Mobile Application Architecture

MVVM Architecture

Lecture and Demo

Jumail Bin Taliba
School of Computing, UTM

July 2020

Agenda

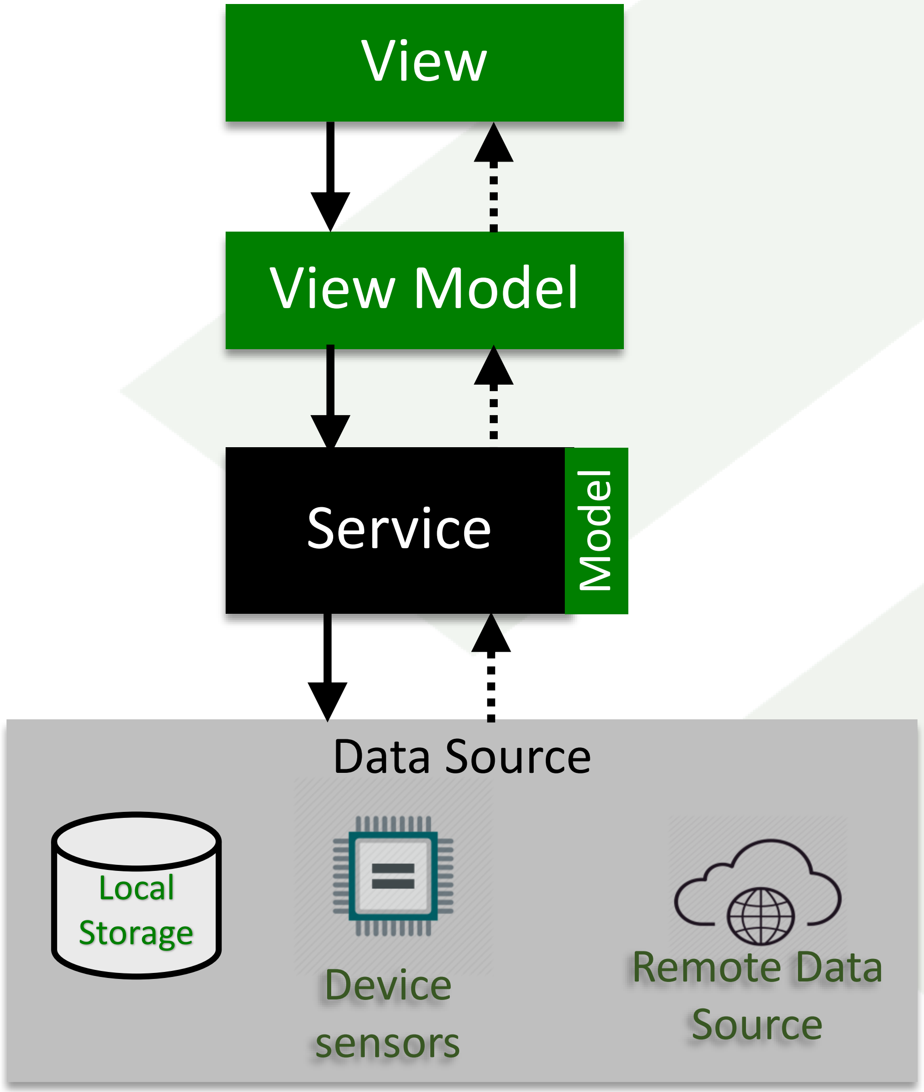
- Introduction to MVVM
- MVVM Components
- How Provider Relates to MVVM
- Setup MVVM Project
- Code Refactoring
- Discussion
- Summary

Introduction to MVVM

- MVVM stands for Model View Viewmodel
- Proposed by Microsoft for their WPF (a UI Framework for .NET).
- An architectural Pattern, other similar thing: MVC, MVP
  - Separation of concern – split code rather than putting them in a single place
  - For maintainability and extendibility, and easy for unit testing
  - Platform-agnostic

- Has clear separation of the UI (View) and application logic (Viewmodel and Model)
- Great for applications with interactive UI
- Has different variations and implementations
- Key principle: View Models synchronize Views and Models

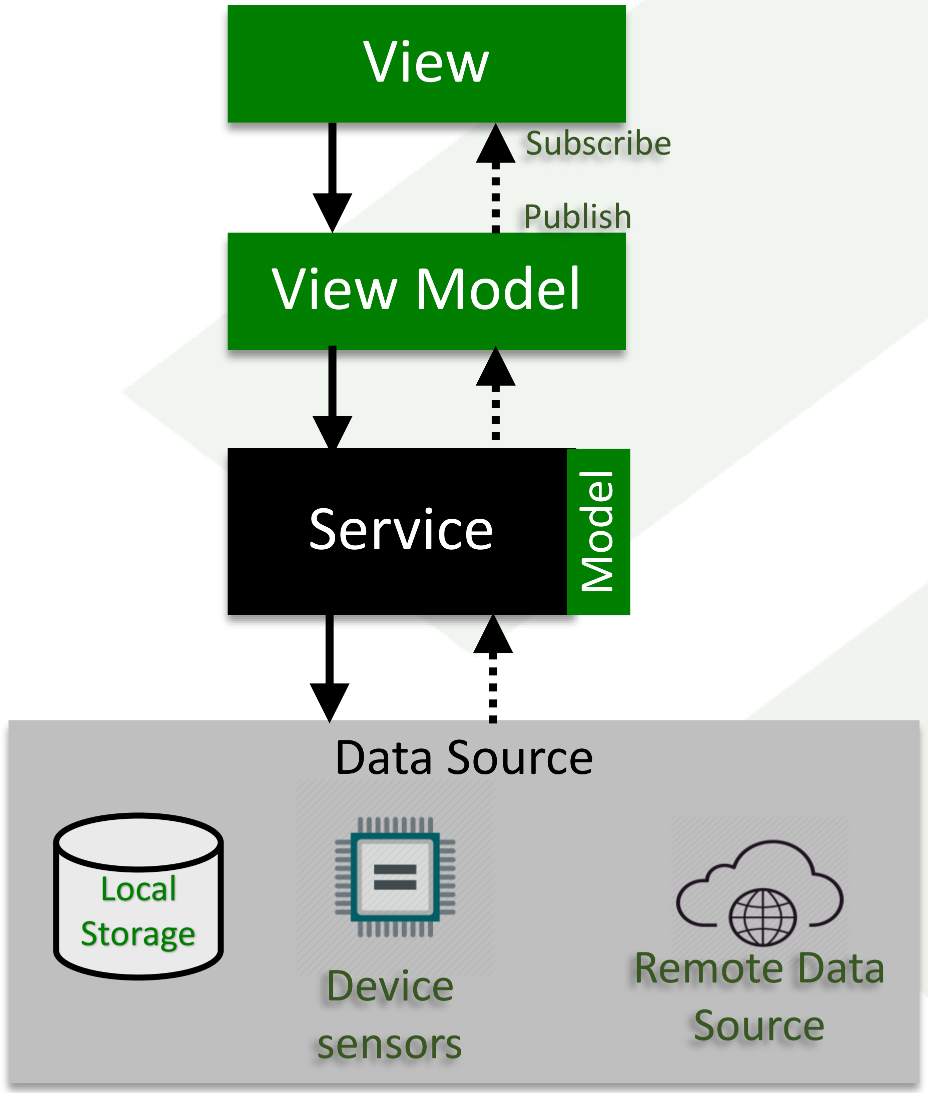
MVVM Components



View

- Handles what user sees and interacts with
- In Flutter, they are widgets
- Example: screens, buttons, app bar, list view, etc.
- View has reference to View Model *(as shown by the solid arrow line in the diagram)*

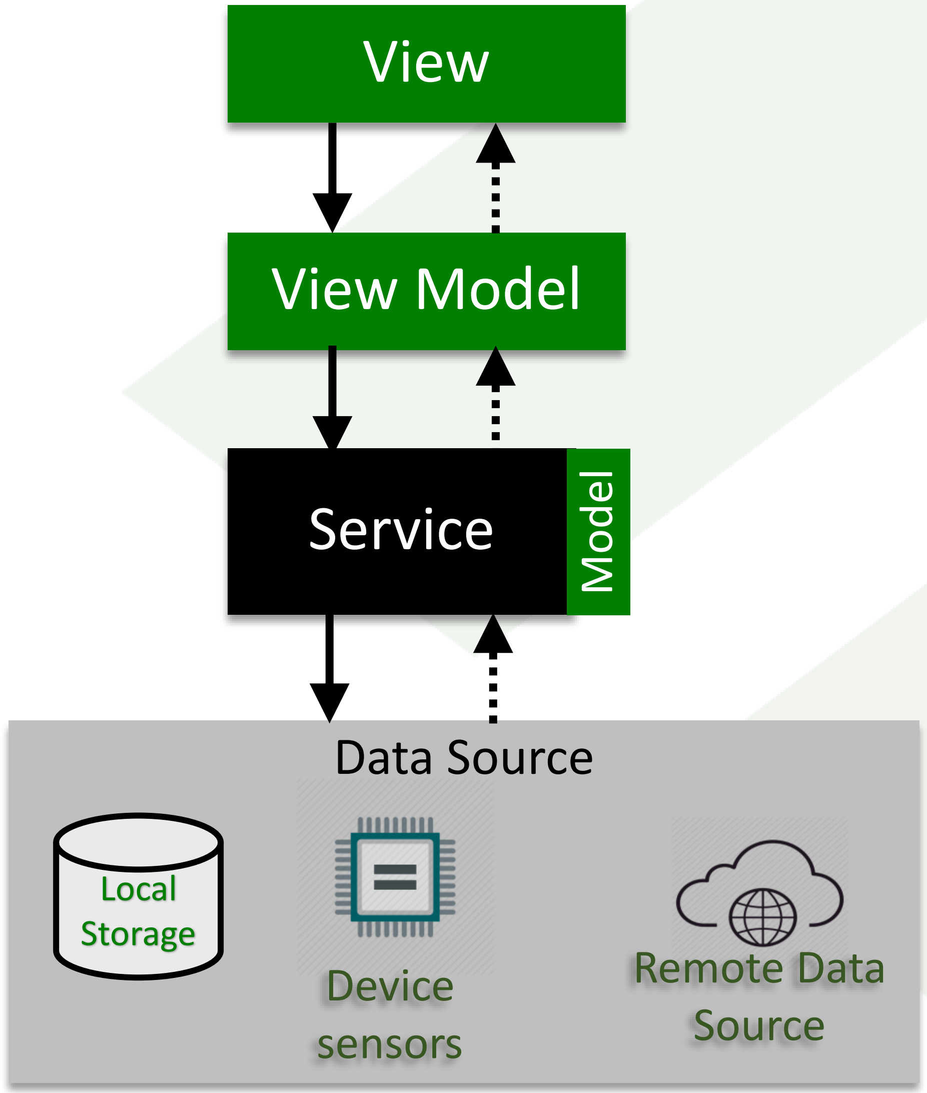
MVVM Components (2)



View Model

- The model of view.
  - View may display data differently from the data source, e.g. date with different format.
  - Viewmodel responsibles to format the data source to the form that the view requires.
- It synchronizes between UI and what is going on behind the scene
- It does not know about the view
- It only exposes its data to be observed by the View *(as shown by the dashed arrow line in the diagram)*
- It adopts the Publish-Subscribe design pattern

