Mobile Development :

10 : Building the Beta Version : Part 1

Architecture, Backends, Starting the Dev

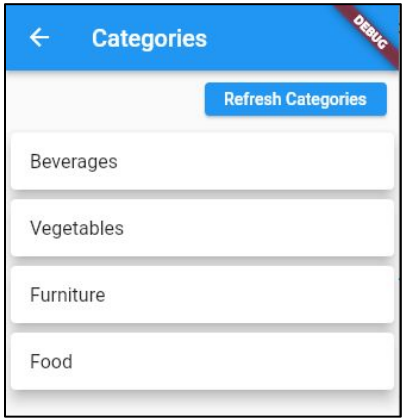
Professor Imed Bouchrika
National School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

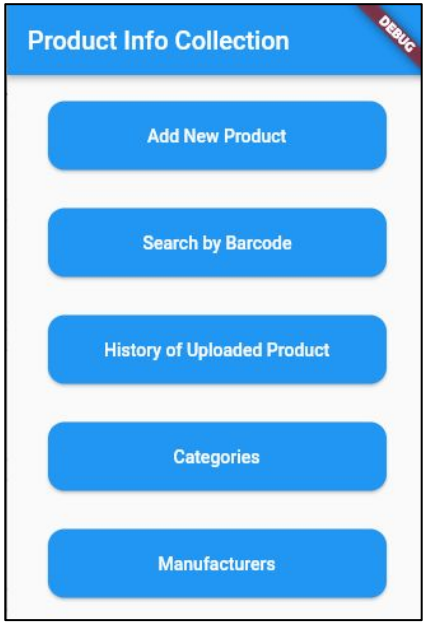
Outline :

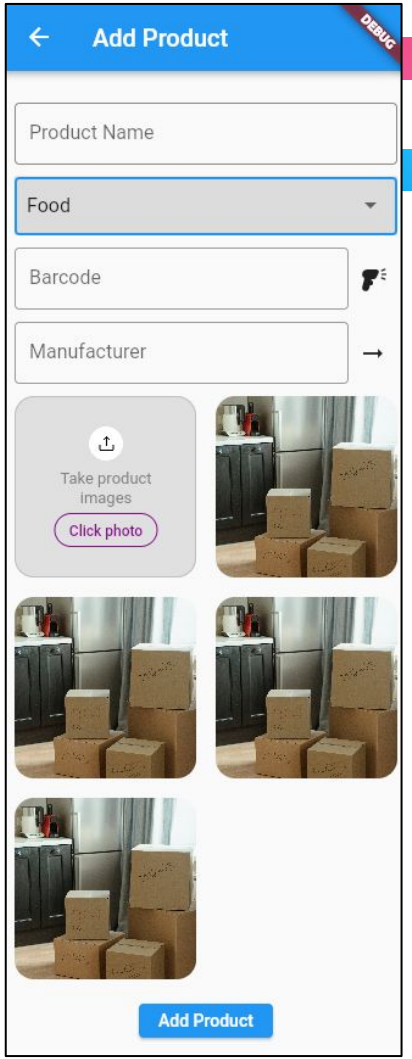
- **Case Study : from MVP to Beta**
  - ***Description : Product Info Collection***
  - ***Architecture of Components***
  - ***Use of FutureBuilder***
  - ***Databases : Remote & Local***
  - ***Connecting to Backends***
  - ***Synching & Background Services***
  - ***Navigation between Screens***

