CMSC 23: Mobile Computing

Week 06: State Management

Objectives

At the end of this Module, the students will be able to accomplish the following:

- understand the use of state management packages;
- Implement an app that uses state management.

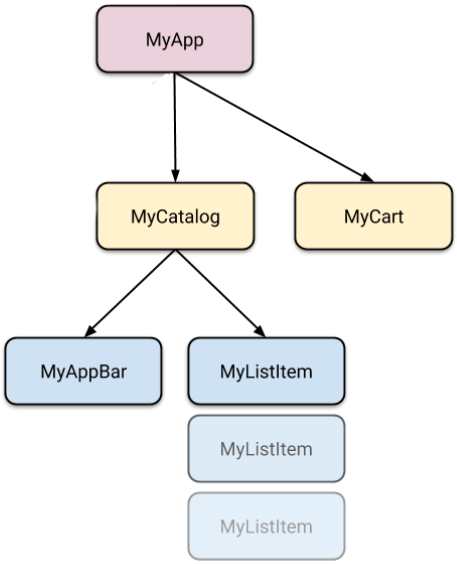
We have discussed that a state is a synchronous information about a widget. It is created when a widget is built and it changes through the widget's lifetime. As our application becomes more and more complex, we need to manage the different created states. This is where state management comes into play.

State Management

State Management is a vital part of Flutter development because it allows us to centralize every UI state and control the data flow in the application.

Provider

It is the most used state management package for Flutter beginners. It is easy to understand and it doesn't use much code. It also uses concepts that are applicable to every other approach.



To better understand the use of providers, we can use this example. In the figure to the left is a widget tree of an application. MyCatalog and MyCart are both on the same level. For example, we want to put the MyListItem inside the MyCart widget. To do that, we use the concept of "Lifting the State up".

In Flutter, if you want to change the UI, you have to rebuild it. A new widget is constructed every time its contents are changed.

Lifting the state up

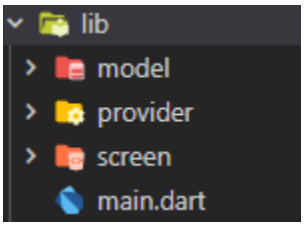
It is hard to imperatively change a widget from the outside and then rebuild it. In our example, whenever myCart rebuilds, the contents must not be replaced by the newly built cart. By lifting the state up, the mutable state lives high in the widget tree and is managed by passing properties down the tree.

Try this!

Step 1 - Install the provider package

```
flutter pub add provider
```

Step 2 - Organize your folders and create a model, provider, and screen folders.



Each folder will contain different sets of code. The first is on the Screen folder. Create a file named **MyCatalog.dart**

```
import 'package:flutter/material.dart';
import '../model/Item.dart';
import "package:provider/provider.dart";
import '../provider/shoppingcart_provider.dart';

class MyCatalog extends StatefulWidget {
  const MyCatalog({super.key});

  @override
  State<MyCatalog> createState() => _MyCatalogState();
}

class _MyCatalogState extends State<MyCatalog> {
  List<Item> productsCatalog = [
    Item("Shampoo", 10.00, 2),
    Item("Soap", 12, 3),
    Item("Toothpaste", 40, 3),
  ];
  @override
  Widget build(BuildContext context) {
```

```
      return Scaffold(
            appBar: AppBar(title: const Text("My Catalog")),
            body: ListView.builder(
                  itemBuilder: (BuildContext context, int index) {
                     return ListTile(
                        leading: const Icon(Icons.star),
                        title: Text(
                              "${productsCatalog[index].name} -
${productsCatalog[index].price}"),
                        trailing: TextButton(
                           child: const Text("Add"),
                          onPressed: () {
                           context
                                 .read<ShoppingCart>()
                                   .addItem(productsCatalog[index]);
                              ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                                 content: Text("${productsCatalog[index].name}
added!"),
                                 duration: const Duration(seconds: 1, milliseconds:
100),
                              ));
                          },
                        ),
                     );
                  },
                   itemCount: productsCatalog.length),
            floatingActionButton: FloatingActionButton(
               child: const Icon(Icons.shopping_cart),
              onPressed: () {
                  Navigator.pushNamed(context, "/cart");
              },
            ));
   }
}
```

MyCatalog.dart has a list of productItems which are shampoo, soap, and toothpaste. In the build function, it outputs each item using a list builder. Take note that each item in the list is an Item object. The add button uses context.read<ShoppingCart>() and context.read() so we'll need to make an instance of the shopping cart provider to inform its listeners.

MyCart.dart

```
import 'package:flutter/material.dart';
import '../model/Item.dart';
import "package:provider/provider.dart";
import "../provider/shoppingcart_provider.dart";

class MyCart extends StatelessWidget {
   const MyCart({super.key});

   @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("My Cart")),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          getItems(context),
          const Divider(height: 4, color: Colors.black),
          Flexible(
                child: Center(
                    child: Row(
                        mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                        children: [
                    ElevatedButton(
                        onPressed: () {
                          context.read<ShoppingCart>().removeAll();
                        },
                        child: const Text("Reset")),
                ]))),
          TextButton(
            child: const Text("Go back to Product Catalog"),
            onPressed: () {
              Navigator.pushNamed(context, "/products");
            },
          ),
        ],
      ),
    );
  }

  Widget getItems(BuildContext context) {
    List<Item> products = context.watch<ShoppingCart>().cart;
    String productname = "";
    return products.isEmpty
        ? const Text('No Items yet!')
        : Expanded(
            child: Column(
              children: [
                Flexible(
                    child: ListView.builder(
                  itemCount: products.length,
                  itemBuilder: (BuildContext context, int index) {
                    return ListTile(
                      leading: const Icon(Icons.food_bank),
                      title: Text(products[index].name),
                      trailing: IconButton(
                        icon: const Icon(Icons.delete),
                        onPressed: () {
```

```
                                productname = products[index].name;

context.read<ShoppingCart>().removeItem(productname);

                                if (products.isNotEmpty) {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                                    content: Text("$productname removed!"),
                                    duration:
                                        const Duration(seconds: 1,
milliseconds: 100),
                                ));
                            } else {
                                ScaffoldMessenger.of(context)
                                    .showSnackBar(const SnackBar(
                                    content: Text("Cart Empty!"),
                                    duration: Duration(seconds: 1,
milliseconds: 100),
                                ));
                            }
                        },
                    ),
                );
            },
        )),
        ],
    ));
  }
}
```