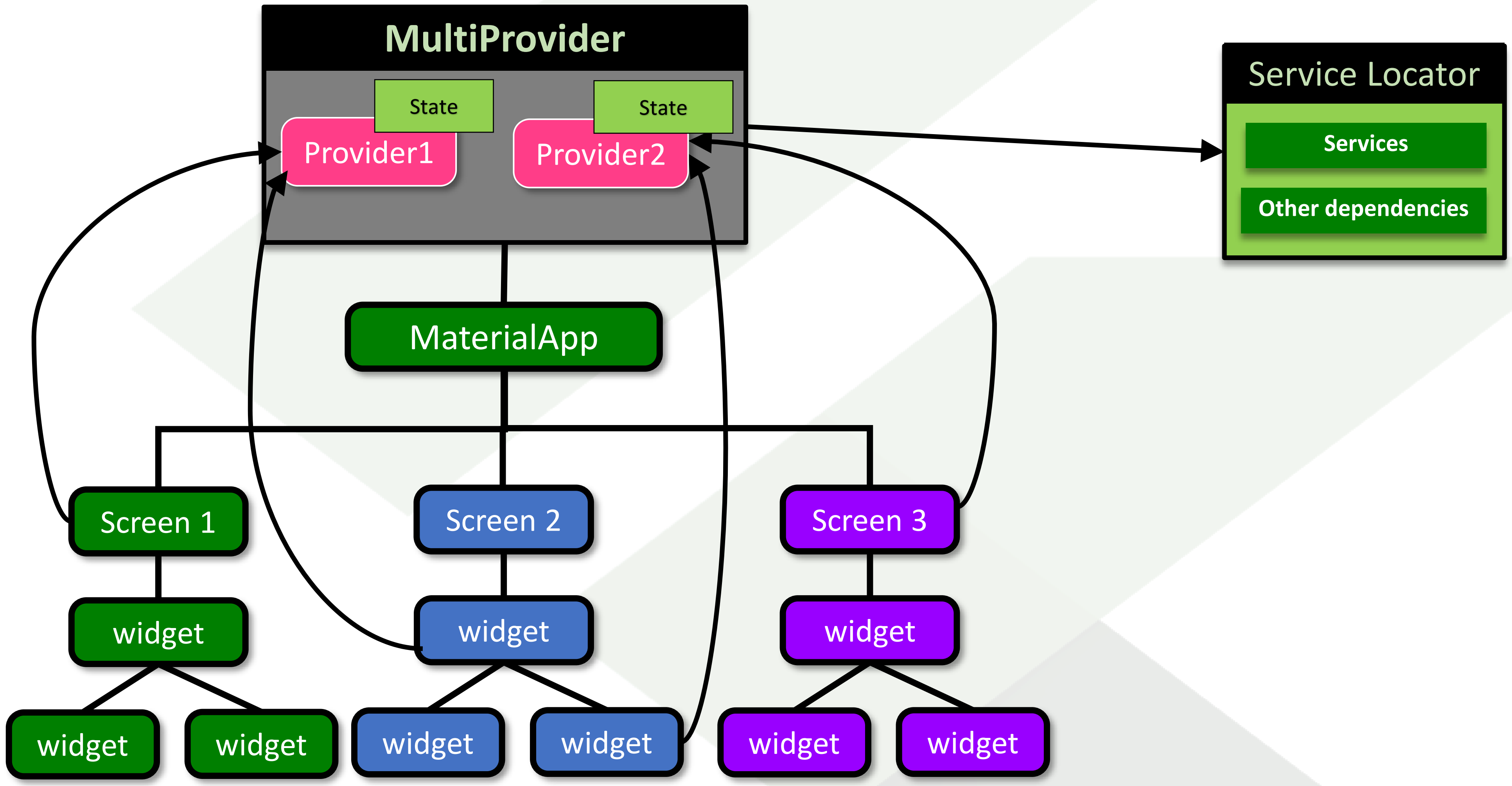
MVVM Components (3)



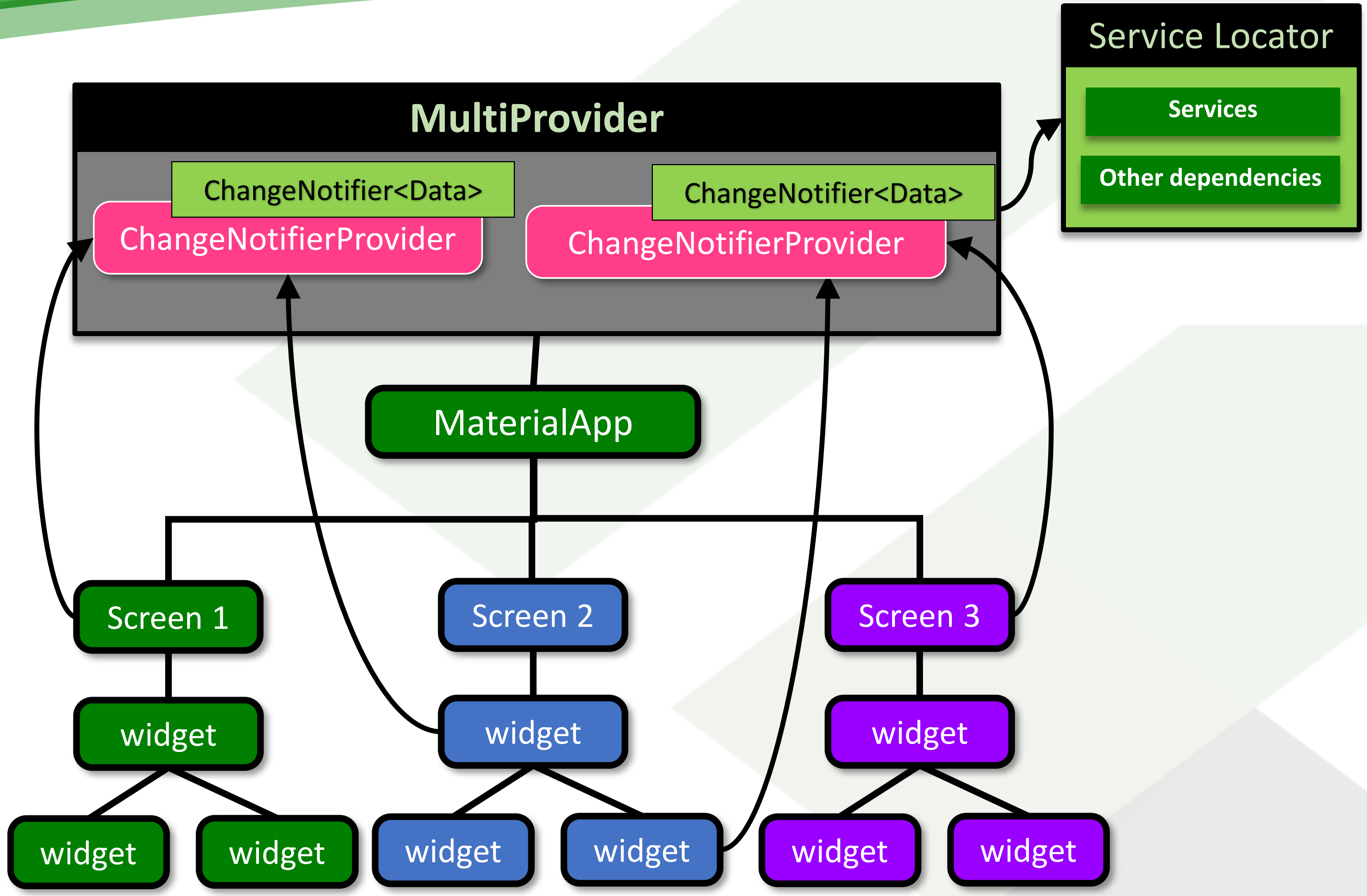
Service and Model

- Service gets data *(requested by View via the View Model)* from the data source
- Service does the actual work
- View Model only takes requests from View and forward them to service.
- Model is meant for representing data in a more convenient way