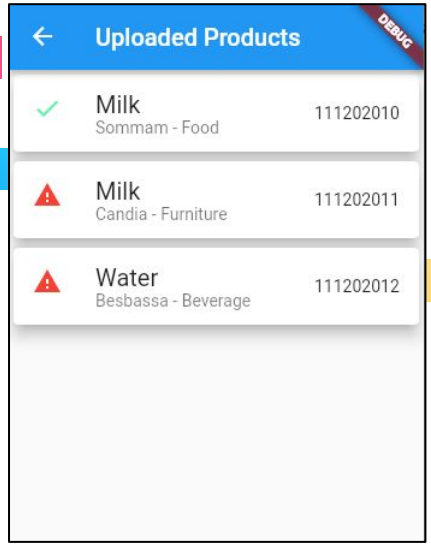
From MVP to Beta Version Case Study

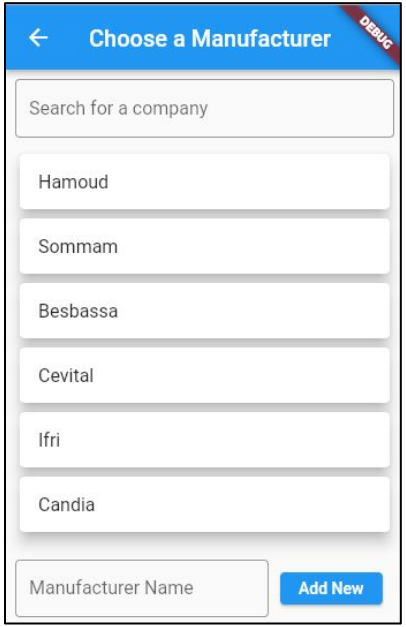
- **App for Collecting Product Descriptions**
  - The customer needs an App for their employees to collect information about products.
  - Information includes:
    - *Product name*
    - *Barcode*
    - *Product Manufacturer*
    - *Product Image**s***
  - The app should work in offline mode. Preferably, when there is internet connection, **collected data can be uploaded** and sync can be performed.
  - The staff, who are the app users can search for a product by barcode to see all product information being **uploaded by them or by other staff.**

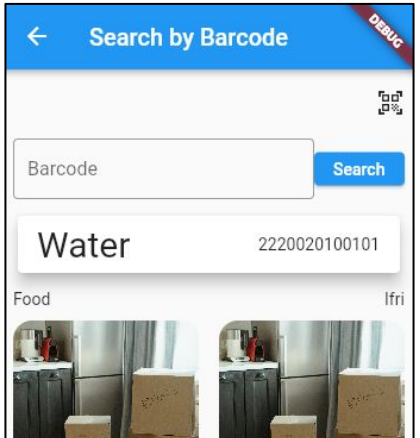

Categories
DEBUG
Refresh Categories
Beverages
Vegetables
Furniture
Food

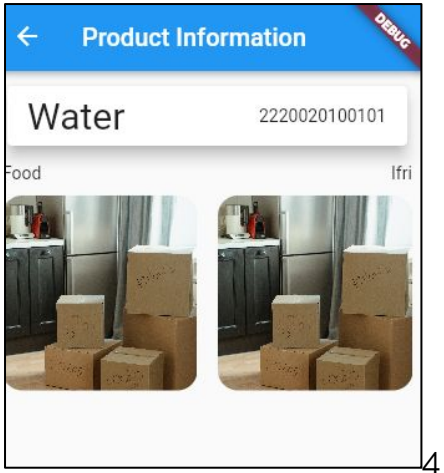

Product Info Collection
DEBUG
Add New Product
Search by Barcode
History of Uploaded Product
Categories
Manufacturers


Add Product
DEBUG
Product Name
Food
Barcode
Manufacturer
Take product images
Click photo
Add Product


Uploaded Products
DEBUG
Milk
Sommam - Food
111202010
Milk
Candia - Furniture
111202011
Water
Besbassa - Beverage
111202012

