Mobile Application Architecture

Provider

Lecture and Demo

Jumail Bin Taliba
School of Computing, UTM
July 2020

Agenda

- Introduction to Providers
- Setting Up Providers
- Consuming Providers
- Provider Dependency Injection
- Types of Providers
- Provider State Management

Source Code

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

Introduction to Provider

What is Provider?

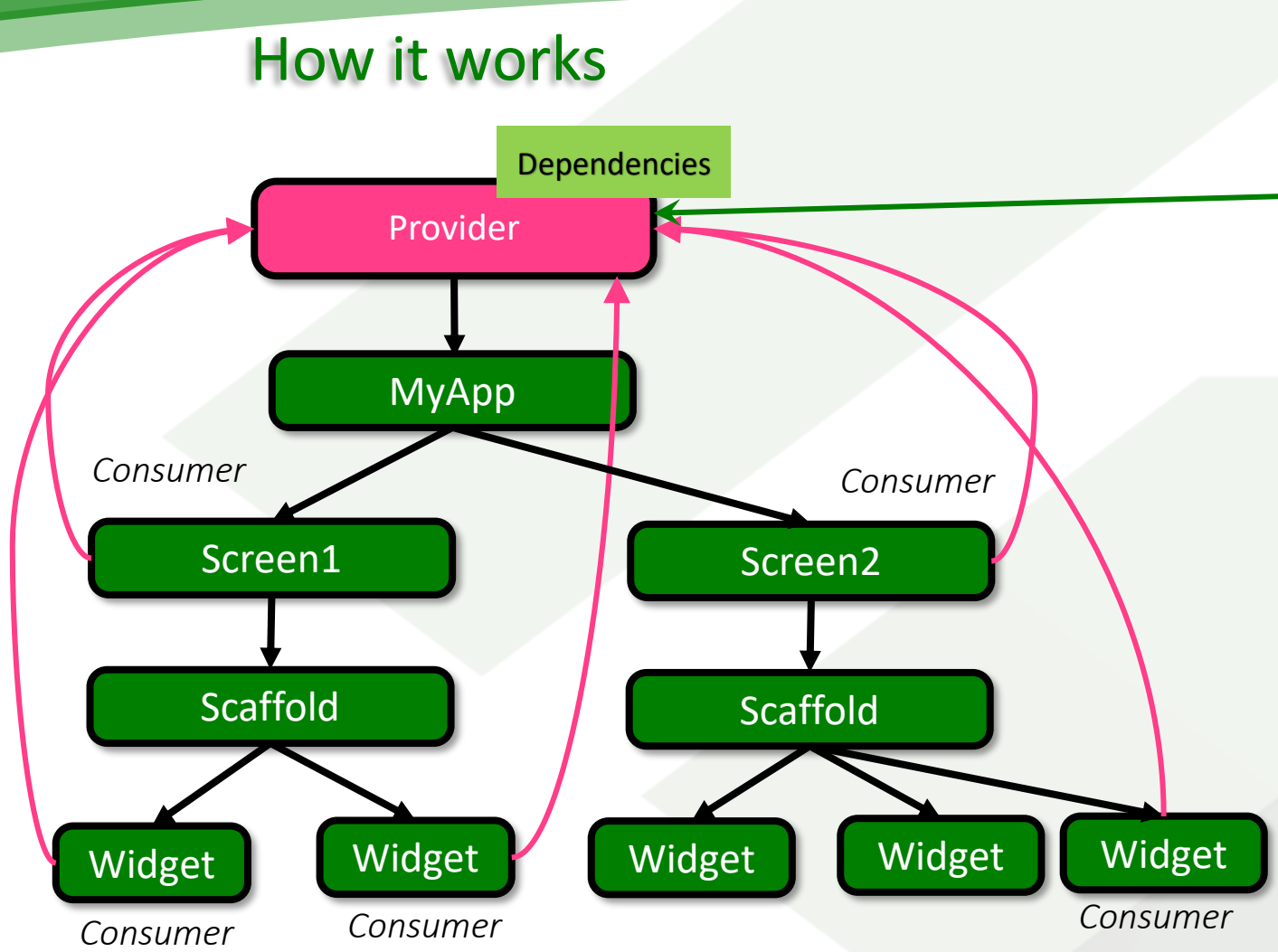
- A wrapper around InheritedWidget
- Used for Dependency Injection and State management

Adopts Publish-Subscribe Design Pattern

- Provider (the Publisher) signals its consumers if it gets updated
- It is up to the consumers (the Subscribers) to pull the data from the provider.

Introduction to Provider (2)

How it works

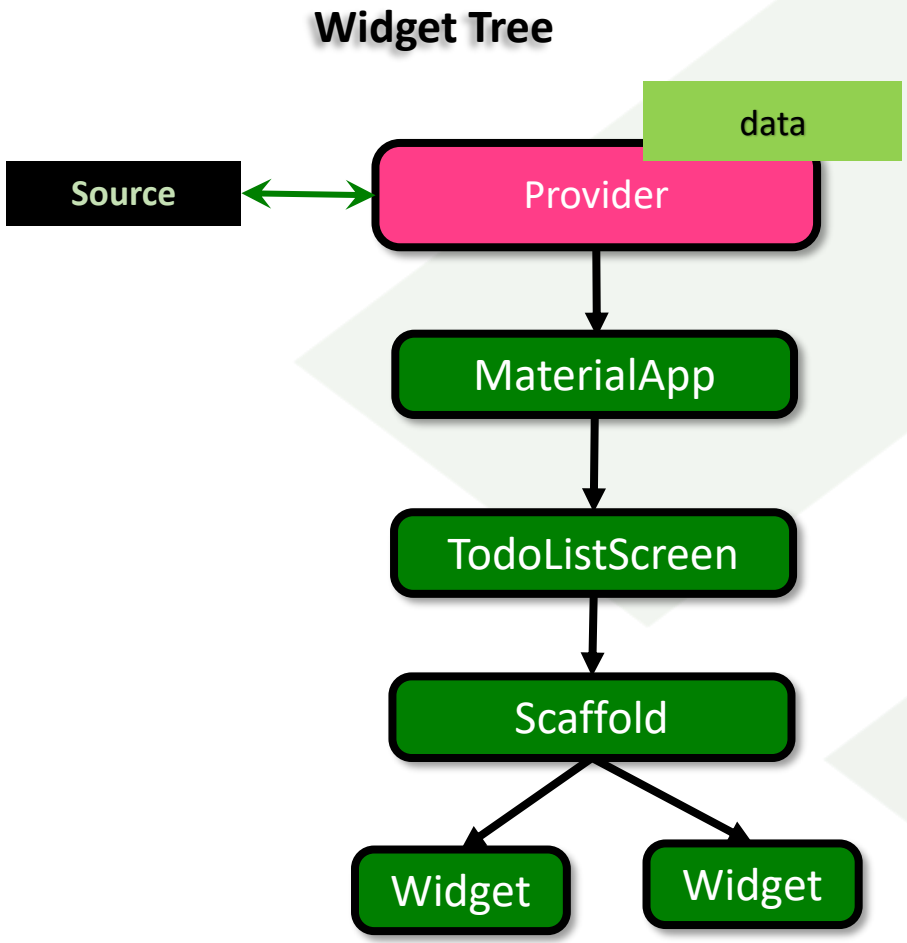


Source

- API
- Firebase
- Widgets
- Streams
- Futures
- ChangeNotifiers
- Providers

- Providers:
  - Hold and expose the dependencies
  - Signal the consumers if the dependencies get updated
- Consumers:
  - Need to be registered to the provider in order to get notified the changes.
  - Need to pull the update from the Provider on their own.
- Source:
  - Feeder to the dependencies
  - Make changes / updates

Setting Up Providers



- A provider is a widget
- Wrap the parent of the consumer widget tree with a provider widget

Code example:

```
runApp(
    Provider<User>.value(
      value: user,
      child: MaterialApp(
        home: TodoListScreen(),
      ),
    ),
);
```

Setting Up Providers (2)

Provider works by data types:

- A provider needs to be specified with a type, i.e. based on the data it will provide
- A type is used only once per widget tree

Code example:

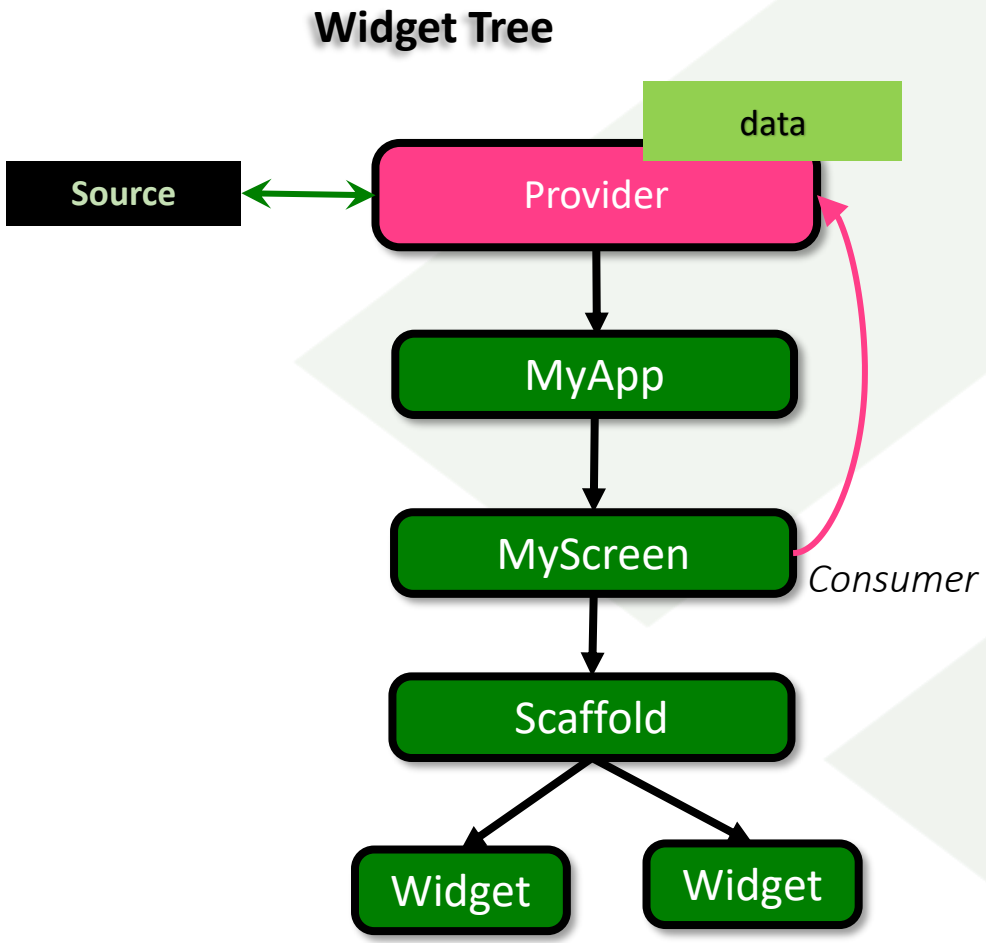
```
runApp(
    Provider<User>.value(
      value: user,
      child: MaterialApp(
        home: TodoListScreen(),
      ),
    ),
  );
```