Using Providers - Revisit



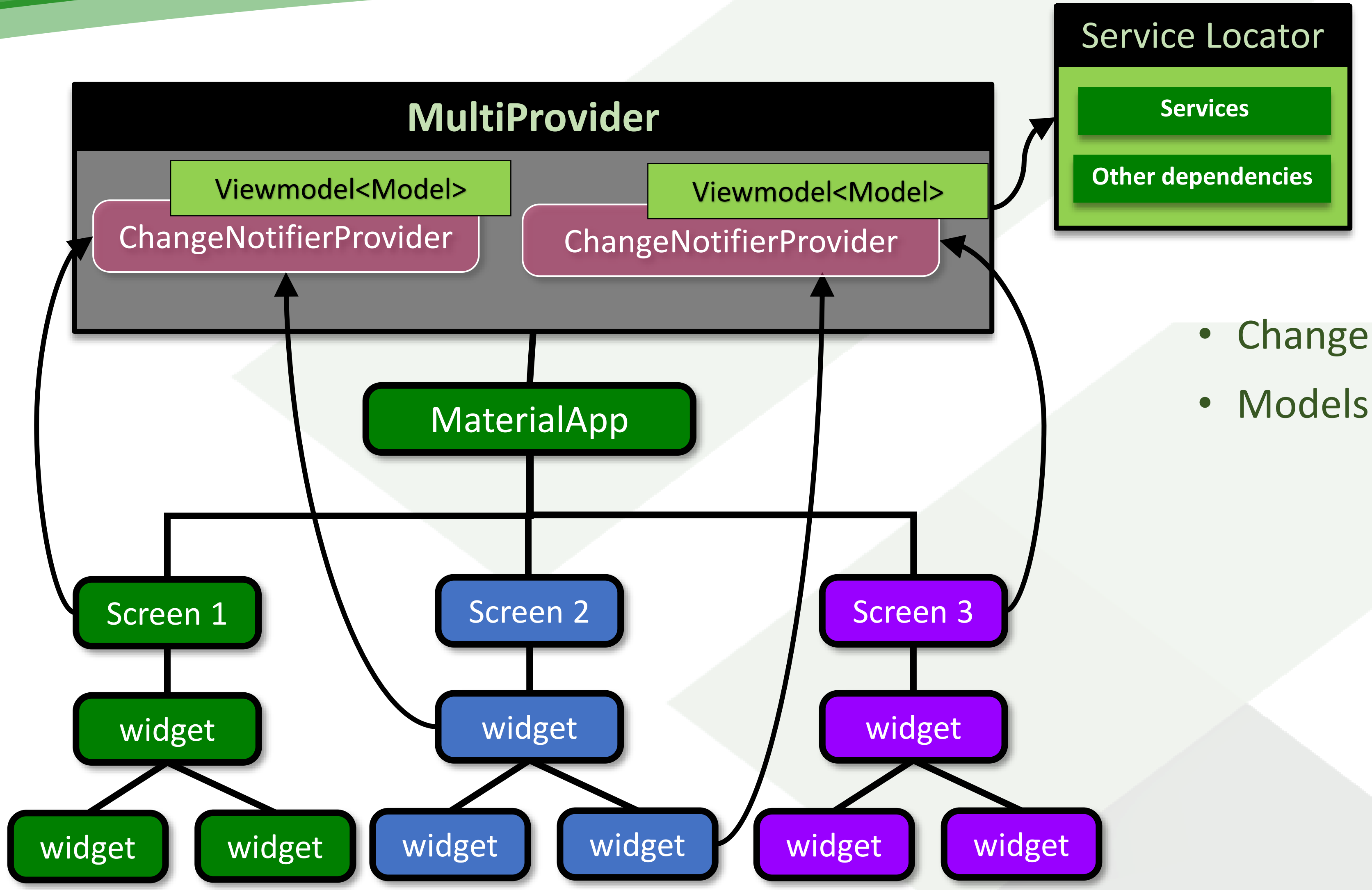
Implementing MVVM with Providers



Using ChangeNotifierProvider

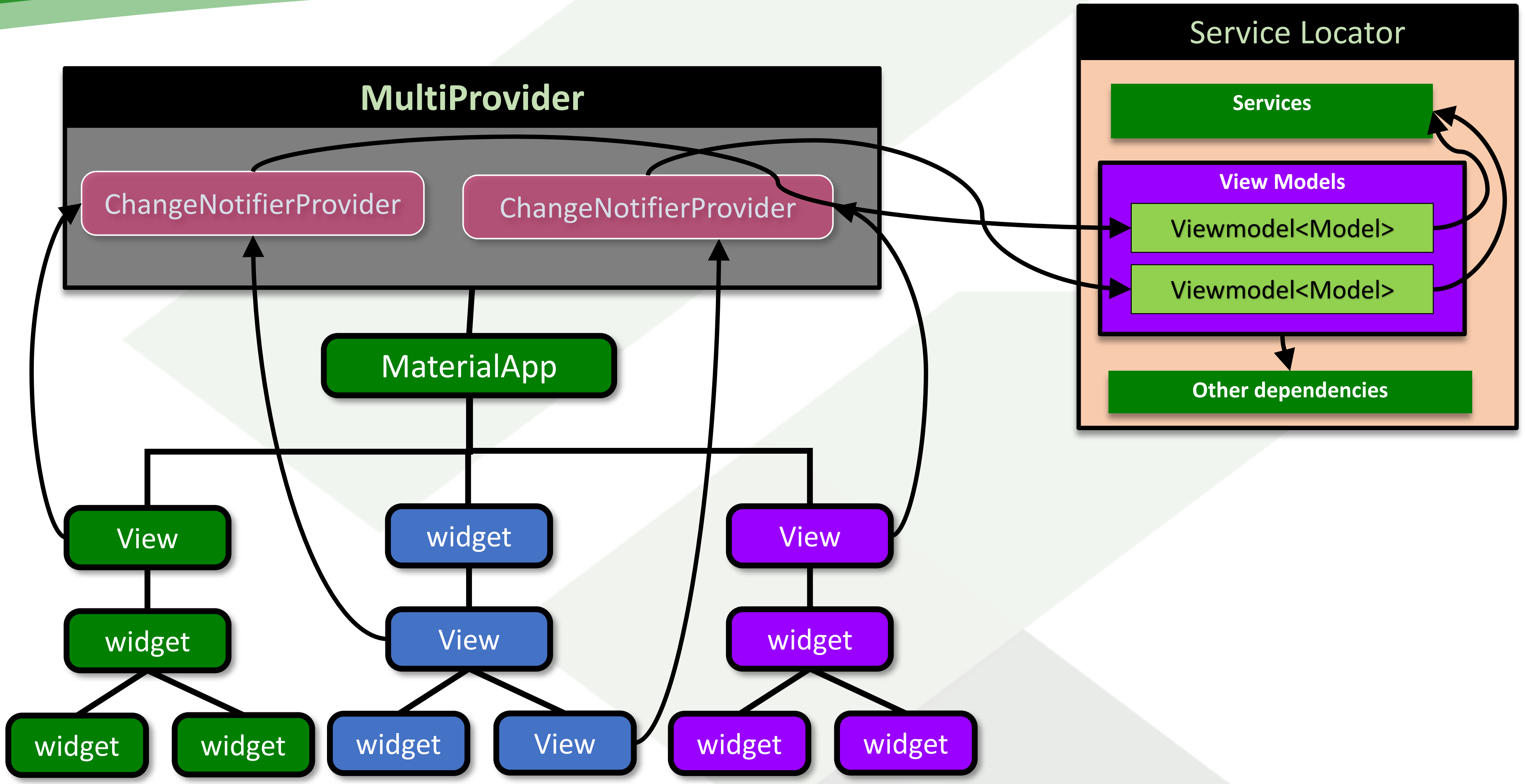
- to allow update from UI
- for state management

Implementing MVVM with Providers (2)