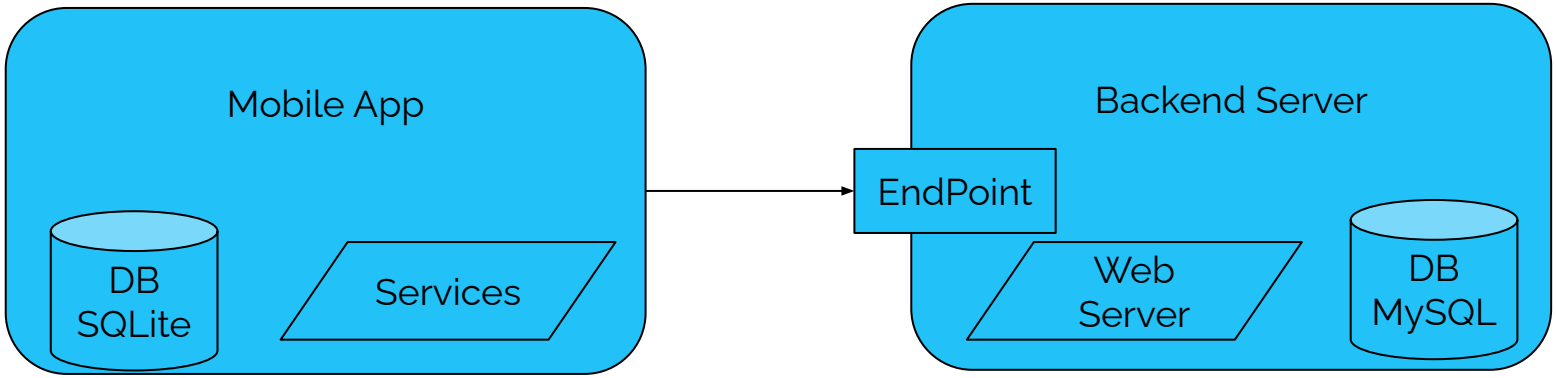

Choose a Manufacturer
DEBUG
Search for a company
Hamoud
Sommam
Besbassa
Cevital
Ifri
Candia
Manufacturer Name
Add New