Setting Up Providers (3)

Types of providers:

- Provider
- FutureProvider
- StreamProvider
- ChangeNotifierProvider
- ProxyProvider
- ChangeNotifierProxyProvider
- … many more

Consuming Providers



- All widgets under the Provider have access to the it
- Accessing the provider done with

`Provider.of<T>(context)`

Code example:

```
Scaffold _buildMainScreen(context) {
  final user = Provider.of<User>(context);

  return Scaffold(
    appBar: AppBar(
      leading: CircleAvatar(
        backgroundImage: NetworkImage(user.avatar),
      ), // CircleAvatar
      title: Text(user.name),
    ), // AppBar
```

Consuming Providers (2)

- The method Provider.of() does two things:
  - Accessing providers
  - Registering the corresponding widget as a listener / consumer / subscriber to the provider

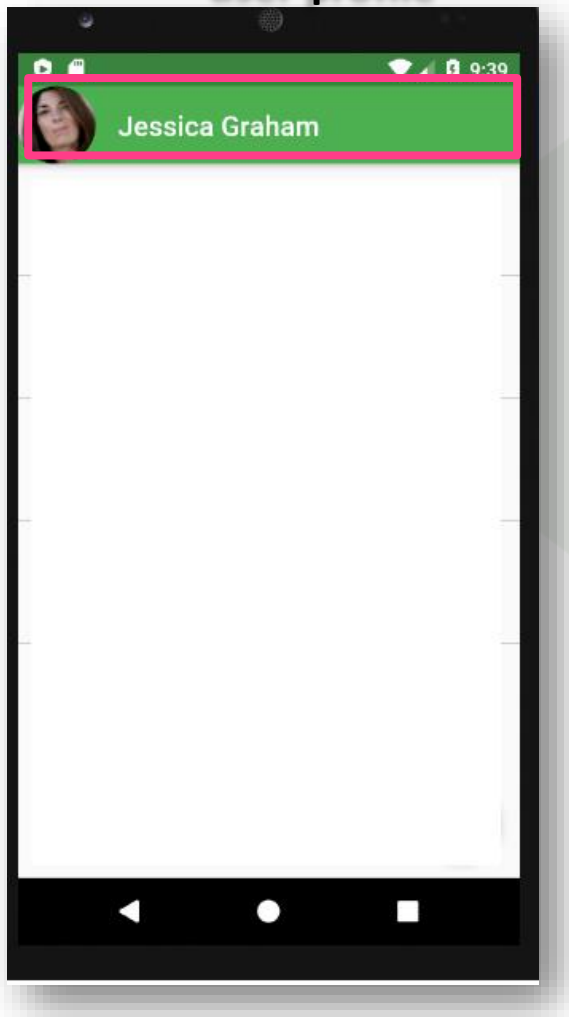
- To only access the provider without registering a widget as a listener:

```
Provider.of<T>(context, listen: false)
```

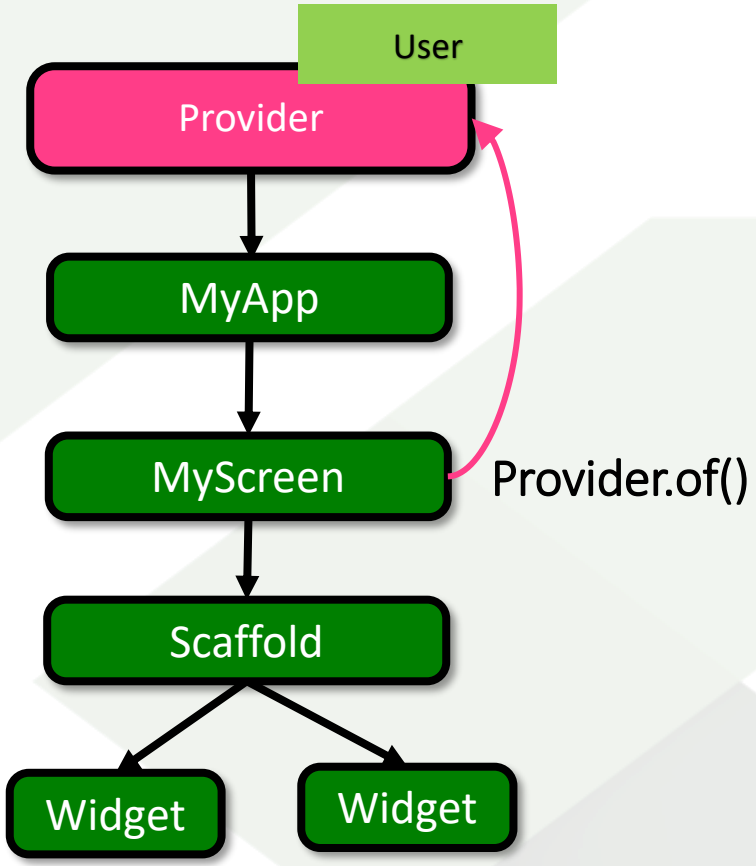
- Convenient widgets to use the method Provider.of():
  - Consumer
  - Selector

Consuming Providers (3)

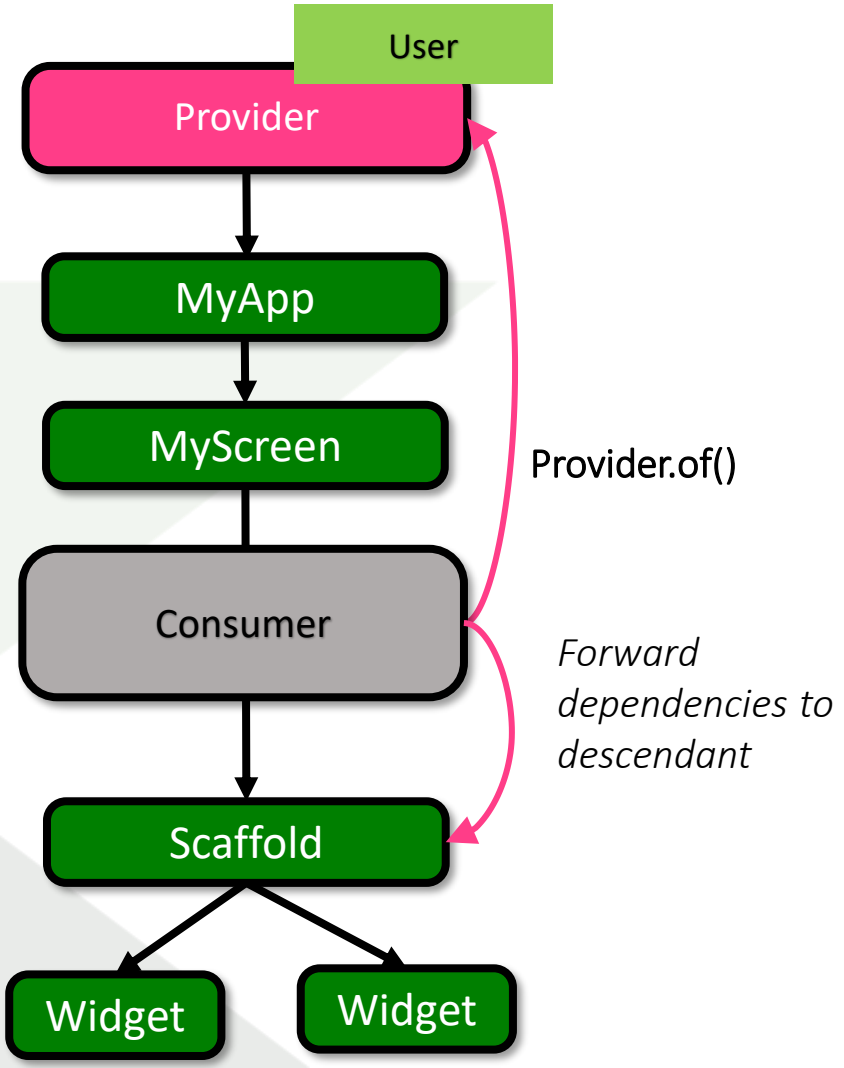
Example: Show user profile



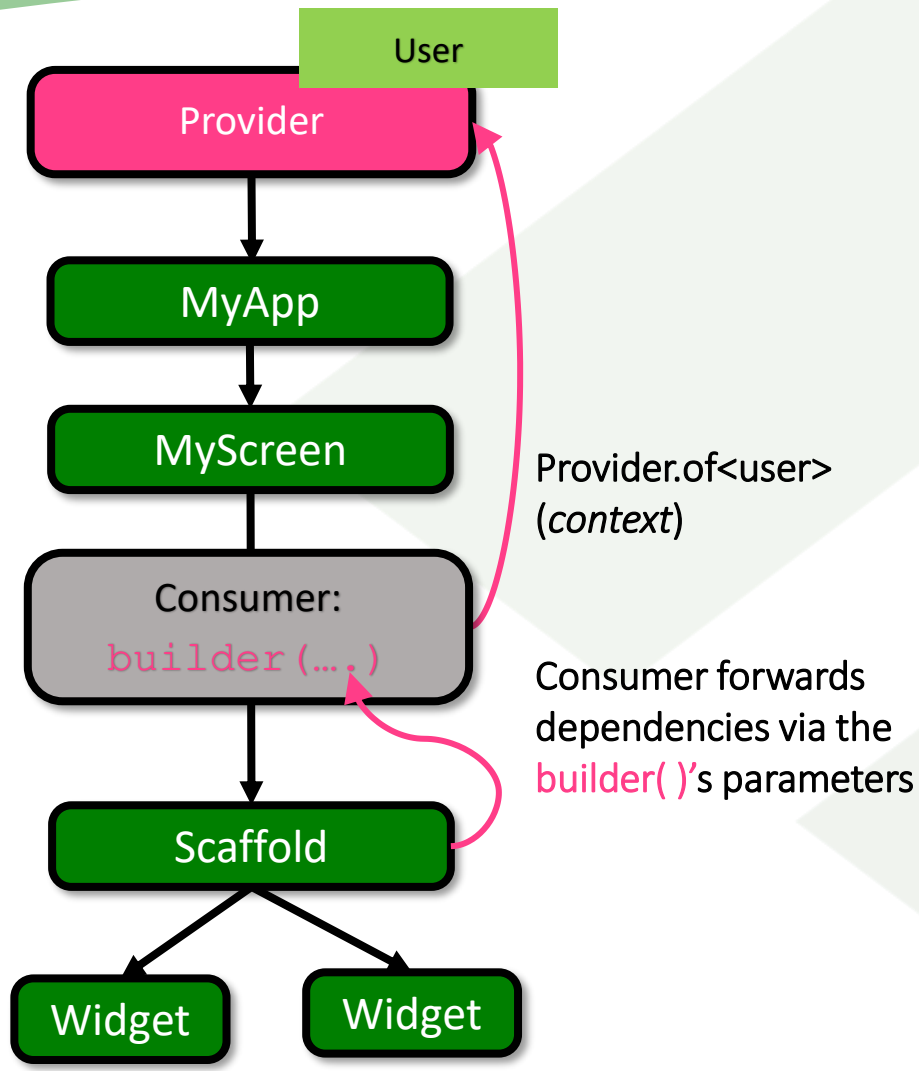
With Provider.of()