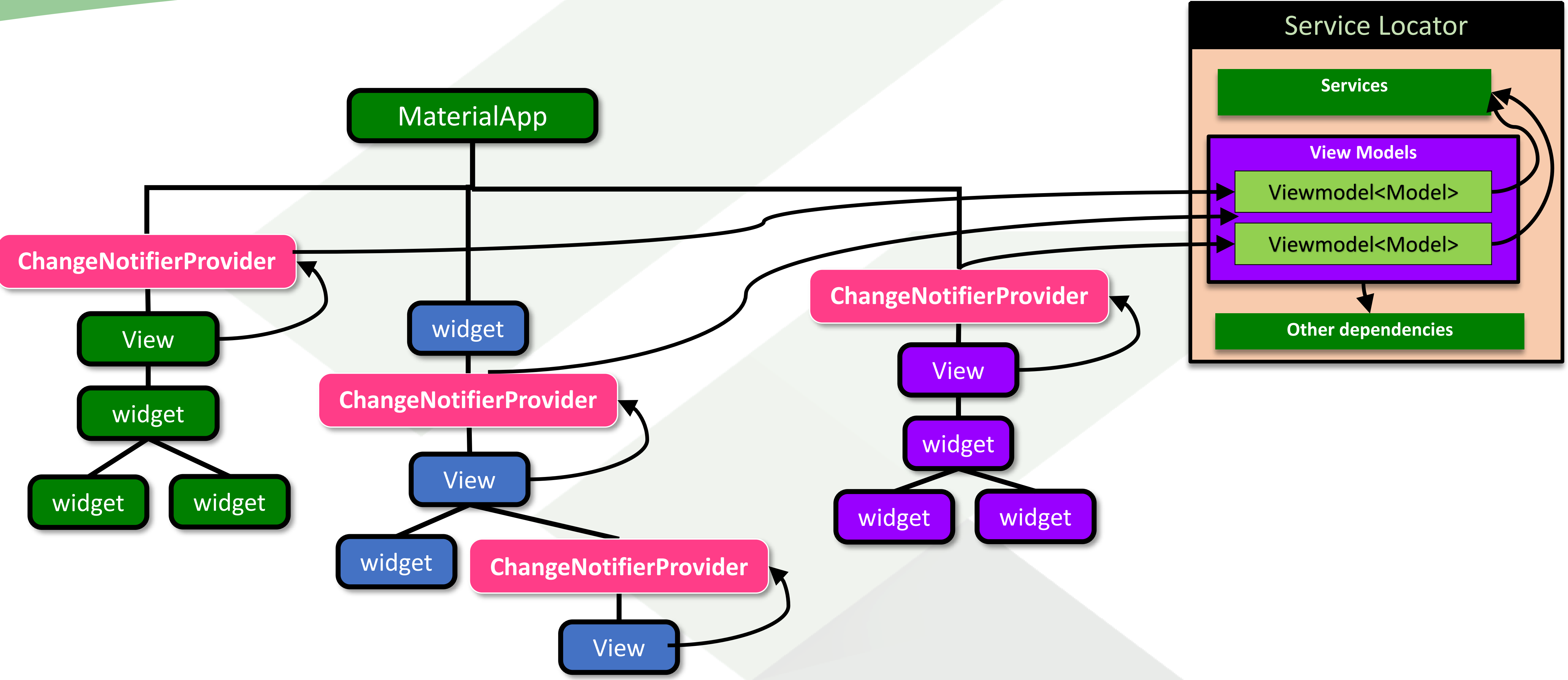


- ChangeNotifiers are View Models
- Models are held by ChangeNotifiers

Refactoring: Move View Models to Service Locator



Refactoring: Each Consumer Has its Own Provider



Demo

MVVM Architecture

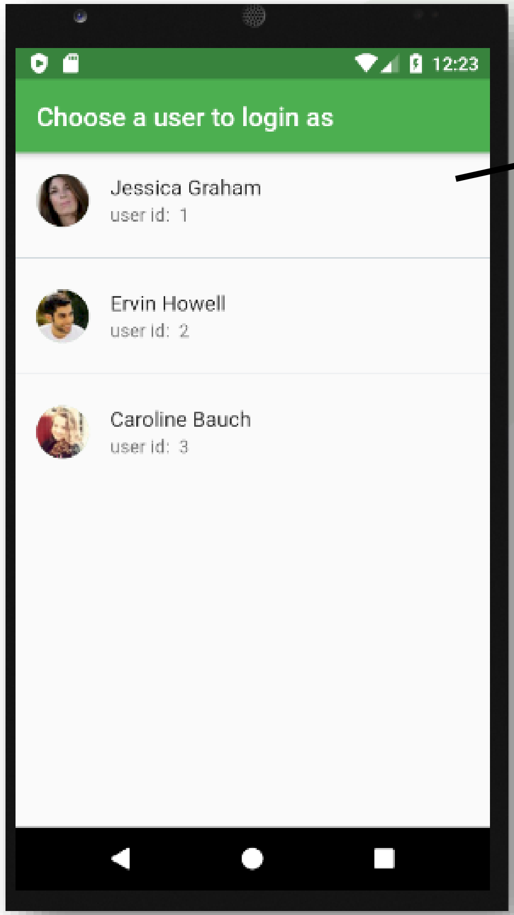
Source Code

https://github.com/jumail-utm/architecture_mvvm

About The Demo

What we are going to build

LoginScreen



12:23

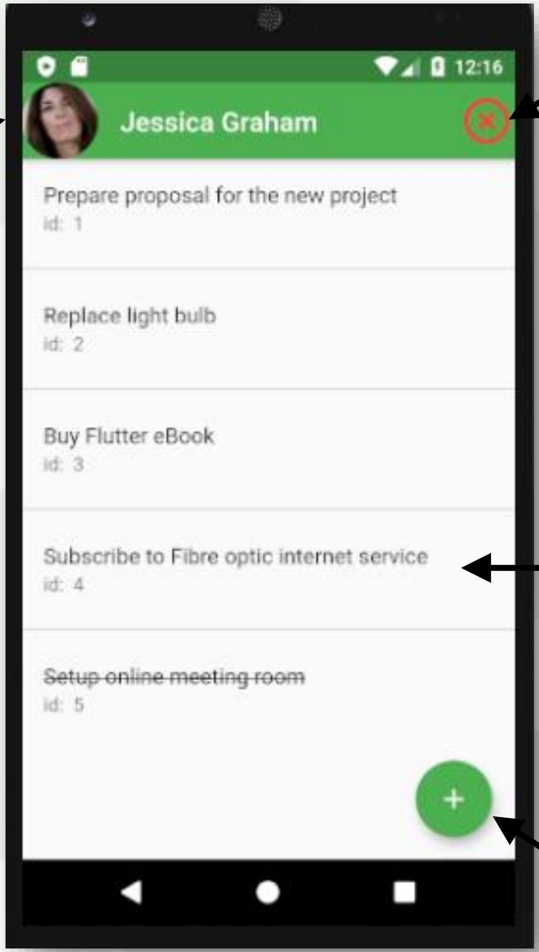
Choose a user to login as

Jessica Graham
user id: 1

Ervin Howell
user id: 2

Caroline Bauch
user id: 3

TodolistScreen



12:16

Jessica Graham

Prepare proposal for the new project
id: 1

Replace light bulb
id: 2

Buy Flutter eBook
id: 3

Subscribe to Fibre optic internet service
id: 4

~~Setup online meeting room~~
id: 5

Logout

The todo list displayed for the active user

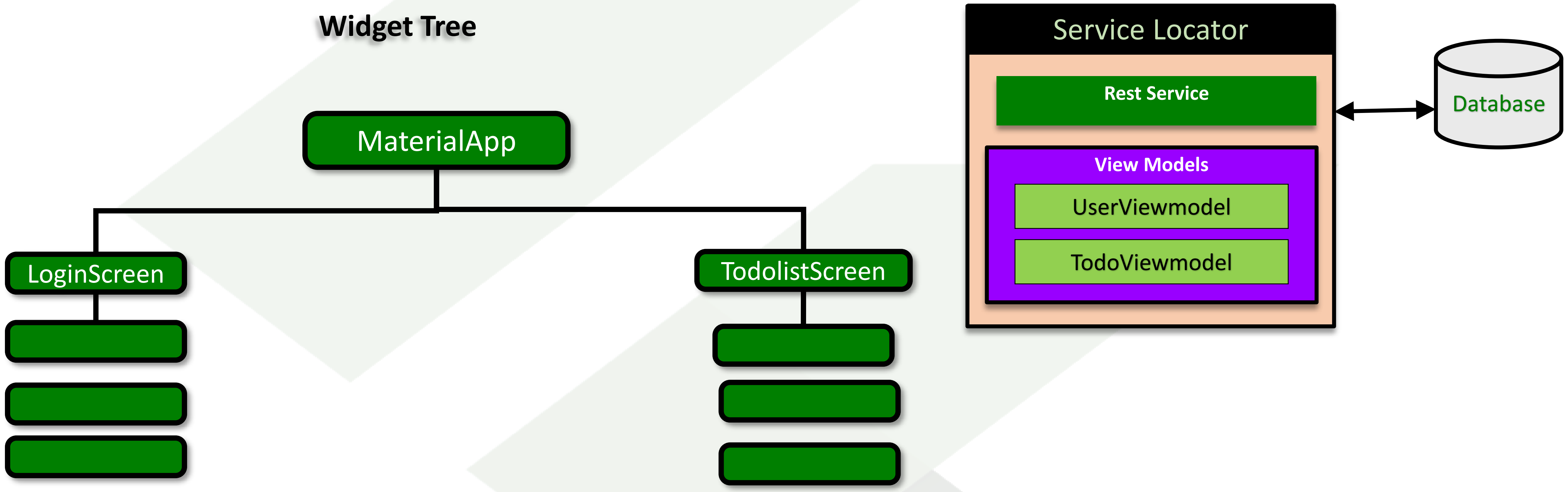
onTap: Toggle status

onLongPressed: Delete the todo item

Add a new todo item

About the Demo (2)

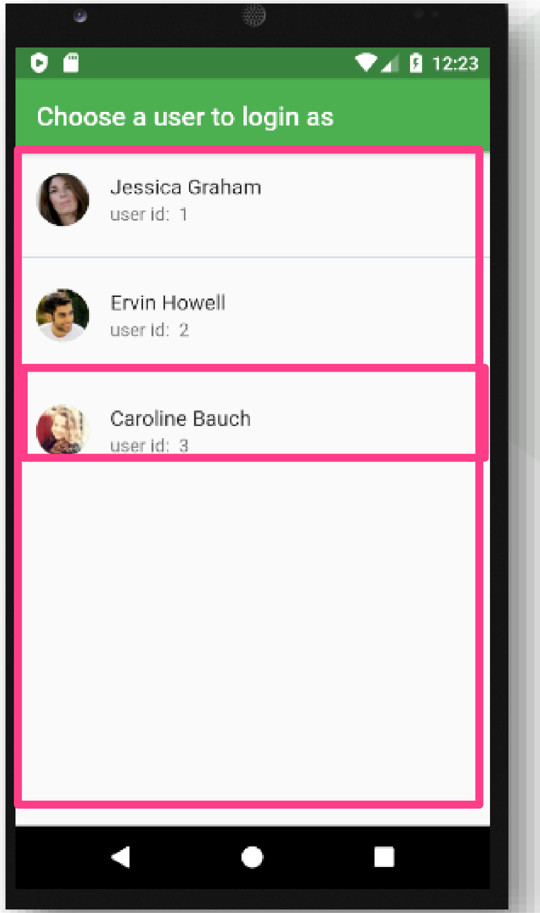
How we are going to build it

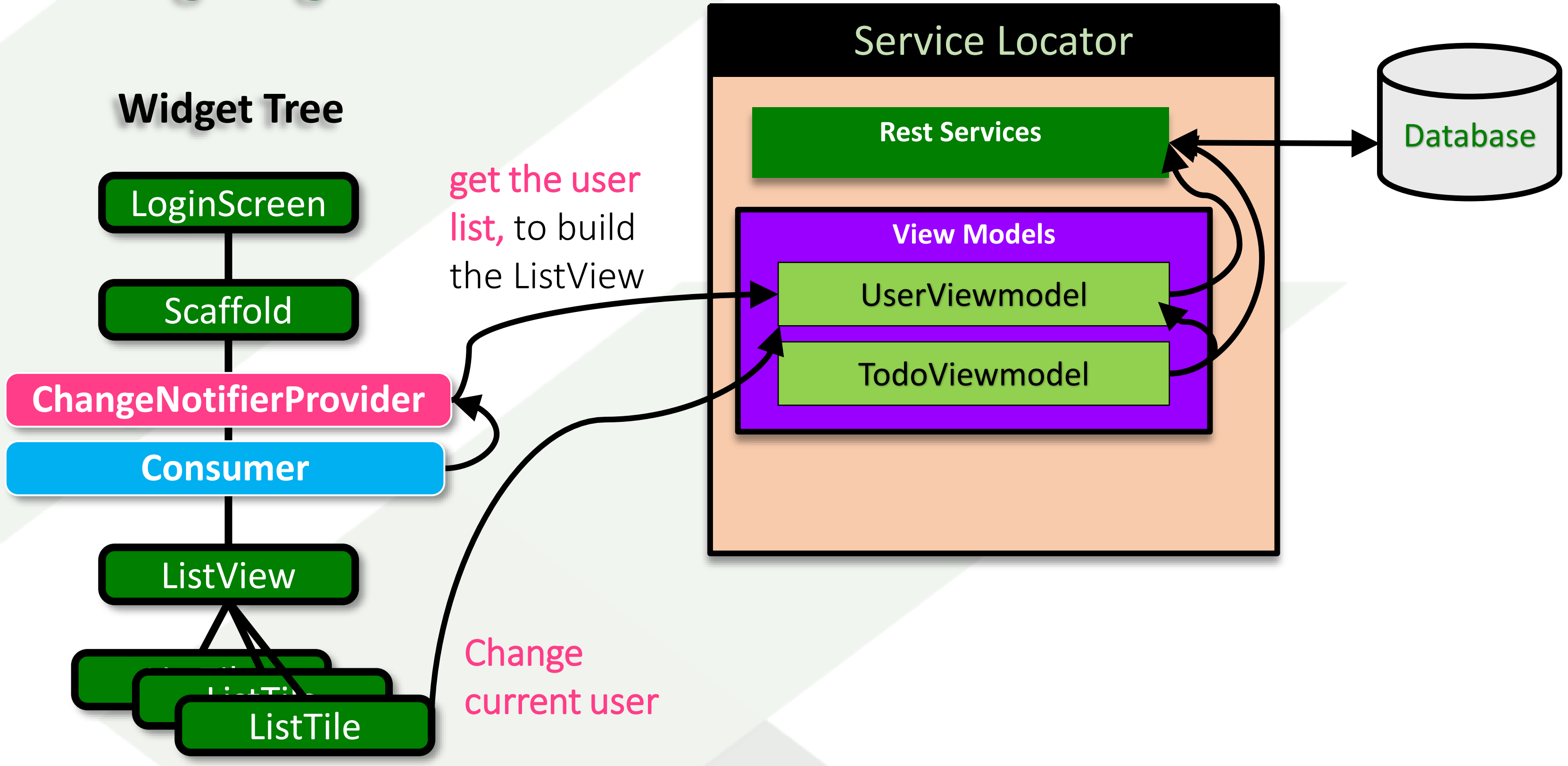


About the Demo (3)

