redux

npm install @reduxjs/toolkit

To install the react navigation, use the following commands

npm install @react-navigation/native --save

npm install @react-navigation/native-stack --save

npm install react-native-screens react-native-safe-area-context --save

# Simple Counter Example with Redux Toolkit

Create the following files:

**redux/store.js**

import { configureStore } from '@reduxjs/toolkit'

import counterReducer from '../counterSlice'

export const store = configureStore({

  reducer: {

    counter: counterReducer,

  },

})

**counterSlice.js**

import { createSlice } from '@reduxjs/toolkit'

const initialState = {

  value: 0,

}

export const counterSlice = createSlice({

  name: 'counter',

  initialState,

  reducers: {

    increment: (state) => {

      // Redux Toolkit allows us to write "mutating" logic in reducers. It

      // doesn't actually mutate the state because it uses the Immer library,

      // which detects changes to a "draft state" and produces a brand new

      // immutable state based off those changes

      state.value += 1

    },

    decrement: (state) => {

      state.value -= 1

    },

    incrementByAmount: (state, action) => {

      state.value += action.payload

    },

  },

})

// Action creators are generated for each case reducer function

export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer

**Counter.js**

import React from 'react'

import { View, Button, Text } from 'react-native'

import { useSelector, useDispatch } from 'react-redux'

import { decrement, increment } from './counterSlice'

import { NavigationContainer } from '@react-navigation/native';

import { createNativeStackNavigator } from '@react-navigation/native-stack';

export default function Counter({navigation}) {

  const count = useSelector((state) => state.counter.value)

  const dispatch = useDispatch()

  return (

      <View>

        <Button

          title="Increment value"

          onPress={() => dispatch(increment())}

        />

        <Text>{count}</Text>

        <Button

          title="Decrement value"

          onPress={() => dispatch(decrement())}

        />

<Button

title = "Go to display screen"

onPress={() => navigation.navigate('Display')}

/>

      </View>

  )

}

**Display.js**

import React from 'react'

import { Text } from 'react-native'

import { useSelector } from 'react-redux'

export default function DisplayScreen() {

    const count = useSelector((state) => state.counter.value)

    return(

        <Text>{count}</Text>

    )

}

**App.js**

import React from 'react';

import { NavigationContainer } from '@react-navigation/native';

import { createNativeStackNavigator } from '@react-navigation/native-stack';

import { store } from './redux/store';

import { Provider } from 'react-redux';

import  Counter  from './Counter';

import  DisplayScreen from './Display';

const Stack = createNativeStackNavigator();

const App = () => {

  return (

    <Provider store={store}>

    <NavigationContainer>

      <Stack.Navigator initialRouteName="Counter">

        <Stack.Screen name="Counter" component={Counter} />

        <Stack.Screen name="Display" component={DisplayScreen} />

      </Stack.Navigator>

    </NavigationContainer>

    </Provider>

  );

};

export default App;

# Example of globally storing person attributes with Redux Toolkit

Create the following files:

**App.js**

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';

import HomeScreen from './screens/Home';

import DetailsScreen from './screens/Details';

import DisplayScreen from './screens/Display';

import { store } from './redux/store'

import { Provider } from 'react-redux'

const Stack = createNativeStackNavigator();

function App() {

  return (

    <Provider store={store}>

    <NavigationContainer>

      <Stack.Navigator initialRouteName="Home">

        <Stack.Screen name="Home" component={HomeScreen} />

        <Stack.Screen name="Details" component={DetailsScreen} />

        <Stack.Screen name="Display" component={DisplayScreen} />

      </Stack.Navigator>

    </NavigationContainer>

    </Provider>

  );

}

export default App;

**personSlice.js**

import { createSlice } from '@reduxjs/toolkit'

const initialState = { /\* initial  values here \*/   }

// here we are defining all functions to store the data in global storage

export const personSlice = createSlice({

  name: 'person',

  initialState,

  reducers: {

    addEmail: (state, action) => {

     return { ...state, email: action.payload };

    },

    addName: (state, action) => {

      return { ...state, uname: action.payload };

    },

    addAge: (state, action) => {

      return { ...state, age: action.payload };

    },

    addCity: (state, action) => {

      return { ...state, city: action.payload };

    },

  },

})

// Action creators are generated for each case reducer function

export const { addEmail, addName, addAge, addCity } = personSlice.actions;

export default personSlice.reducer;

**screens\Home.js**

import { useState } from 'react';

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

import { useDispatch } from 'react-redux'

import { addEmail, addName, addAge, addCity } from '../personSlice';

// create a class to send its values to global storage

class City {

  constructor(cityName, cityLocation)

  {

    this.cityName = cityName;

    this.cityLocation = cityLocation;

  }

}

export default function HomeScreen({ navigation }) {

  // dispatcher object is used to dispatch the action methods

  const dispatch = useDispatch();

  dispatch( addEmail("ali@gmail.com") );

  dispatch( addName("Ali Khan") );

  dispatch( addAge(45) );

  // to store a class object, it must be serialized first

  dispatch( addCity(JSON.stringify( new City('Karachi', 'Clifton') )));

 return (

      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>

        <Text>Home Screen</Text>

        <Button

          title="Go to Details Screen"

          onPress={() => { navigation.navigate('Details'); }}

        />

        <Button

          title="Go to Display Screen"

          onPress={() => { navigation.navigate('Display'); }}

        />

      </View>

    );

  }

**screens\Details.js**

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

import { useSelector } from 'react-redux'

export default function  DetailsScreen() {

    const { email, uname, age, city } = useSelector(state => state.person);

 // unserializing the city object (can't show without unserializing)

    var mycity = JSON.parse(city);

    return (

      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>

        <Text>Details Screen</Text>

        <Text>Email: {email}, Name: {uname}, Age: {age} </Text>

        <Text>City Name: {mycity.cityName}, City Location: {mycity.cityLocation}</Text>

      </View>

    );

  }

**screens\Display.js**

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

import { useSelector } from 'react-redux'

export default function  DisplayScreen() {

    const { email, uname, age } = useSelector(state => state.person);

    return (

      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>

        <Text>Display Screen</Text>

        <Text>Email: {email}, Name: {uname}, Age: {age} </Text>

      </View>

    );

  }

**redux\store.js**

import { configureStore } from '@reduxjs/toolkit';

import counterReducer from '../counterSlice';

import personReducer from '../personSlice';

// the store object is used to show the stored data in our app screens

export const store = configureStore({

  reducer: {

    counter: counterReducer,

    person: personReducer,

  },

})