With Consumer widget



Consuming Providers (4)



Consumer widget

- It will call to Provider.of()
  - To obtain data from provider
  - To register itself as a listener to the provider
- It will forward the obtained data to its descendant via the `builder`'s parameters.
- It does not need a `BuildContext` object to setup

Consuming Providers (5)

Consumer Example code:

```
Widget _buildMainScreen(context) {

  return Consumer<User>(

    builder: (_, user, __) => Scaffold(

      appBar: AppBar(
        leading: CircleAvatar(
          backgroundImage: NetworkImage(user.avatar),
        ), // CircleAvatar
        title: Text(user.name),
      ), // AppBar

      body: ListView.separated(
```

Consuming Providers (6)

The `builder` callback

- Build a widget tree based on the obtained value from a provider
- Parameters:

```
builder: (context, data, widget){
    return buildDynamicWidget(…)
}
```

  - *context:* a `BuiltContext` object
  - *data*: obtained from provider
  - *widget*: static widget to be referenced inside builder. This parameter is supplied with the Consumer's **`child`** parameter

Example:

```
Widget _buildMainScreen(context) {
  return Consumer<User>(
    child: Icon(Icons.access_alarm),
    builder: (_, user, widget) => Scaffold(
      appBar: AppBar(
        leading: CircleAvatar(
          backgroundImage: NetworkImage(user.avatar),
        ), // CircleAvatar
        title: Text(user.name),
        actions: [widget], // AppBar
```

Consuming Providers (7)

Selector widget

- An equivalent to Consumer, but with the capability to prevent widgets get rebuilt if they don't change.

- To setup a Selector:

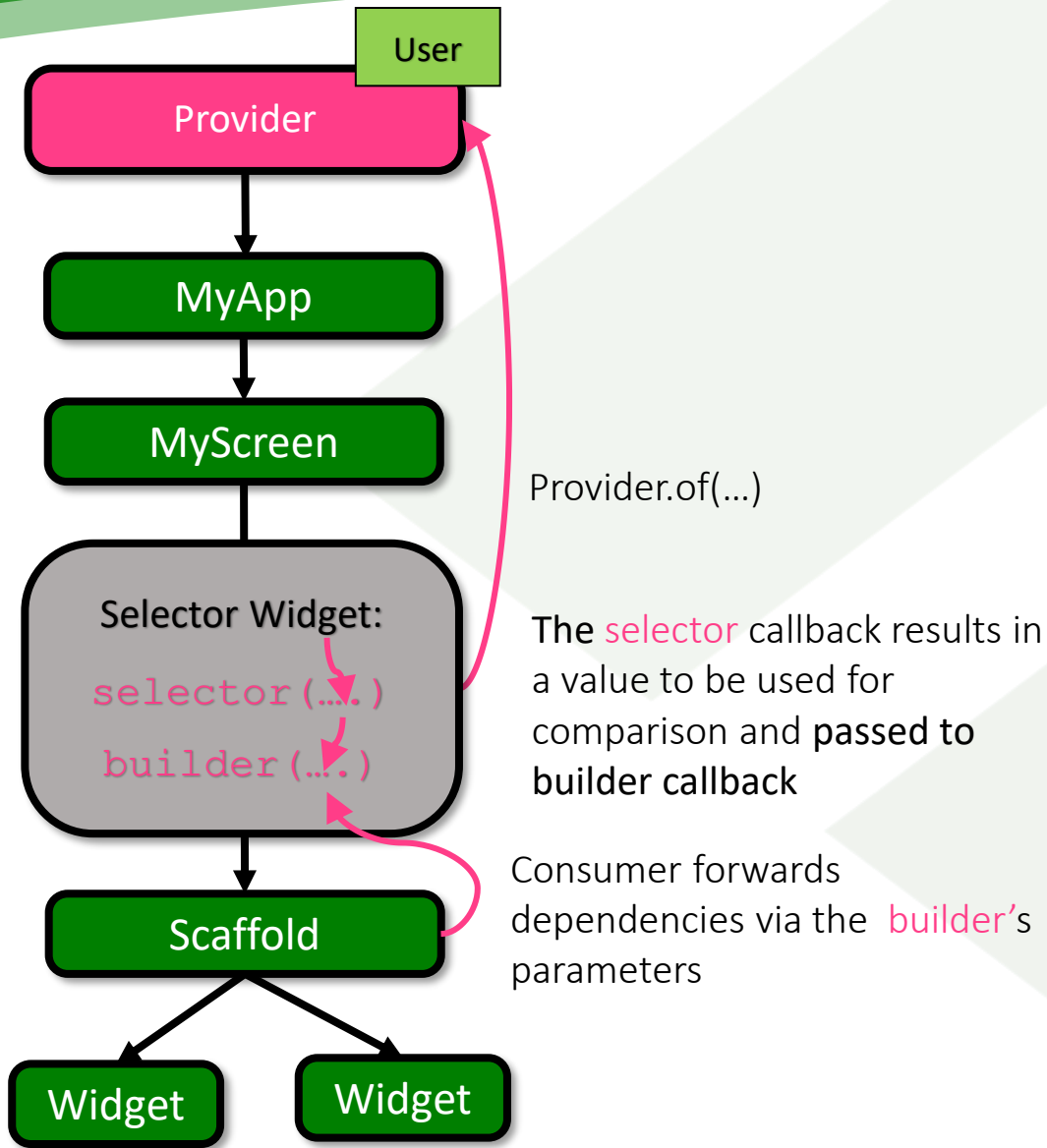
```
Selector<T1,T2>();
```

  - *T1:* The data type of the provider, used to climb up the widget tree.
  - *T2:* The data type of the observed changes.

Example: *Rebuild widgets only the list's size (or length) has changed*

```
Selector< List<DataModel>, int >(
    selector: (_,list)=>list.length,
    builder : (…) => ListView( ….. )
);
```

Consuming Providers (8)



Selector widget

- The Selector widget obtains data using Provider.of(), then passes it to its selector callback.
- The selector callback is then tasked to return an object to be passed to the builder callback.
- The Selector widget determines whether the descendant needs to be rebuilt by comparing the previous and new result of the selector callback.

Consuming Providers (9)

The `selector` callback

- Create a new data based on the obtained data from a provider
- Pass the created data to the builder callback
- The data is also used by the Selector widget to decide whether to rebuild the descendant widget

- Parameters and returns:

```
selector: (context, data){
    return newData
}
```

  - *context:* a `BuiltContext` object
  - *data*: obtained from provider
  - *newData*: the result from the selector

Example:

Rebuild the Scaffold widget only if the user id has changed

```
Widget _buildMainScreen(context) {

  return Selector<User, int>(

    selector: (_, User user) => user.id,

    builder: (context, uid, widget) {

      final user = Provider.of<User>(context, listen: false);

      return Scaffold(

        appBar: AppBar(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(user.avatar),
          ), // CircleAvatar
          title: Text(user.name),
        ), // AppBar
```