How we are going to build it

LoginScreen

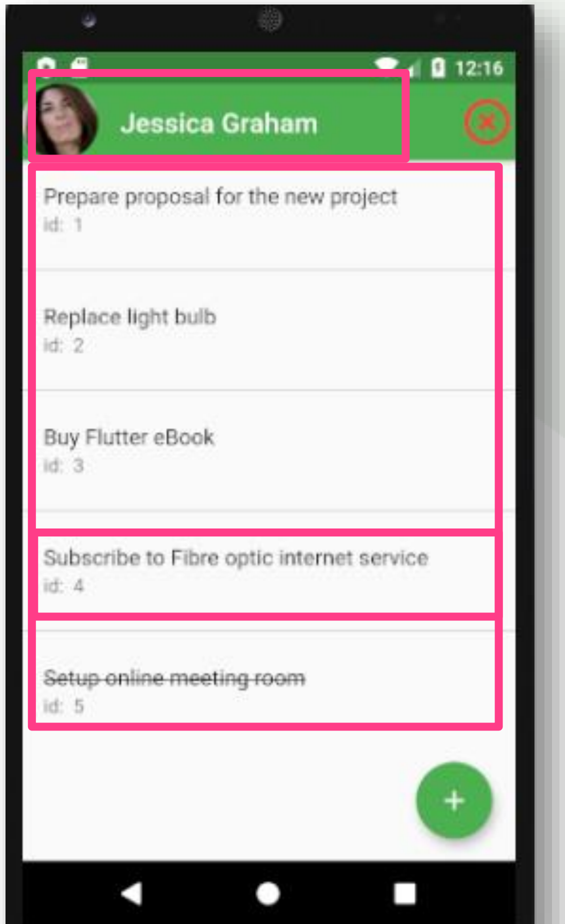




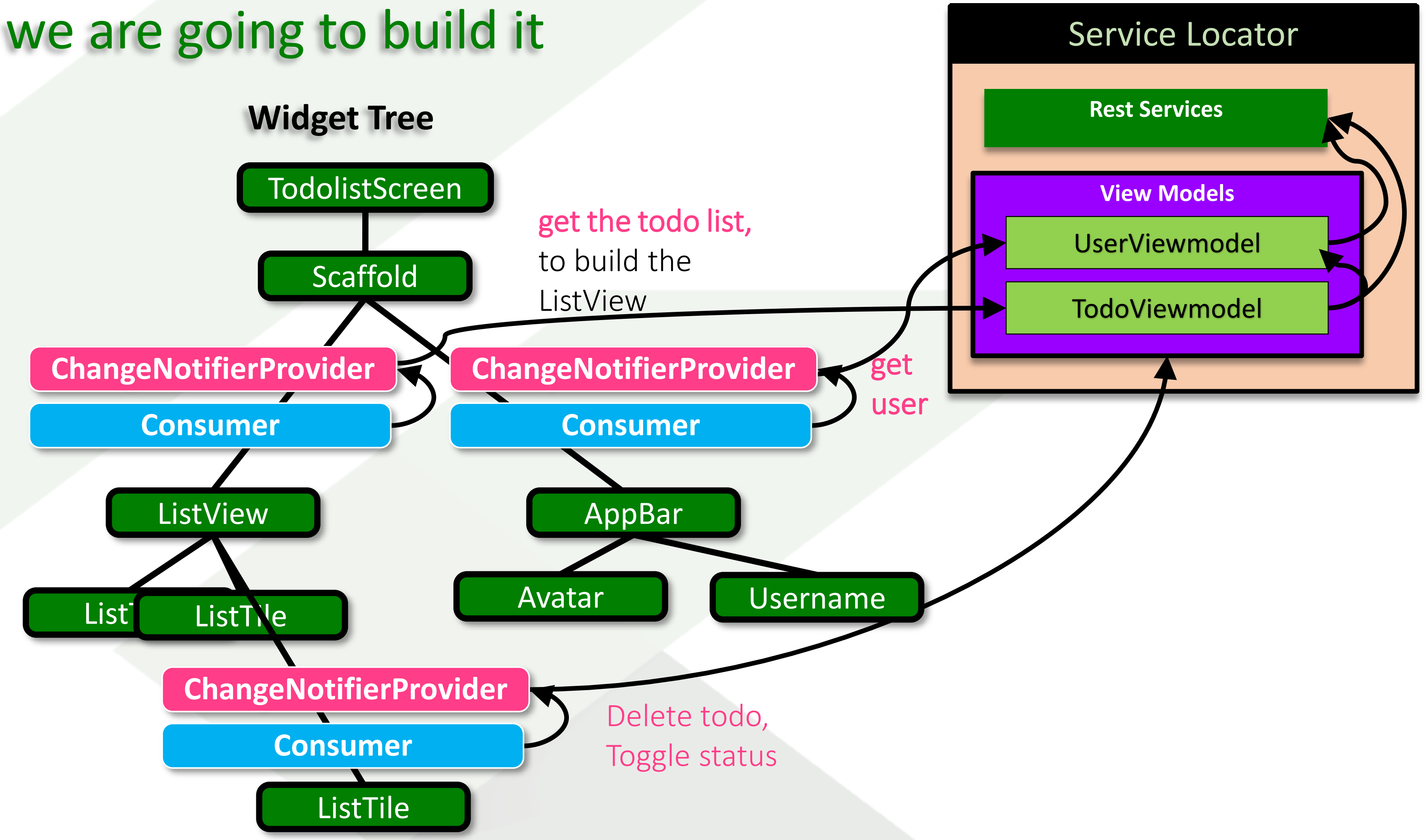
About the Demo (4)

How we are going to build it

TodolistScreen



12:16
Jessica Graham
Prepare proposal for the new project
id: 1
Replace light bulb
id: 2
Buy Flutter eBook
id: 3
Subscribe to Fibre optic internet service
id: 4
~~Setup online meeting room~~
id: 5

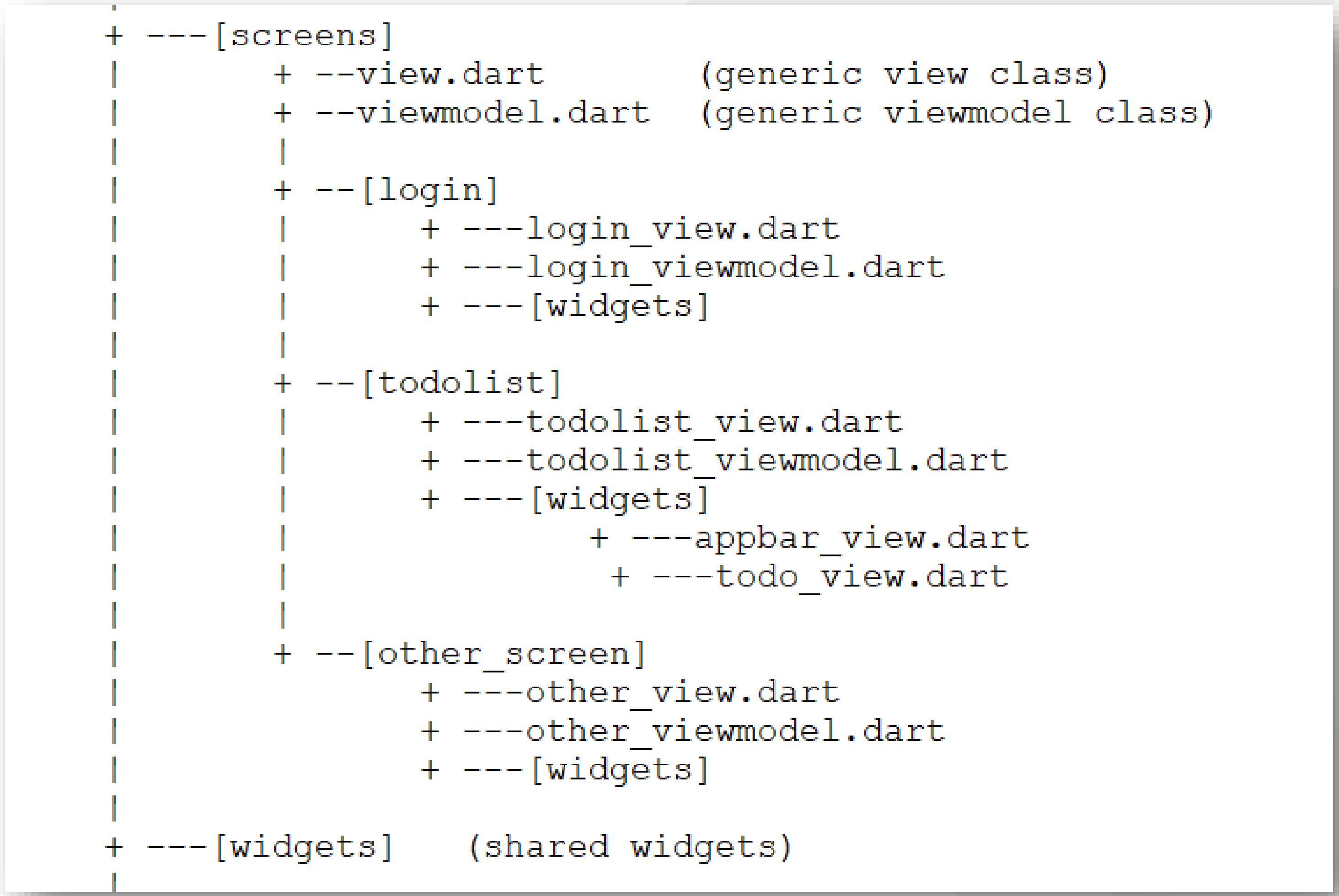


Project File Structure

```
[architecture_mvvm]
          |
          +---[lib]
          |       |
          |       + ---main.dart
          |       |
          |       + ---[app]
          |       |
          |       + ---[models]
          |       |
          |       + ---[screens]
          |       |
          |       + ---[widgets]
          |       |
          |       + ---[services]
```

```
-[lib]
    |
    + ---main.dart
    |
    + ---[app]
    |        + ---dependencies.dart   (service registry)
    |        + ---router.dart
    |        + ---other_app_wide_stuff.dart
    |
    + ---[models]
    |        + ---todo.dart
    |        + ---user.dart
    |
```

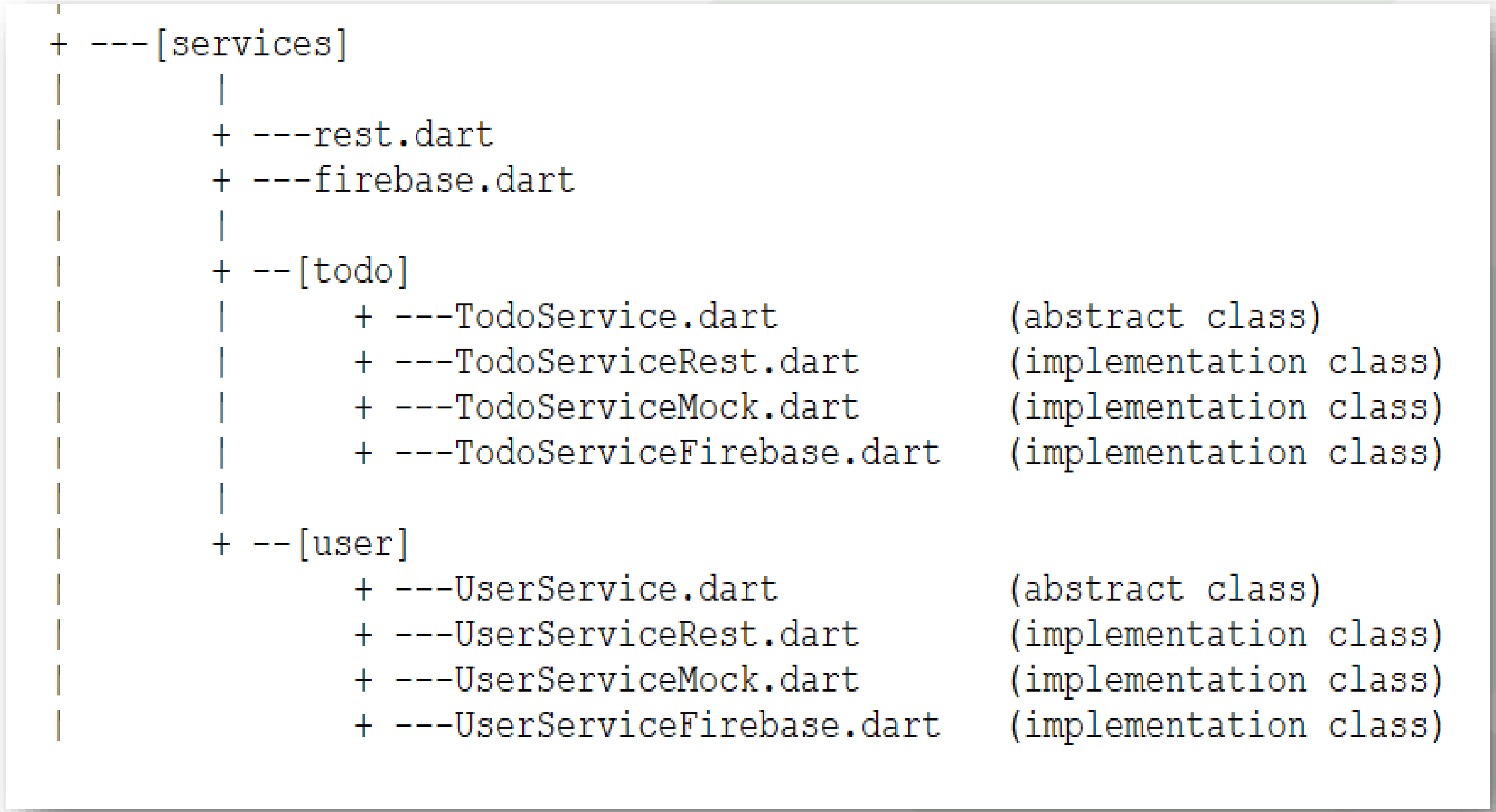
Project File Structure (2)



```
|
+ ---[screens]
|       + --view.dart        (generic view class)
|       + --viewmodel.dart   (generic viewmodel class)
|       |
|       + --[login]
|       |       + ---login_view.dart
|       |       + ---login_viewmodel.dart
|       |       + ---[widgets]
|       |
|       + --[todolist]
|       |       + ---todolist_view.dart
|       |       + ---todolist_viewmodel.dart
|       |       + ---[widgets]
|       |               + ---appbar_view.dart
|       |                + ---todo_view.dart
|       |
|       + --[other_screen]
|               + ---other_view.dart
|               + ---other_viewmodel.dart
|               + ---[widgets]
|
+ ---[widgets]   (shared widgets)
|
```

- Each screen has its own folder
- Views and viewmodels for the screen are put in the same folder
- The widgets folder in each screen folder is meant for refactoring widgets from the screen. In case a widget code is too large to be placed in the same screen code.
- View and Viewmodel are generic classes used to build the screens.
- The widgets folder at outside are for custom widgets shared between screens.