In **MyCart.dart**, in the getItems function, it uses context.watch<ShoppingCart>().cart to get the updated contents of the shopping cart list that was updated from the MyCatalog.dart. It also uses a conditional rendering where if the cart is empty, it will only output the text, “No Items Yet!” and the text button, go back to the product catalog.

In the model folder, create **Item.dart**

```
class Item {
  String name;
  double price;
  int quantity;

  Item(this.name, this.price, this.quantity);
}
```

Item.dart shows only the contents/attributes of object Item.

In the **main.dart**

```
import 'package:flutter/material.dart';
import 'package:state_management/screen/MyCart.dart';
import 'package:state_management/screen/MyCatalog.dart';
import 'package:provider/provider.dart';
import 'provider/shoppingcart_provider.dart';


void main() {
  runApp(MultiProvider(providers: [
      ChangeNotifierProvider(create: (context) => ShoppingCart()),
  ], child: const MyApp()));
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
       title: 'Flutter Demo',
      theme: ThemeData(
          primarySwatch: Colors.blue,
       ),
        initialRoute: "/",
       routes: {
          "/cart": (context) => const MyCart(),
          "/products": (context) => const MyCatalog(),
       },
       home: const MyCatalog(),
    );
  }
}
```

In the provider folder, **shoppingcart_provider.dart**

```
import 'package:flutter/material.dart';
import '../model/Item.dart';

class ShoppingCart with ChangeNotifier {
  final List<Item> _shoppingList = [];
  double cartTotal = 0;
  List<Item> get cart => _shoppingList;

  void addItem(Item item) {
    _shoppingList.add(item);
   cartTotal = cartTotal + item.price;
    notifyListeners();
  }
```

```
  void removeAll() {
    _shoppingList.clear();
    cartTotal = 0;
    notifyListeners();
  }

  void removeItem(String name) {
    for (int i = 0; i < _shoppingList.length; i++) {
      if (_shoppingList[i].name == name) {
        cartTotal = cartTotal - _shoppingList[i].price;
        _shoppingList.remove(_shoppingList[i]);
        break;
      }
    }
    notifyListeners();
  }
}
```

The shoppingcart_provider.dart contains the changenotifier. This contains all the functions that are to be done in the shopping cart, such as adding items, removing items, or removing all items.

ChangeNotifier

This allows the UI to rebuild and make the necessary updates on the variables. It also has the function notifyListeners().

notifyListeners()

This is used to call all the registered Listeners. This is called whenever the object changes to nifty any clients that the object may have changed.

context.watch() and context.read()

context.read<Model>() is used when we want to return without listening to its changes while context.watch<Model> is used when we want to listen to the changes on the model. In addition, context.read<Model>() internally returns Provider.of<Model>(context, listen:false), while context.watch<Model>() returns Provider.of<Model>(context).

Take note that whenever we use the context.watch and context.read, all of the widgets are rebuilt. This may cause some performance problems or when it is difficult to obtain a BuildContext. Remember that we need a BuildContext descendant of the provider to use .watch and .read

Due to the mentioned problem above, we can use Consumers.

Consumer

Consumer syntax is as follows. This is often used if you cannot access the context and you want to rebuild only a certain part of the screen.

```
Consumer<Model>(
        Builder:  (context,value,child){
            return
        }
);
```

In our code, we now will try to put a cart total using a consumer, and the cart total will only rebuild if the value of the cart total changed.

MyCart.dart

```
…
Widget computeCost() {
      return Consumer<ShoppingCart>(builder: (context, cart, child) {
            return Text("Total: ${cart.cartTotal}");
      });
  }
```

Then in the part of the builder of MyCart, put the call for the computeCost

```
…
children: [
              getItems(context),
            computeCost(),
              const Divider(height: 4, color: Colors.black),
             TextButton(
                child: const Text("Go back to Product Catalog"),
                onPressed: () {
                    Navigator.pushNamed(context, "/products");
                },
              ),
          ],
…
```


References:

- Biessek, A. (2019). *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2*. Packt Publishing Ltd.
- Iversen, J., & Eierman, M. (2014). *Learning mobile app development: A hands-on guide to building apps with iOS and Android*. Pearson Education.
- Panhale, M. (2016). *Beginning hybrid mobile application development*. New York: Apress.
- Payne, R. (2019). *Beginning App Development with Flutter: Create Cross-Platform Mobile Apps*. Apress.
- https://docs.flutter.dev/development/data-and-backend/state-mgmt/simple

Prepared by KLM Tan

- https://www.desuvit.com/state-management-in-flutter-a-comprehensive-guide/