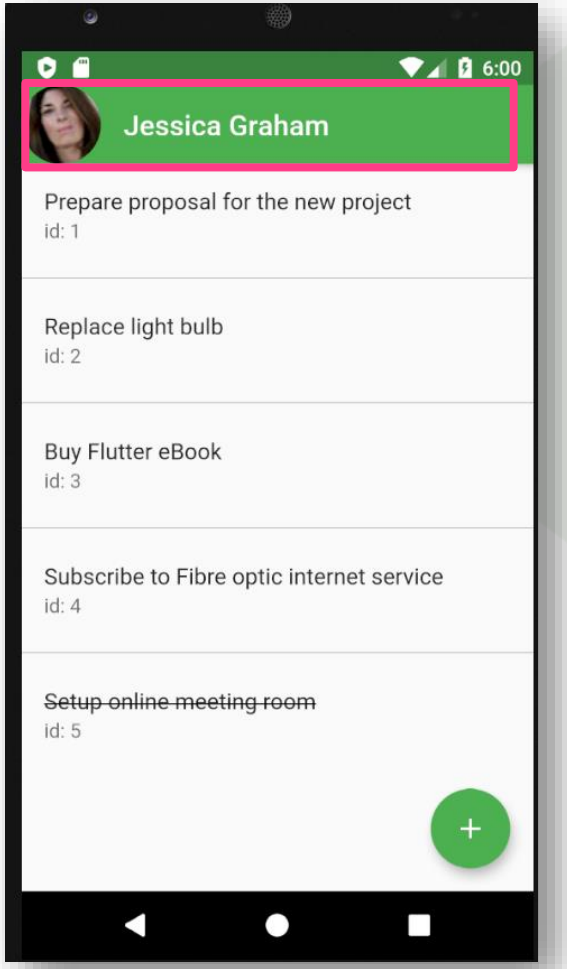
Demo

Provider Dependency Injection

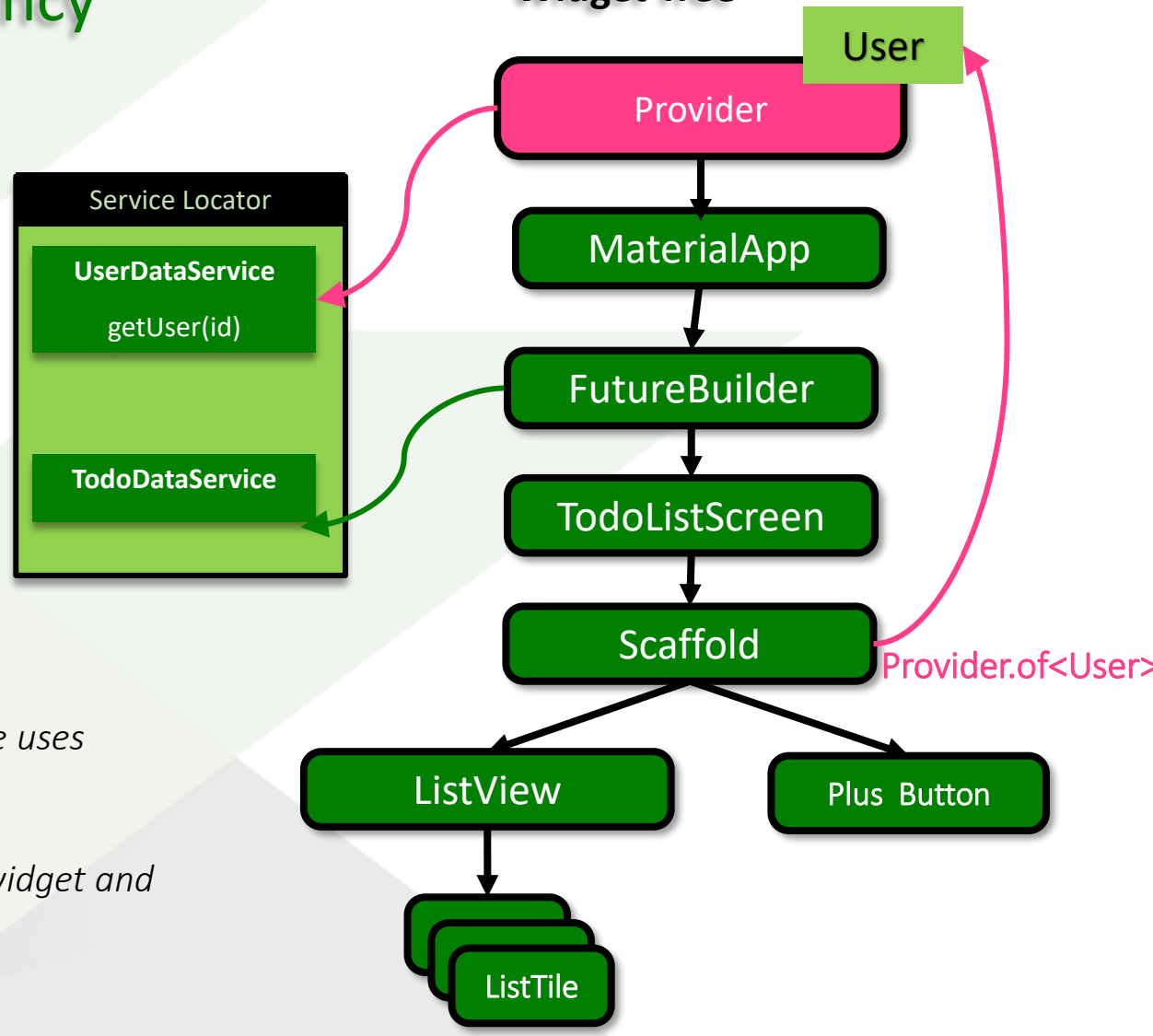
Provider Dependency Injection

Inject User dependency with Provider

User Interface



Widget Tree



Note:
In this example, only the User state uses Provider.

The todo list is still using Stateful widget and FutureBuilder

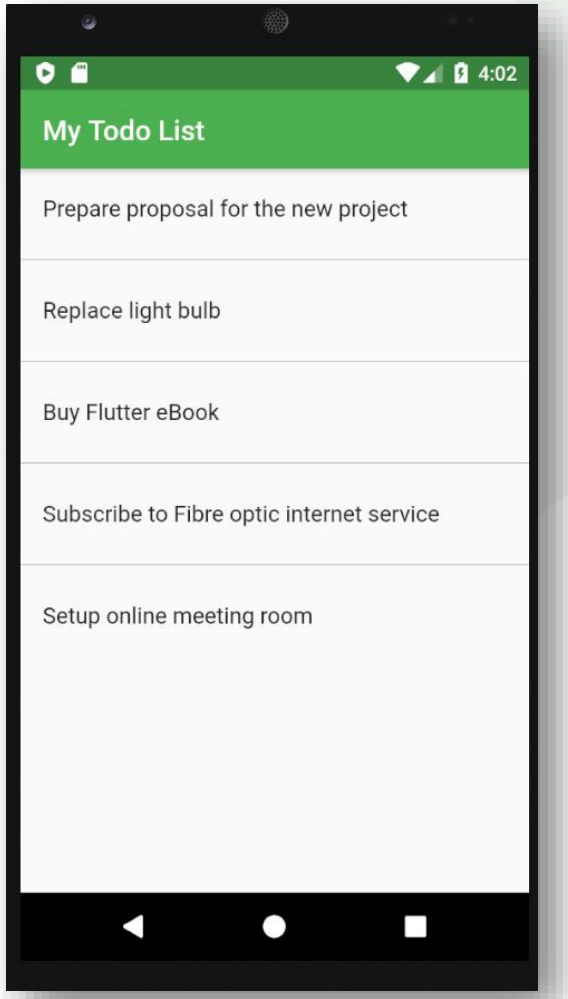
Demo

FutureProvider

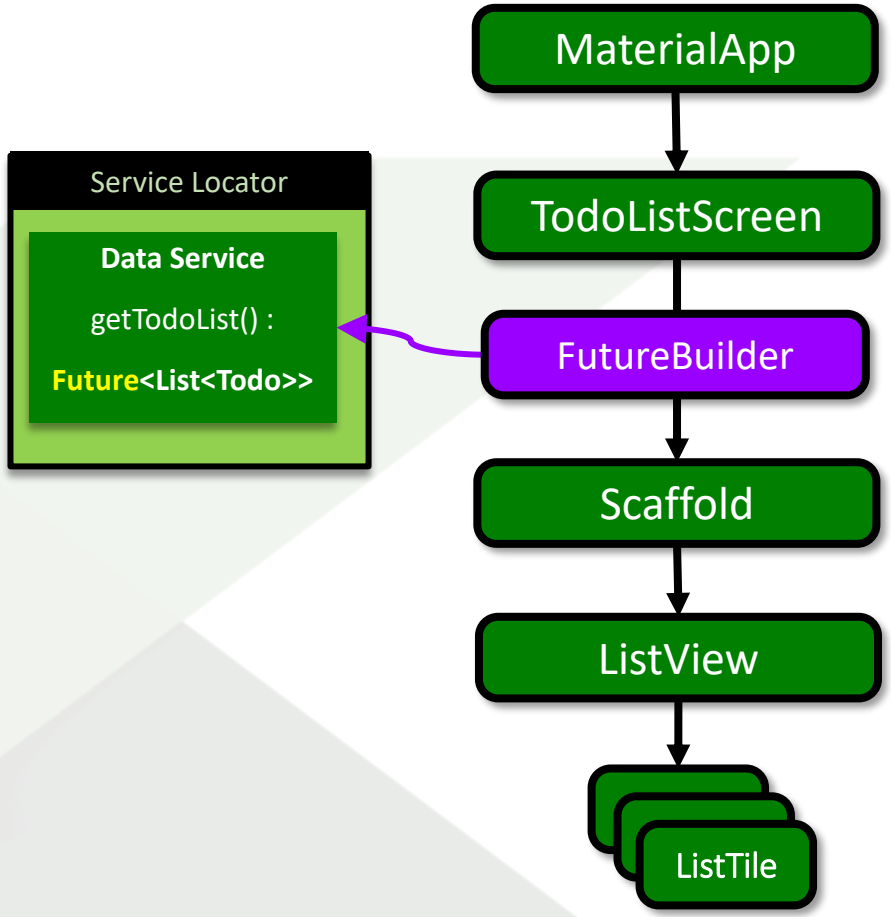
FutureProvider

Using FutureBuilder Widget (No Provider yet)