Project File Structure (3)



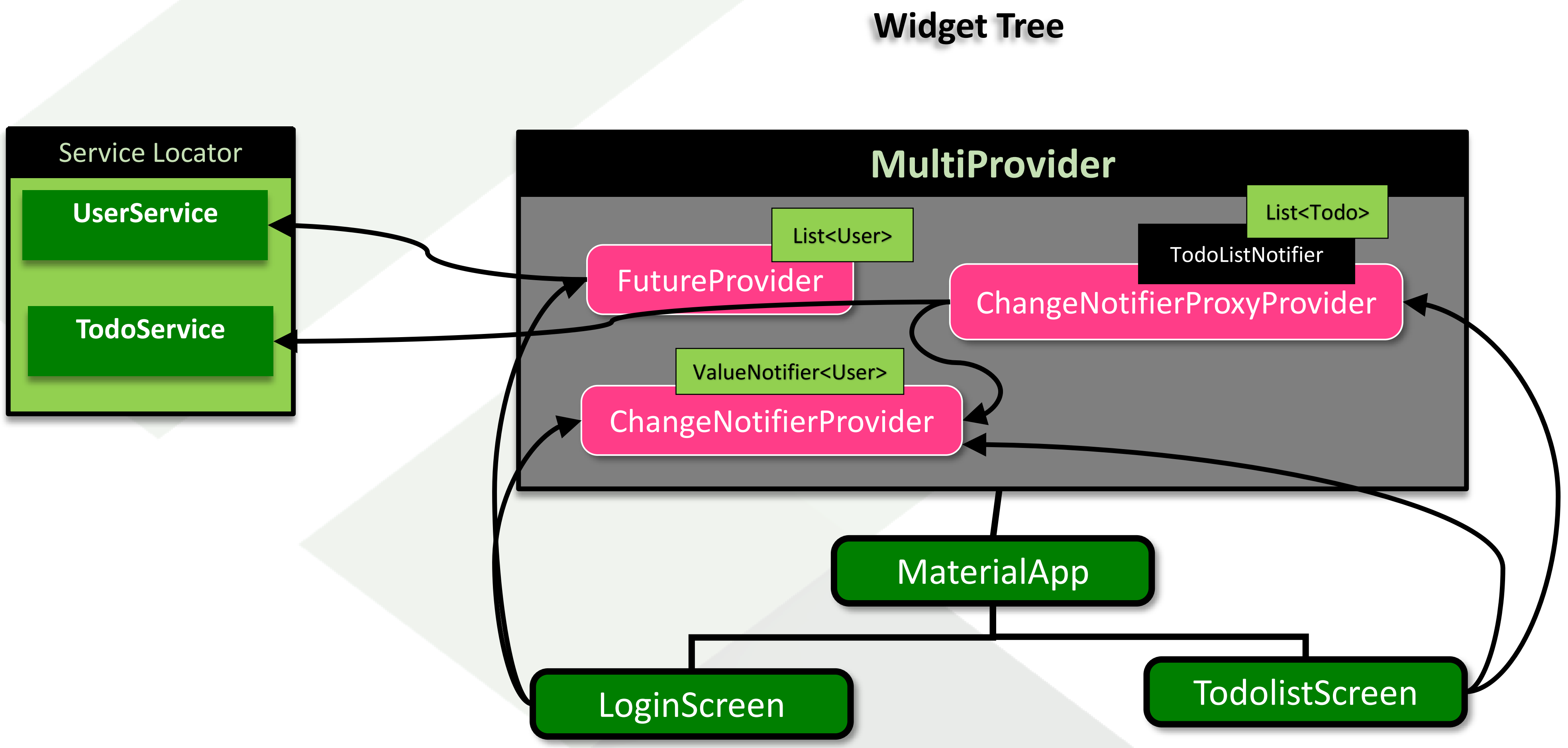
```
|
+ ---[services]
|         |
|         + ---rest.dart
|         + ---firebase.dart
|         |
|         + --[todo]
|         |       + ---TodoService.dart             (abstract class)
|         |       + ---TodoServiceRest.dart         (implementation class)
|         |       + ---TodoServiceMock.dart         (implementation class)
|         |       + ---TodoServiceFirebase.dart     (implementation class)
|         |
|         + --[user]
|                 + ---UserService.dart             (abstract class)
|                 + ---UserServiceRest.dart         (implementation class)
|                 + ---UserServiceMock.dart         (implementation class)
|                 + ---UserServiceFirebase.dart     (implementation class)
```

- Shared services are put at root folder
- A service may have different types of implementations, such actual and mock service
- Thus, create a folder for each service
- Services are registered in Service Locator (at app/dependencies.dart)