User Interface



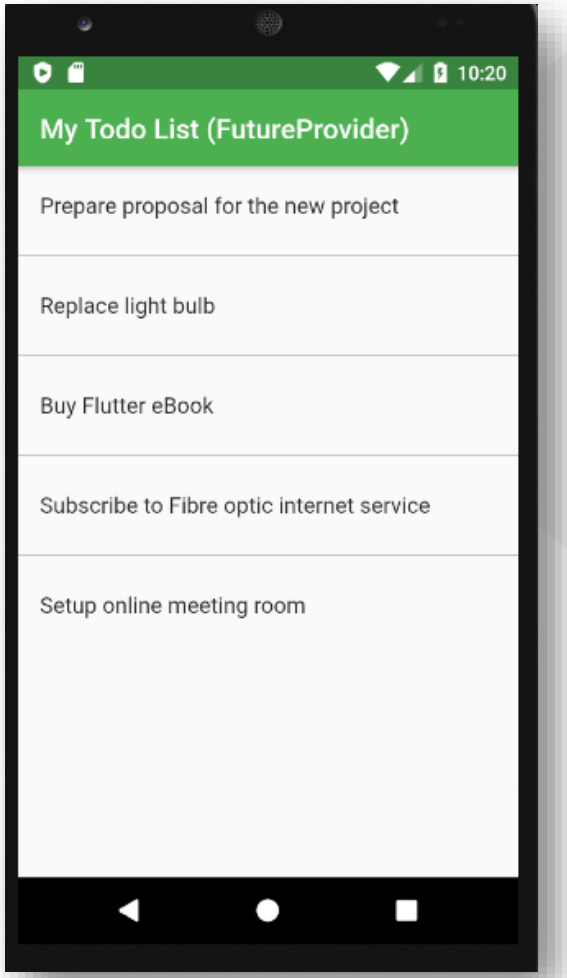
Widget Tree



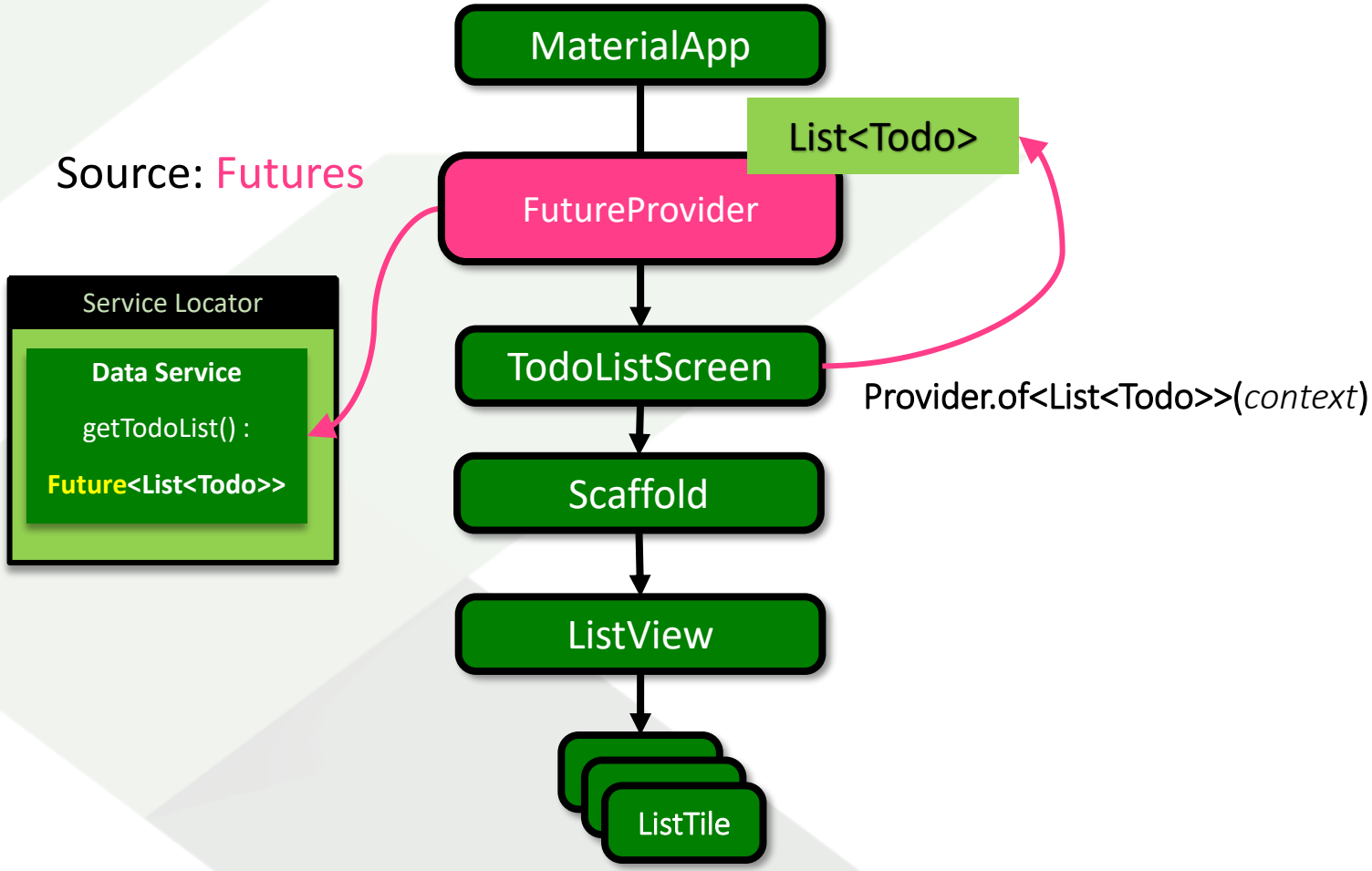
FutureProvider (2)

With FutureProvider (No more FutureBuilder)

User Interface



Widget Tree

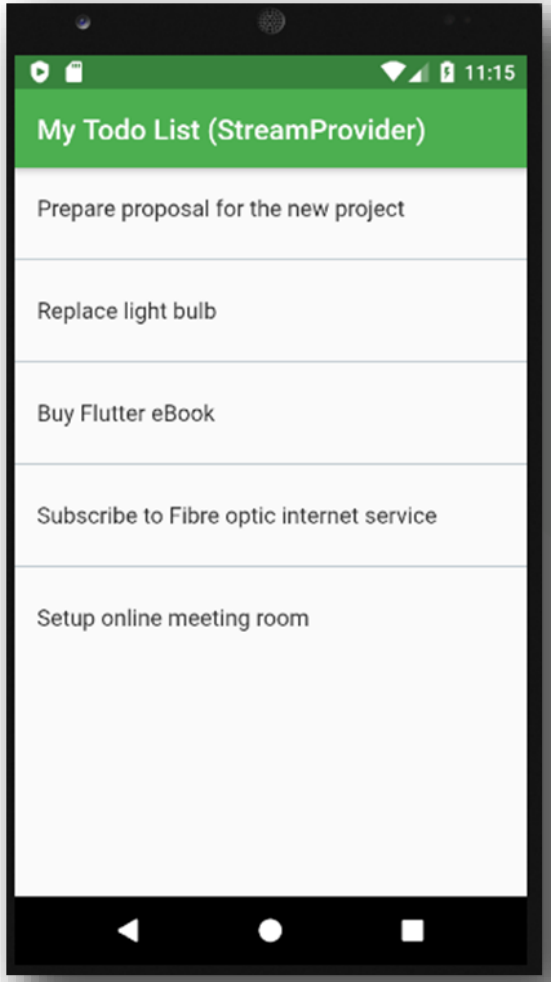


Demo

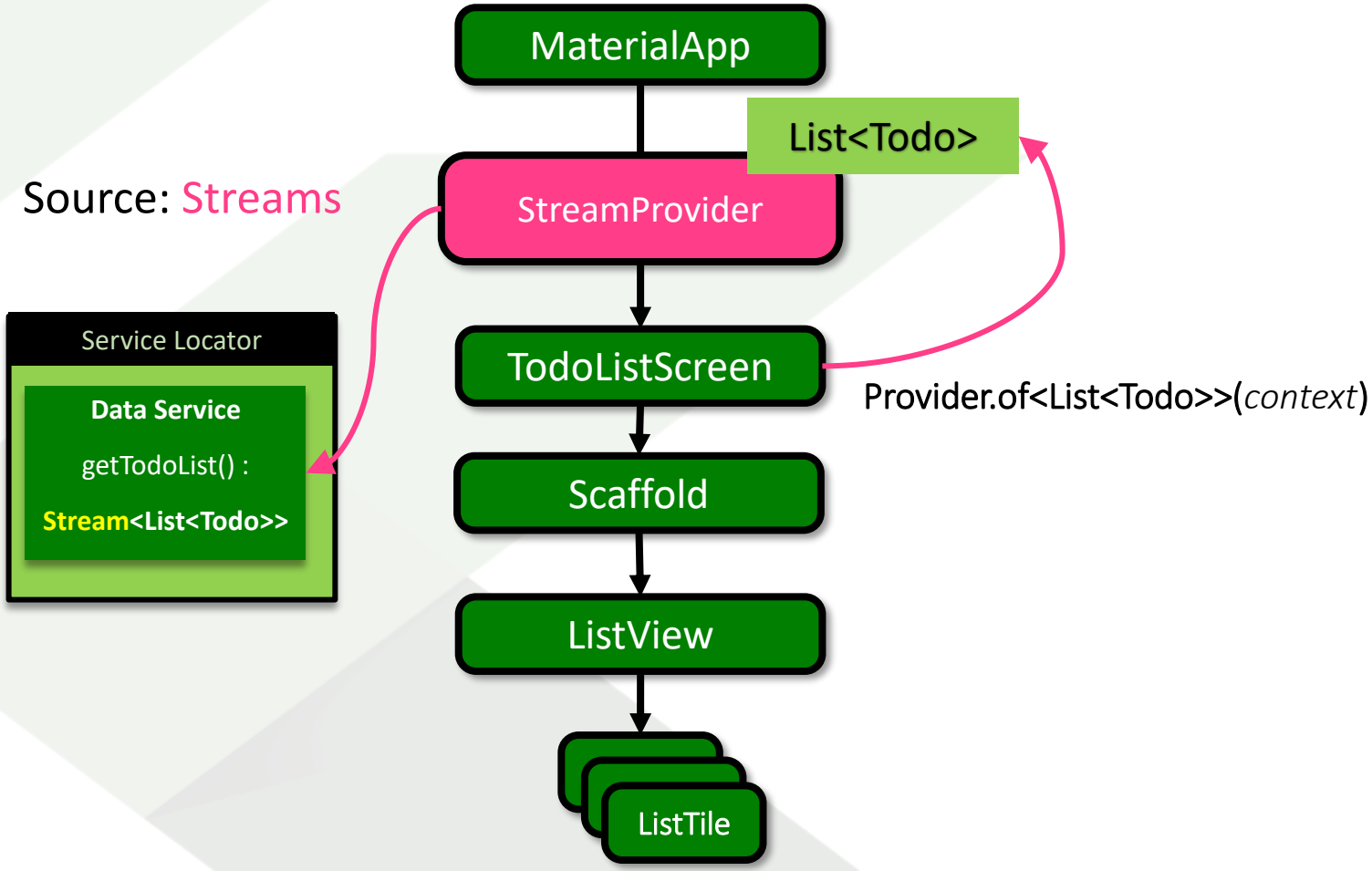
StreamProvider

StreamProvider

User Interface



Widget Tree



Demo

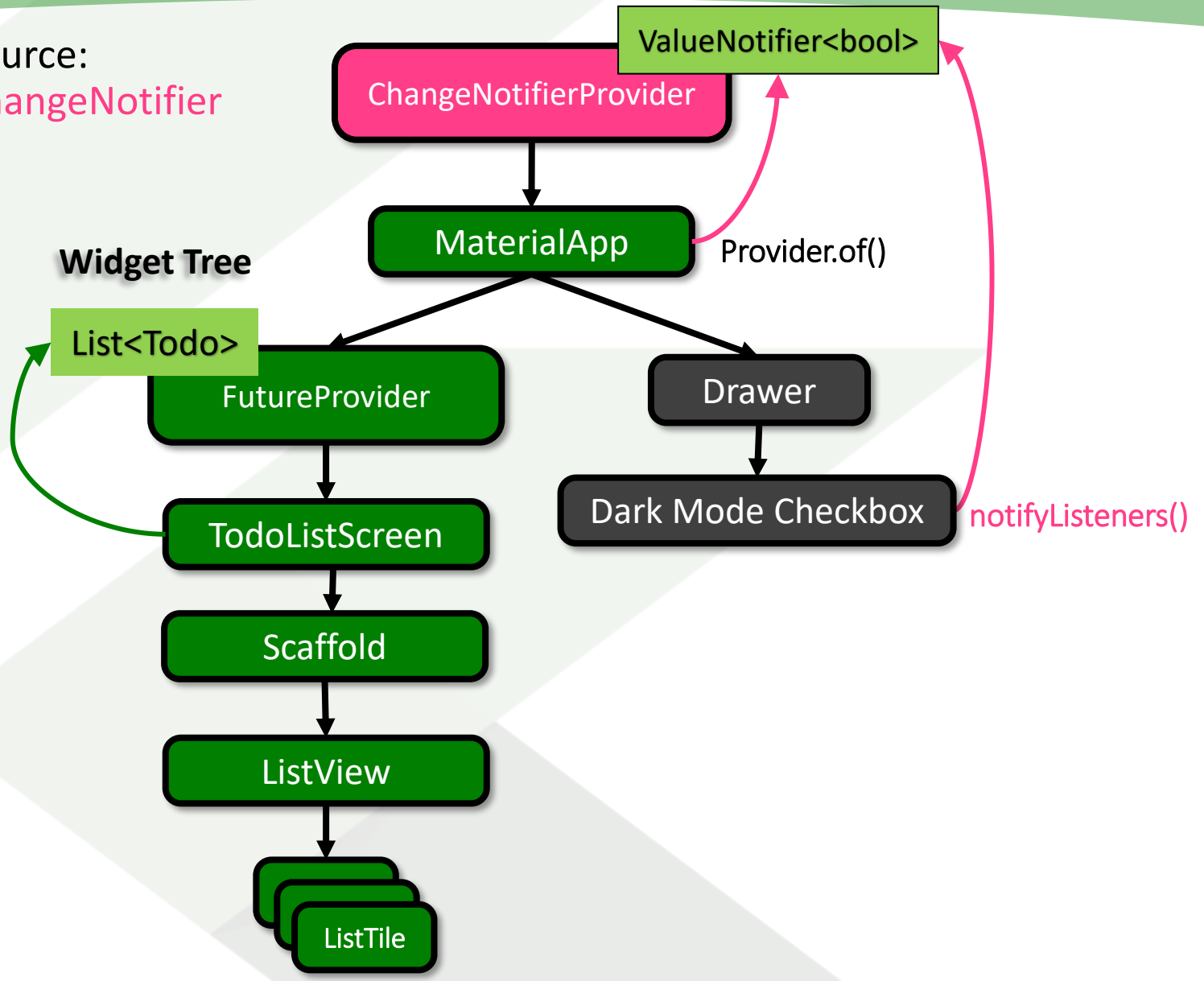
ChangeNotifierProvider

ChangeNotifierProvider

User Interface

Source:
ChangeNotifier

Widget Tree



ChangeNotifierProvider (2)

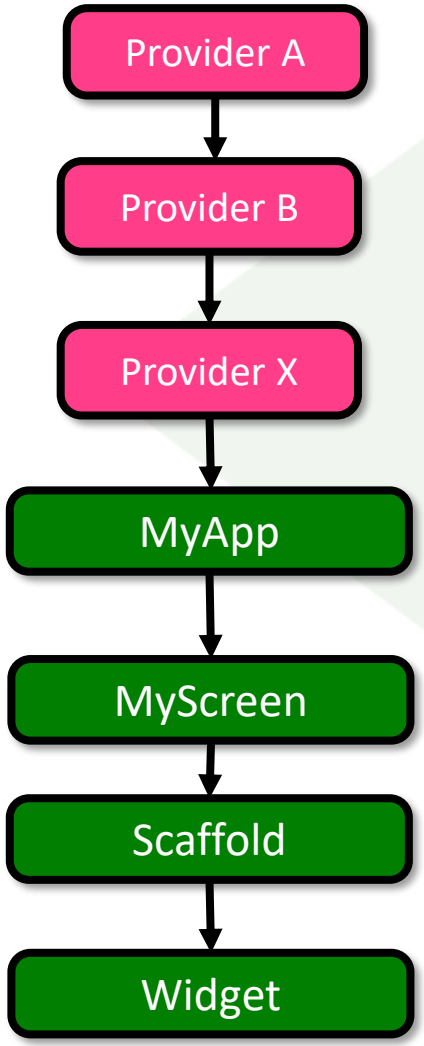
- A ChangeNotifierProvider holds a ChangeNotifier object
- States are placed inside the ChangeNotifier object

- A ChangeNotifier object has special method: notifyListeners()
  - To inform the consumers / listeners that the states have changed.
  - Same as setState() in Stateful widget

- ValueNotifier is a built-in child class of the ChangeNotifier class
- Custom ChangeNotifier class is defined with mixin or extending from the ChangeNotifier class

Working with Multiple Providers

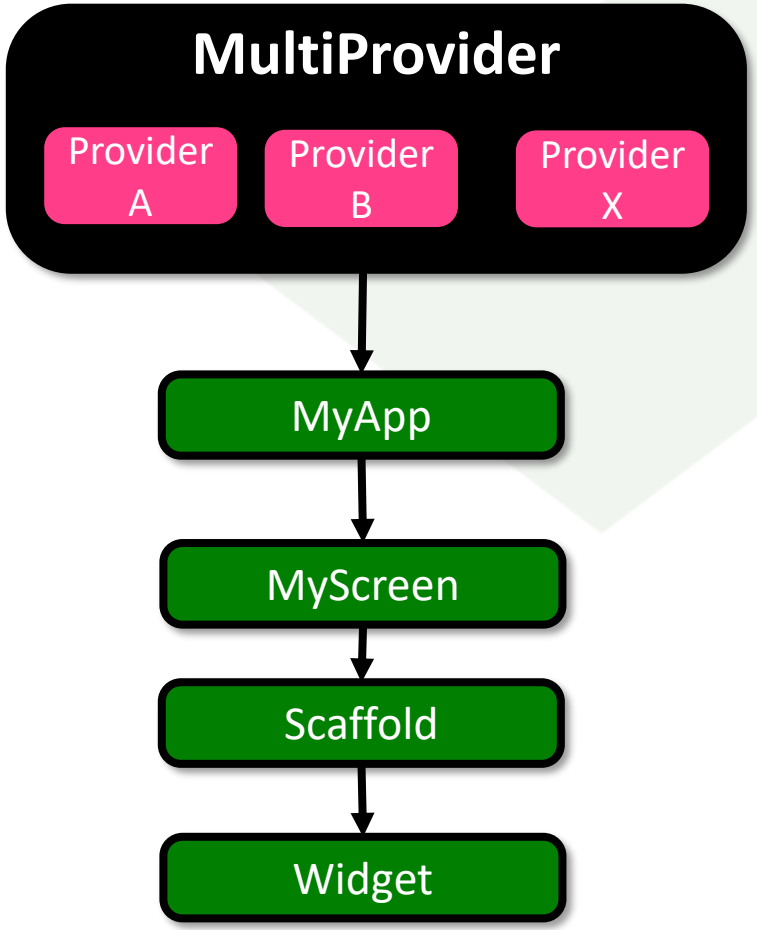
With Nested Providers



- A provider is made as a child of another provider
- The code gets messy if there are too many providers

Working with Multiple Providers (2)

With MultiProvider widget



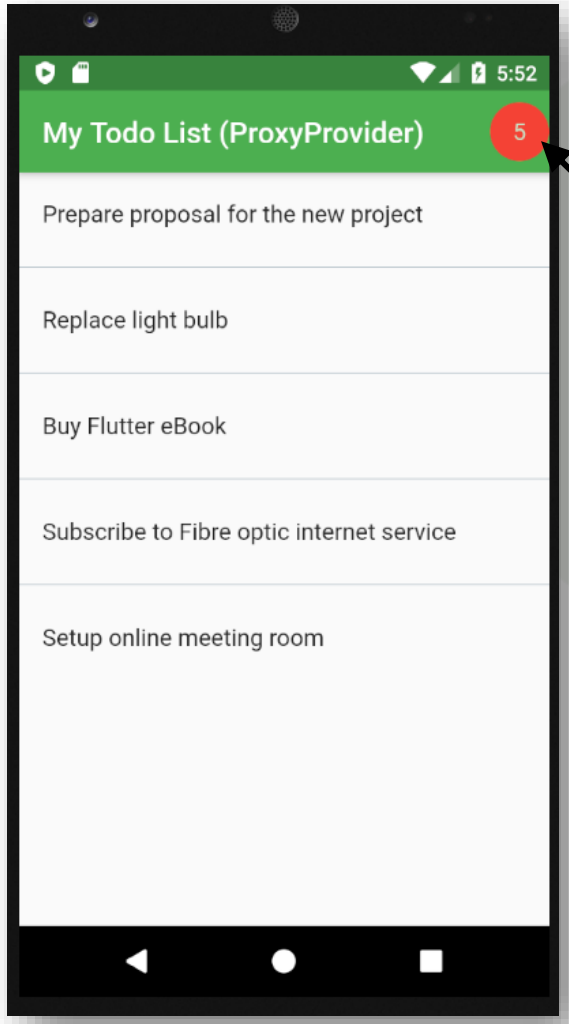
- Holds a list of providers in a single widget
- More readable code

Demo

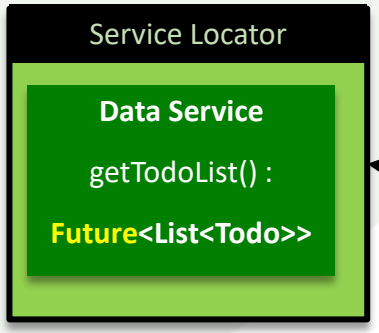
ProxyProvider and Multiple Providers

ProxyProvider

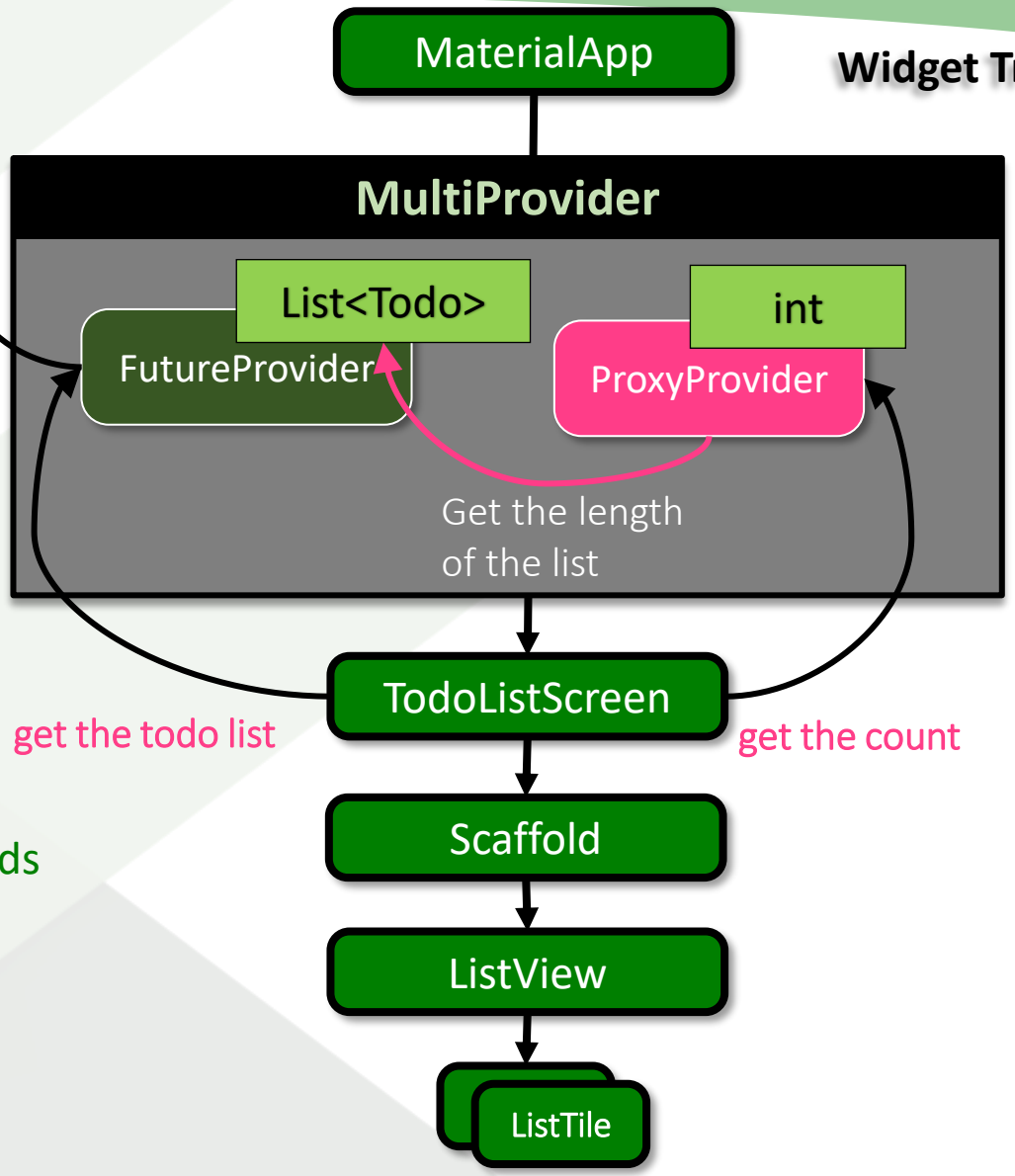
User Interface



Todo list count is to be implemented with ProxyProvider



Widget Tree



Source:

A ProxyProvider depends on other providers

Demo

ChangeNotifierProxyProvider

ChangeNotifierProxyProvider

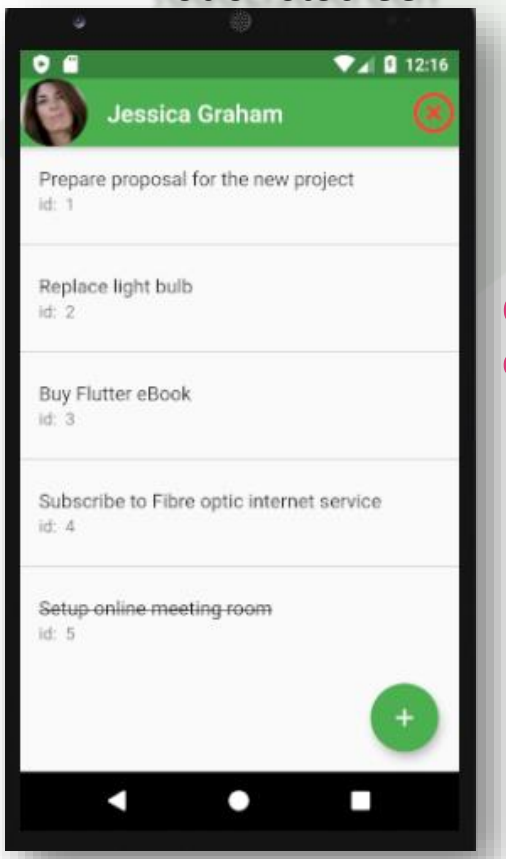
- ChangeNotifierProxyProvider:
  - It is a ChangeNotifierProvider with the capability of ProxyProvider
  - A ChangeNotifierProvider that builds and synchronizes a ChangeNotifier with external values.

- ProxyProvider:
  - It does not hold any state
  - The value it is providing is dynamically generated from its update callback.
  - Key feature: its source come from other providers

- ChangeNotifyProvider:
  - Can hold states (wrapped inside its ChangeNotify object)
  - Key feature: can perform update (via the notifyListeners of the ChangeNotify)

ChangeNotifierProxyProvider (2)

Example



TodoListScreen

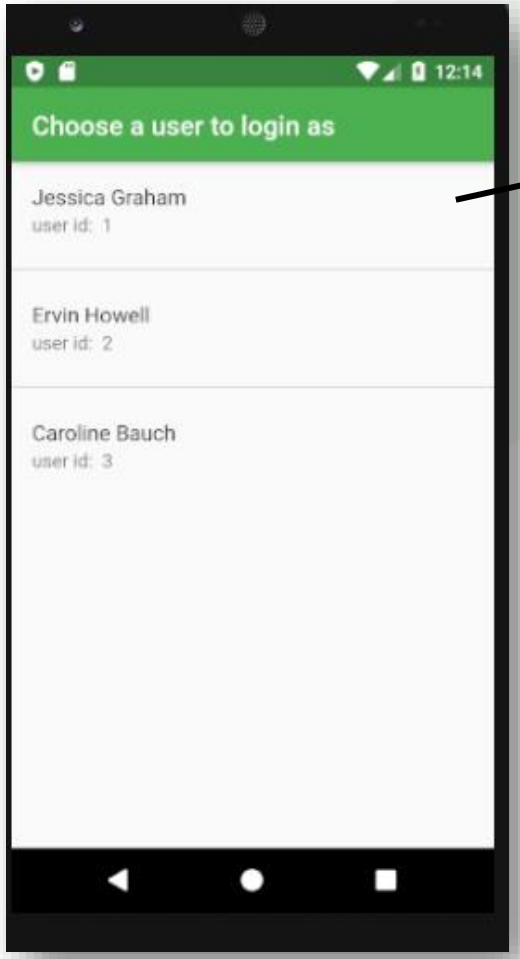
Demo

Provider State Management

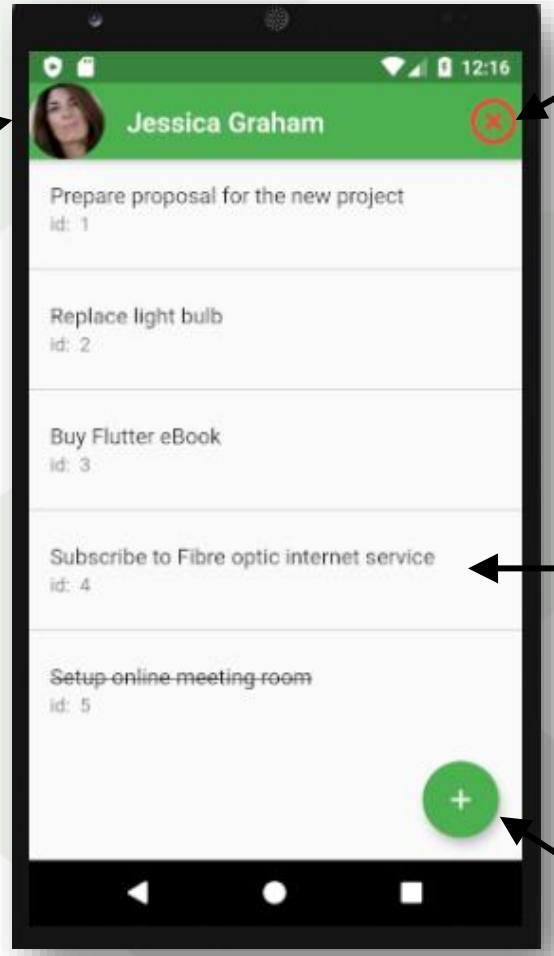
Provider State Management

What we are going to build

LoginAsScreen



TodoListScreen



Logout

The todo list displayed for the active user

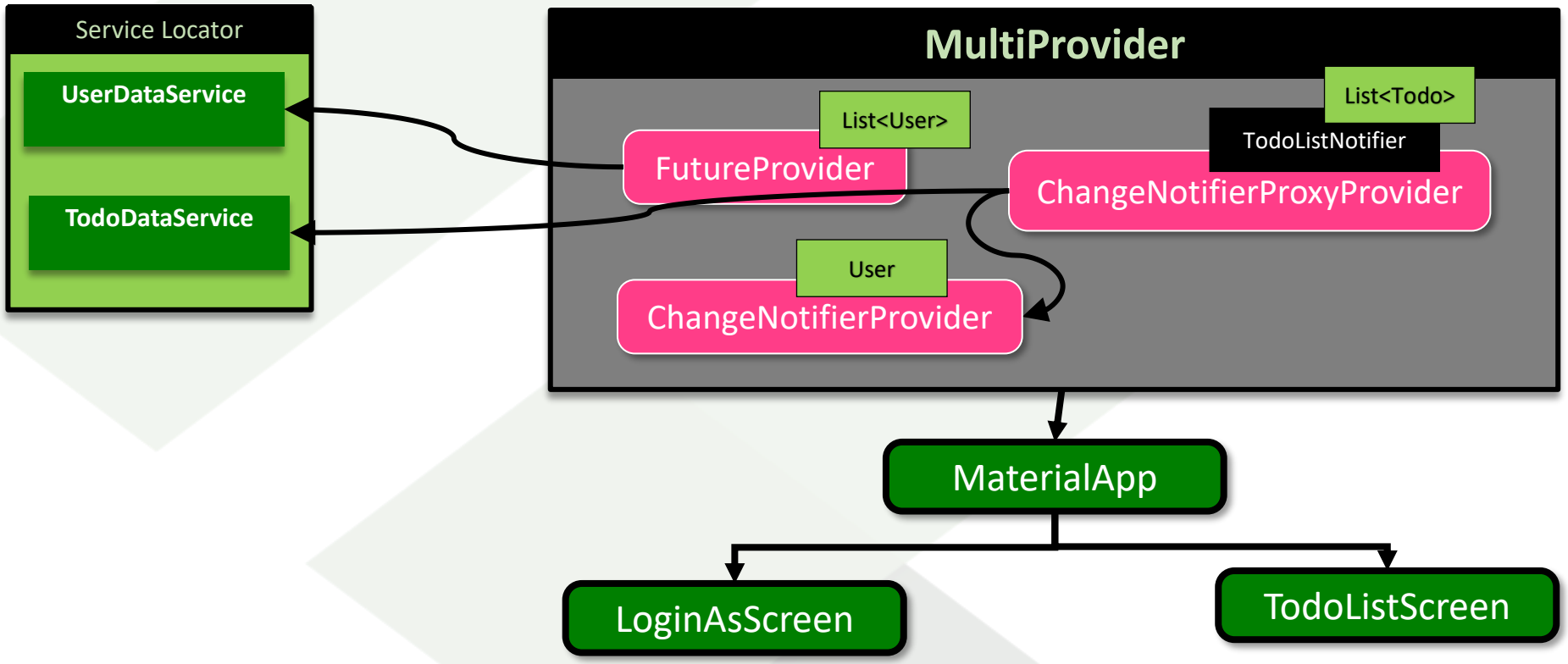
onTap: Toggle status

onLongPressed: Delete the todo item

Add a new todo item

Provider State Management (2)