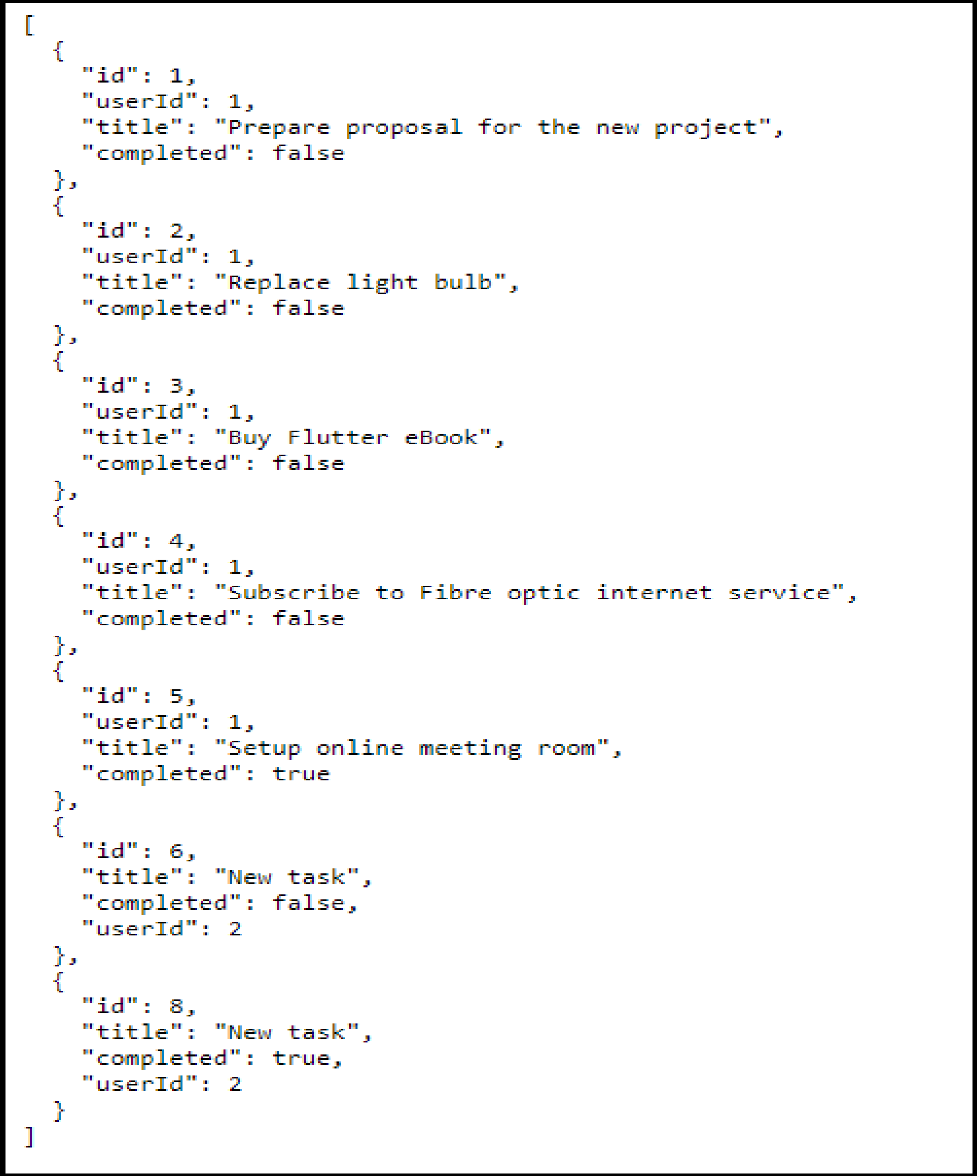
The Codebase

Codebase

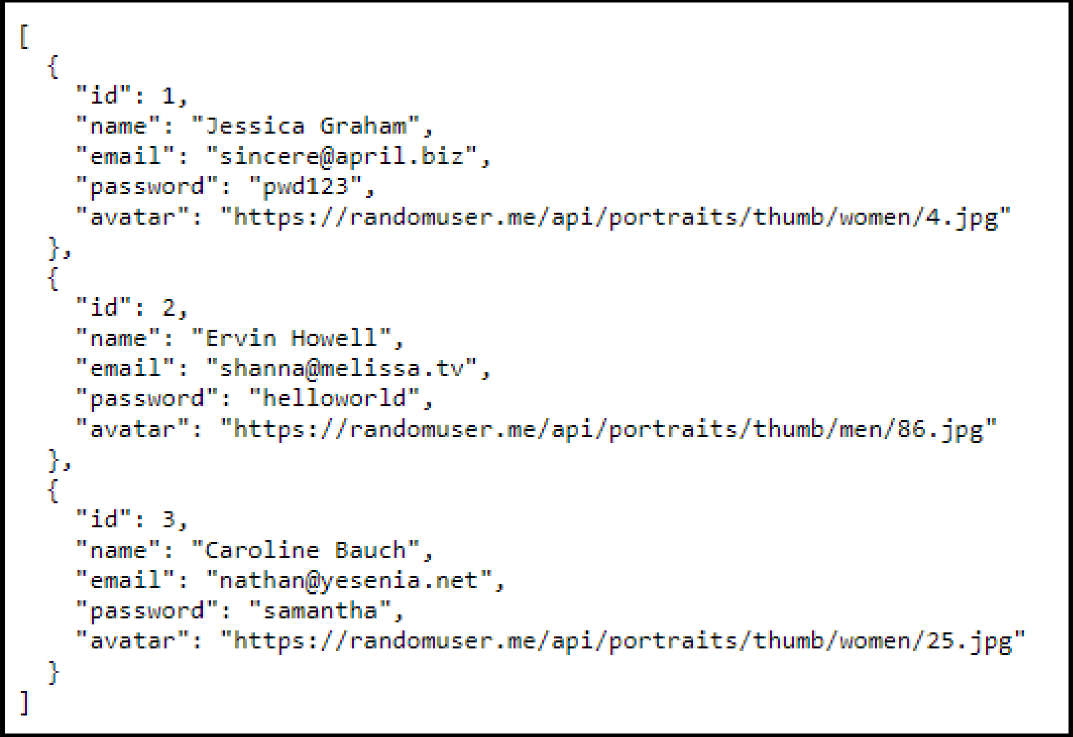
Current Implementation using Providers



The todos and users collections



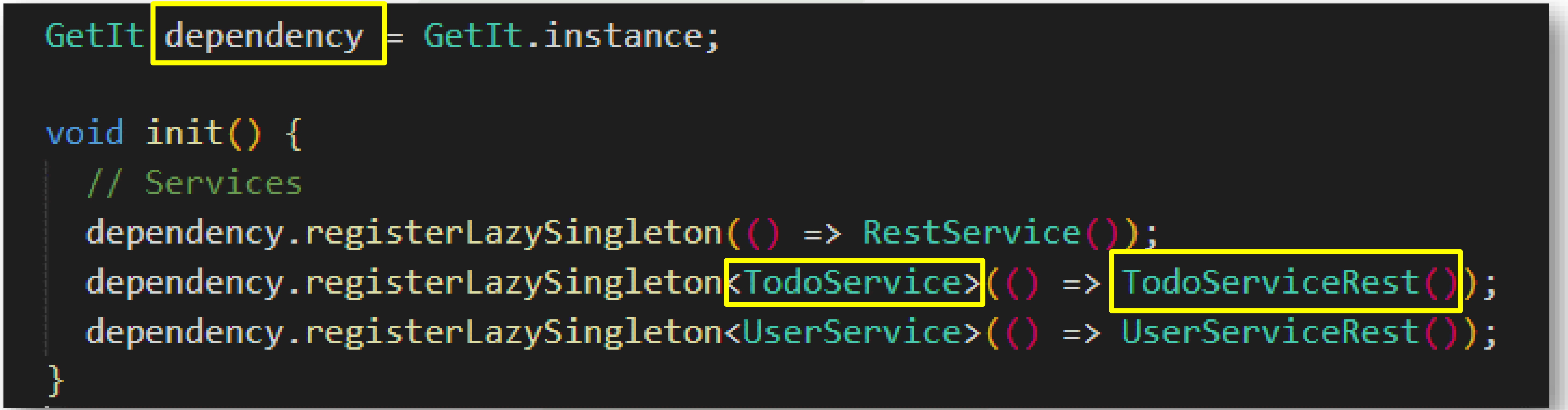
```
[
  {
    "id": 1,
    "userId": 1,
    "title": "Prepare proposal for the new project",
    "completed": false
  },
  {
    "id": 2,
    "userId": 1,
    "title": "Replace light bulb",
    "completed": false
  },
  {
    "id": 3,
    "userId": 1,
    "title": "Buy Flutter eBook",
    "completed": false
  },
  {
    "id": 4,
    "userId": 1,
    "title": "Subscribe to Fibre optic internet service",
    "completed": false
  },
  {
    "id": 5,
    "userId": 1,
    "title": "Setup online meeting room",
    "completed": true
  },
  {
    "id": 6,
    "title": "New task",
    "completed": false,
    "userId": 2
  },
  {
    "id": 8,
    "title": "New task",
    "completed": true,
    "userId": 2
  }
]
```



```
[
  {
    "id": 1,
    "name": "Jessica Graham",
    "email": "sincere@april.biz",
    "password": "pwd123",
    "avatar": "https://randomuser.me/api/portraits/thumb/women/4.jpg"
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "email": "shanna@melissa.tv",
    "password": "helloworld",
    "avatar": "https://randomuser.me/api/portraits/thumb/men/86.jpg"
  },
  {
    "id": 3,
    "name": "Caroline Bauch",
    "email": "nathan@yesenia.net",
    "password": "samantha",
    "avatar": "https://randomuser.me/api/portraits/thumb/women/25.jpg"
  }
]
```

Setup Service Locator

Register dependencies to Service Locator

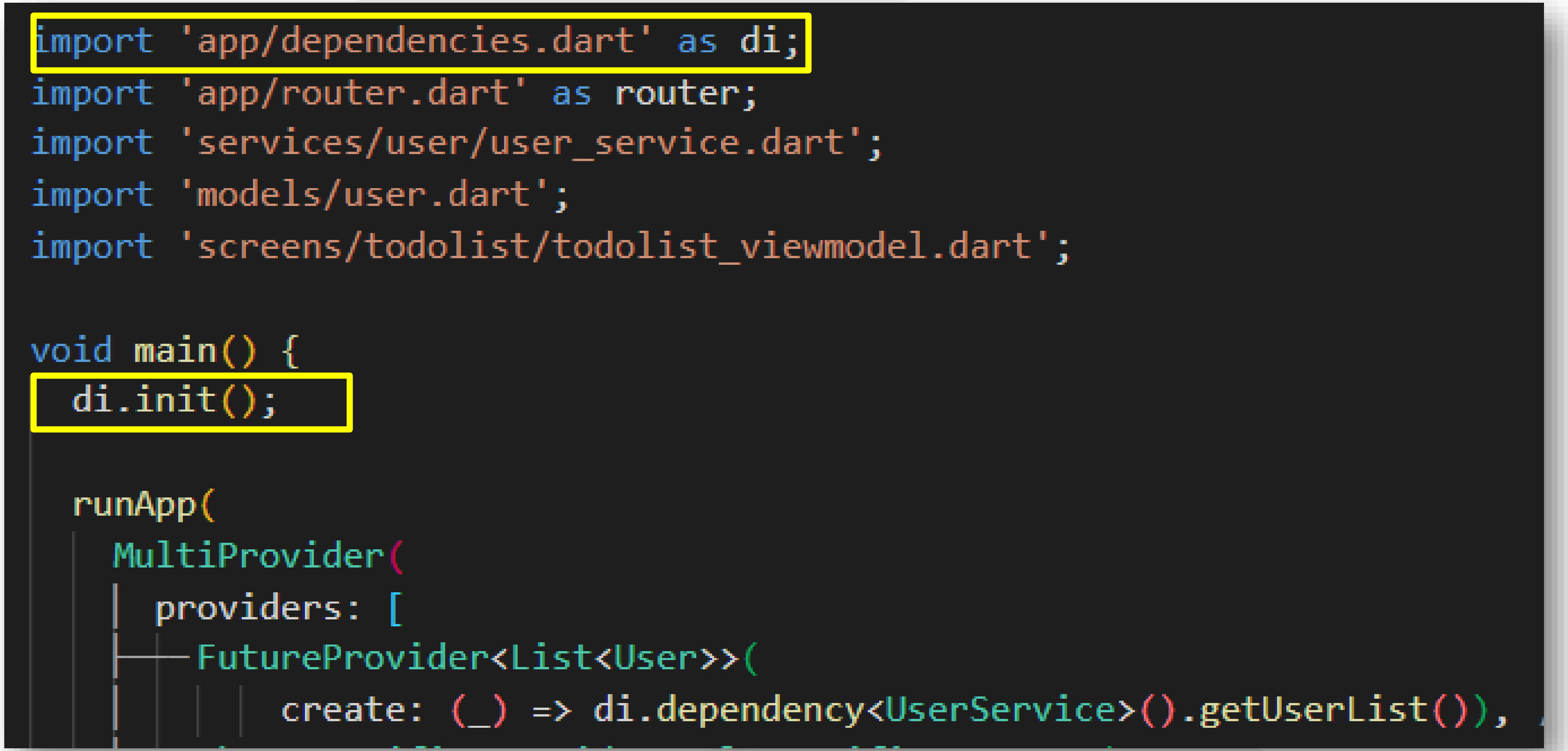


```
GetIt dependency = GetIt.instance;

void init() {
  // Services
  dependency.registerLazySingleton(() => RestService());
  dependency.registerLazySingleton<TodoService>(() => TodoServiceRest());
  dependency.registerLazySingleton<UserService>(() => UserServiceRest());
}
```

Setup Service Locator (2)

Initialize get_it in main()



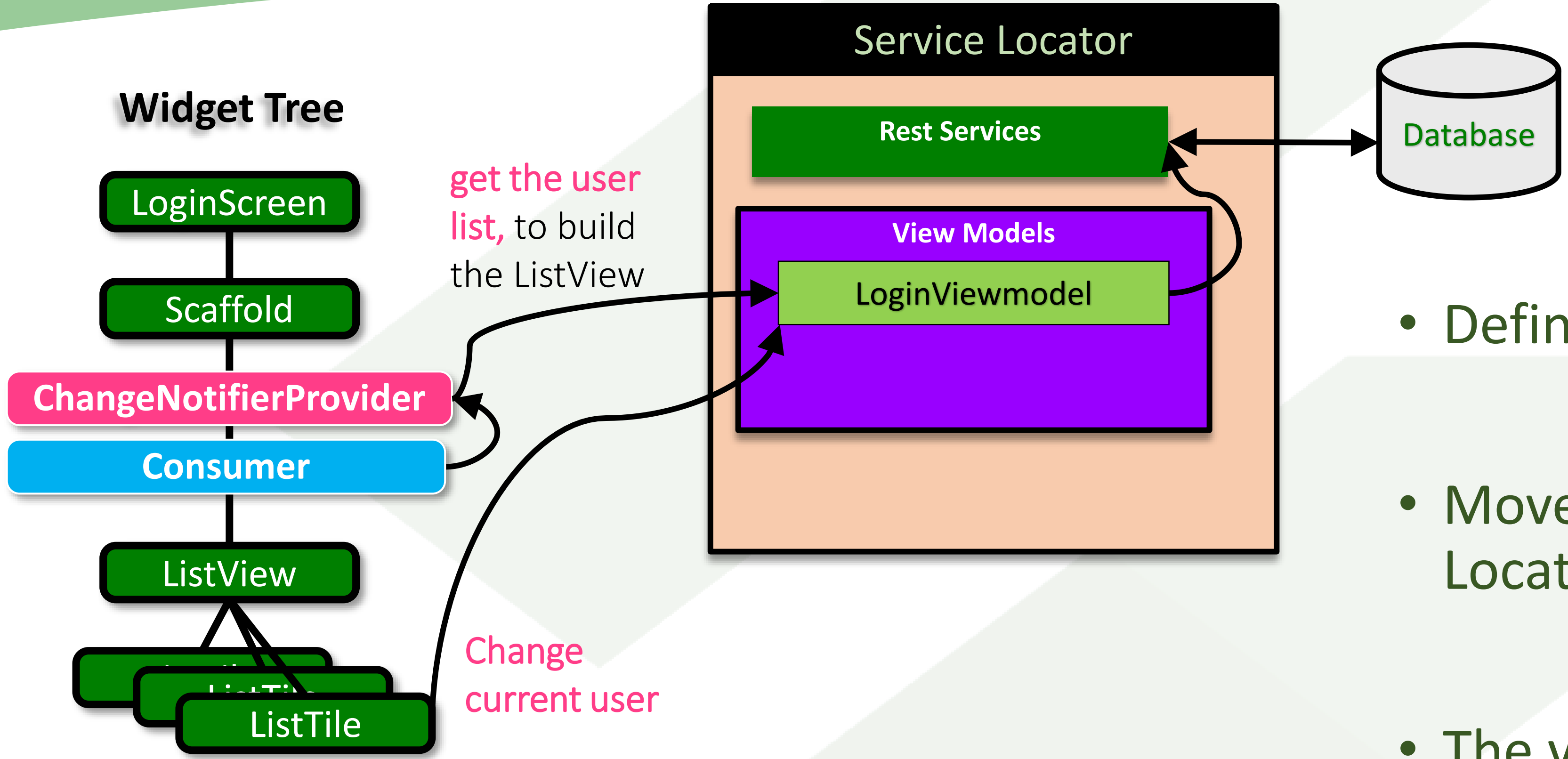
```
import 'app/dependencies.dart' as di;
import 'app/router.dart' as router;
import 'services/user/user_service.dart';
import 'models/user.dart';
import 'screens/todolist/todolist_viewmodel.dart';

void main() {
  di.init();

  runApp(
    MultiProvider(
      providers: [
        FutureProvider<List<User>>(
            create: (_) => di.dependency<UserService>().getUserList()),
```