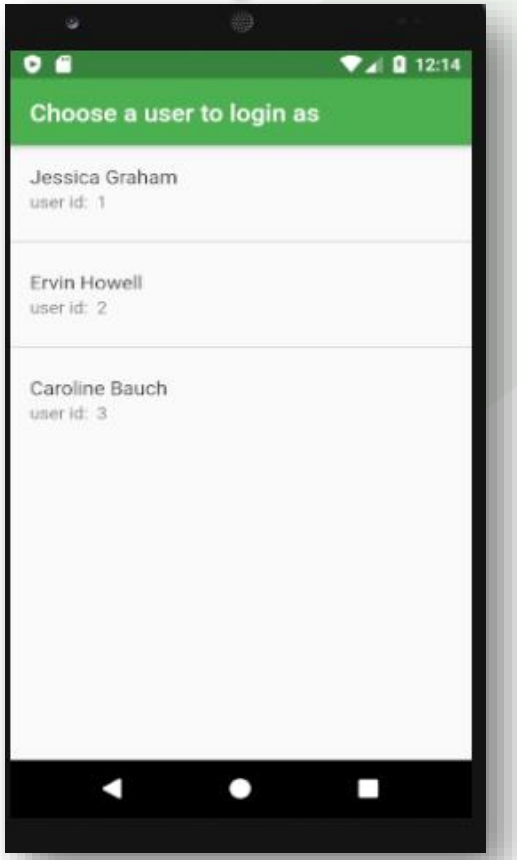
How we are going to build it



Provider State Management (3)

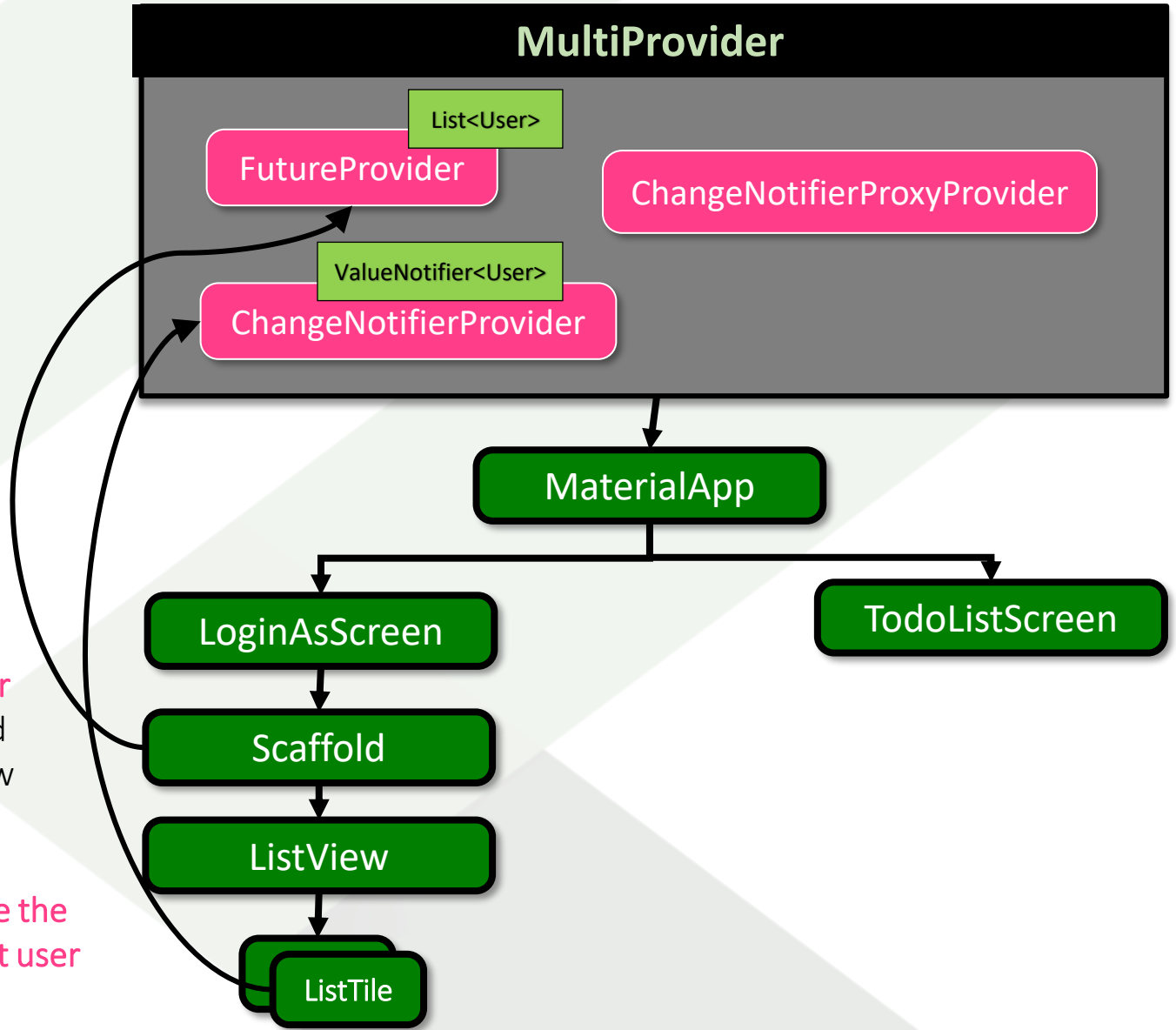
How we are going to build it

LoginAsScreen



get the user list, to build the ListView

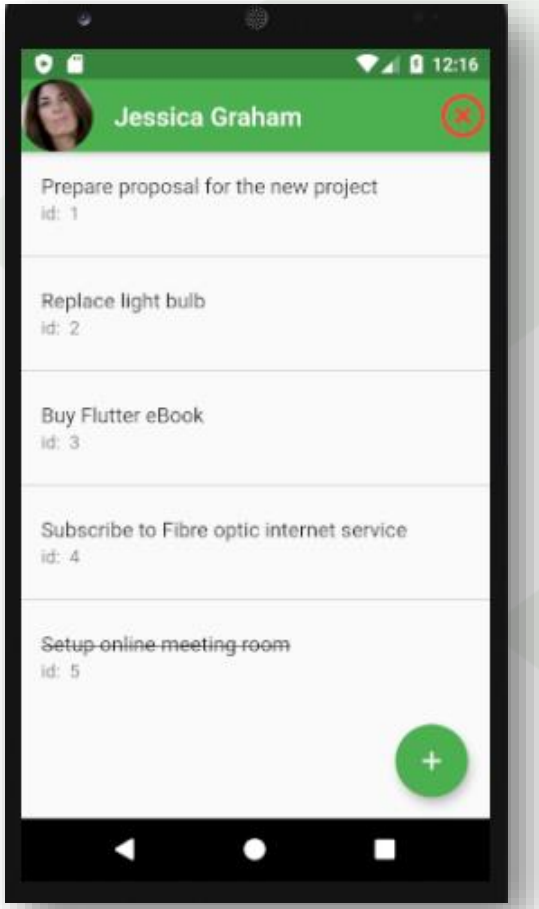
Change the current user

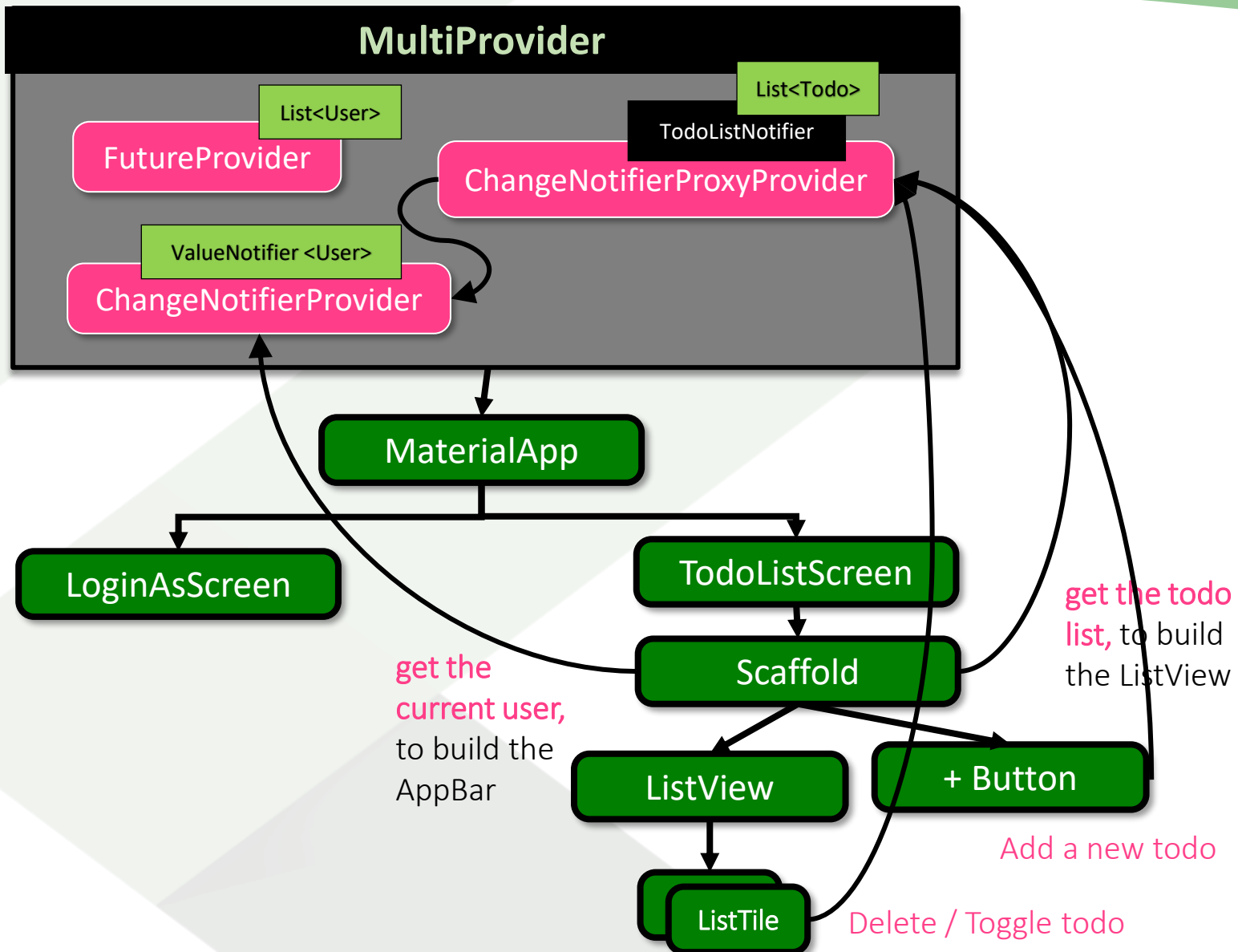


Provider State Management (4)

How we are going to build it

TodoListScreen





The Codebase

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"
  }
]
```

Define TodoListNotifier Class

Setup the FutureProvider and MultiProvider

Setup the ChangeNotifierProvider

Setup the ChangeNotifierProxyProvider

Optimize Widget Rebuilding

Summary

- What are providers?
- Setting Up and Consuming Providers
- Types of Providers
- Provider as DI and State Management