Working on LoginScreen

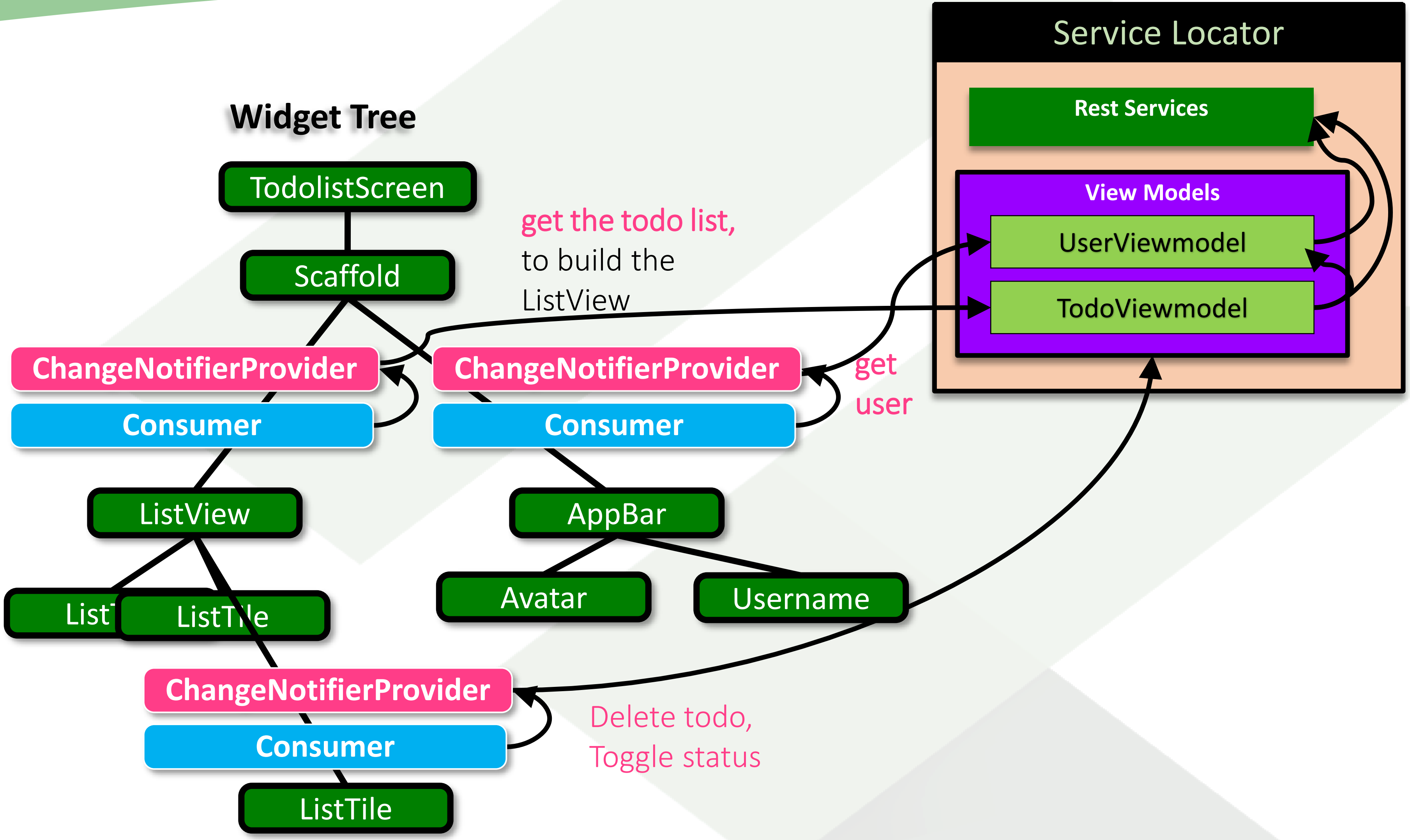
Tasks to do for Login Screen



- Define LoginViewmodel
- Move LoginViewmodel to Service Locator
- The view has its own provider and consumer
- Refactor DI from using Provider.of() to service locator

Working on TodolistScreen

Tasks to do for the Todolist Scren



Define Generic View and Viewmodel Classes

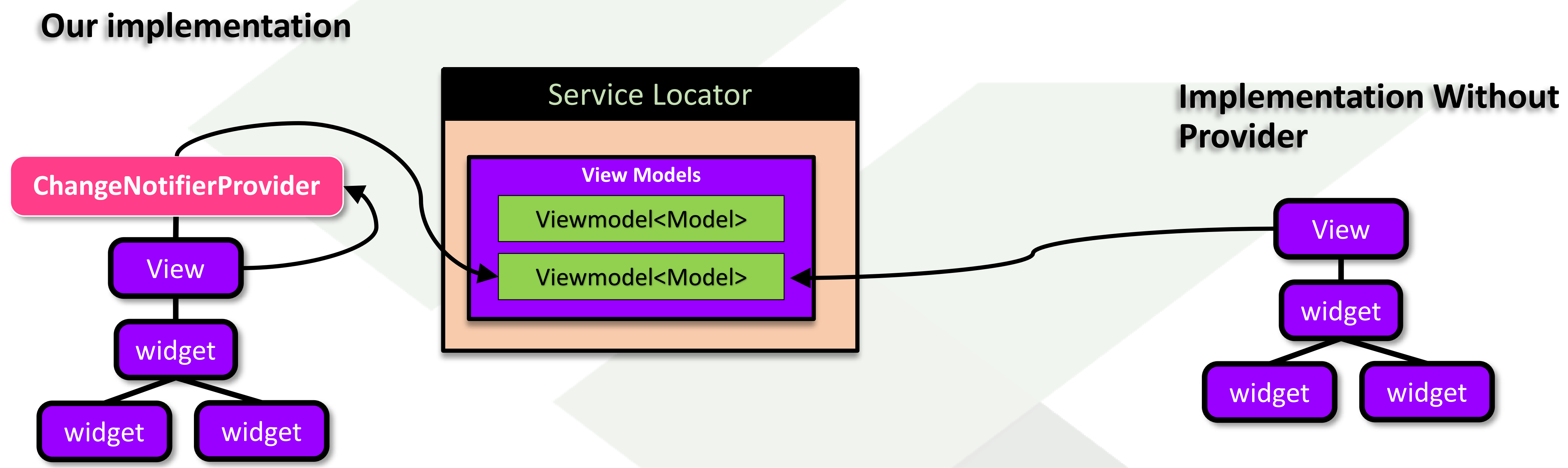
Refactor Both Screen Code Using the Generic Classes

Define Convenient View Classes

Split View to Multiple Child Views

Discussion

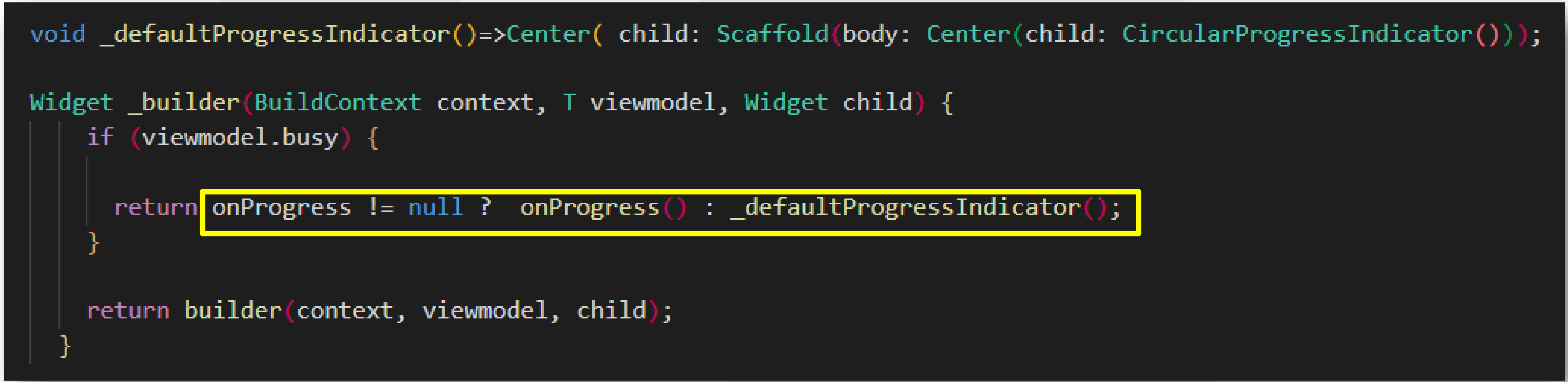
Can we use Viewmodel without Provider?



Discussion (2)

- Define other types of convenient Views
  - Replace ChangeNotifierProvider with ChangeNotifierProxyProvider
  - You may call these View classes like: ProxyView, ProxySelectorView, ProxyWidgetView

- Provide parameter onProgress on the generic View class for custom Progress Indicator



```
void _defaultProgressIndicator()=>Center( child: Scaffold(body: Center(child: CircularProgressIndicator())));

Widget _builder(BuildContext context, T viewmodel, Widget child) {
    if (viewmodel.busy) {

      return onProgress != null ?  onProgress() : _defaultProgressIndicator();
    }

    return builder(context, viewmodel, child);
  }
```

Summary

- About MVVM
- Using Provider and Service Locator for MVVM Projects
- Define Generic Classes for Scalable Project
- Things to extend further