Search by Barcode
DEBUG
Barcode
Search
Water
2220020100101
Food
Ifri


Product Information
DEBUG
Water
2220020100101
Food
Ifri

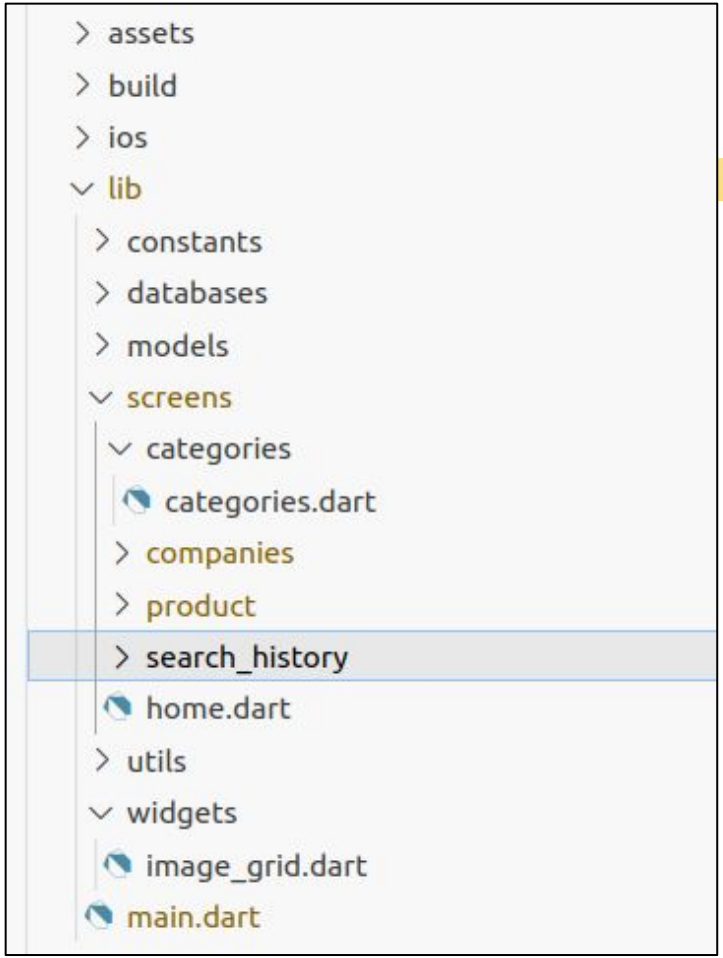
From MVP to Beta Version Case Study

- **Architecture of the Solution (not App)**



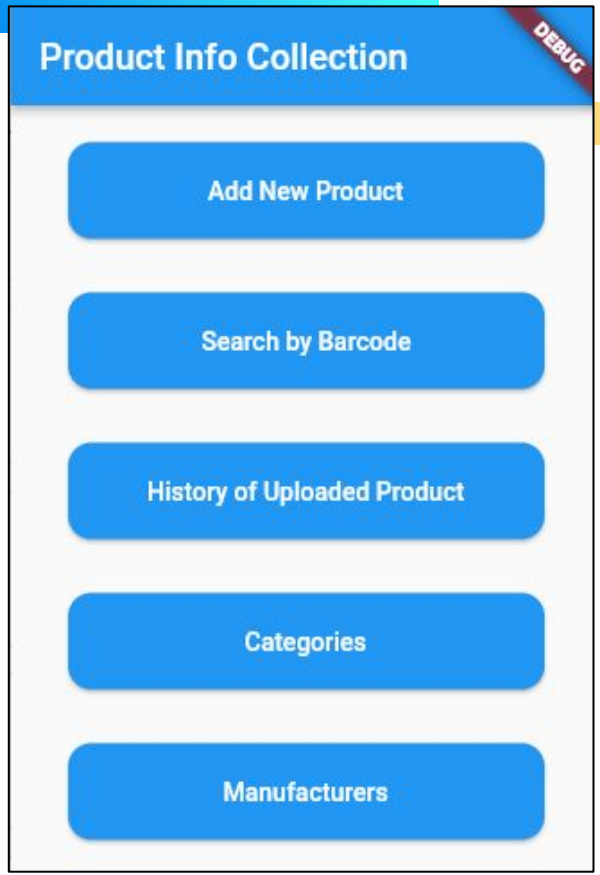
From MVP to Beta Version Case Study

- **Structuring of the Project**



From MVP to Beta Version Case Study

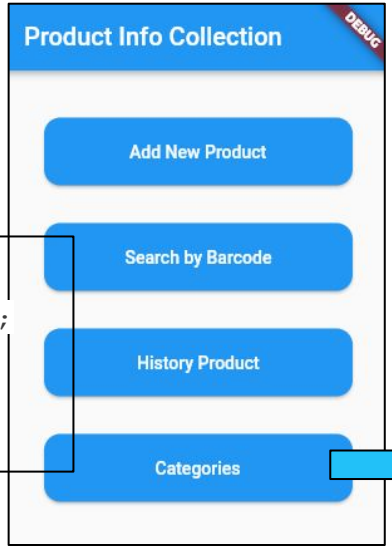
- **HomeScreen**
  - Flutter Widget Tree :
    - Scaffold:
      - Appbar
      - child: Column
        - children: Buttons+
          - onPressed:

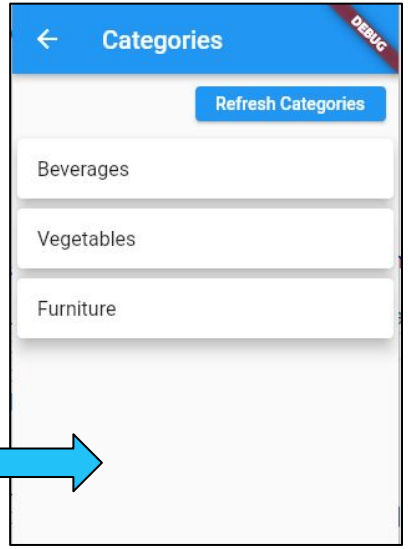


From MVP to Beta Version Case Study

- **Categories Screen**

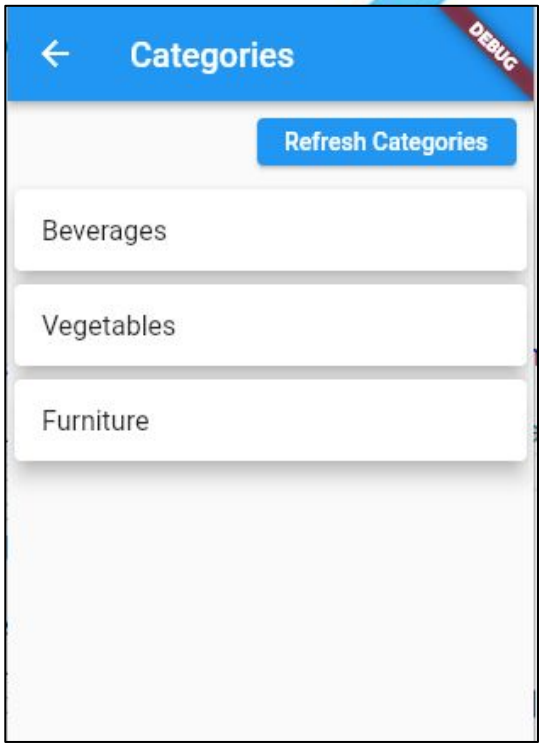
```
onPressed: () {
Navigator.of(context).pushNamed(Categories.pageRoute);
},
```





From MVP to Beta Version Case Study

- **Categories Screen**
  - Scaffold:
    - Appbar
    - Column
      - Row
        - Spacer
        - Button
      - Expanded
        - ListView
          - Card



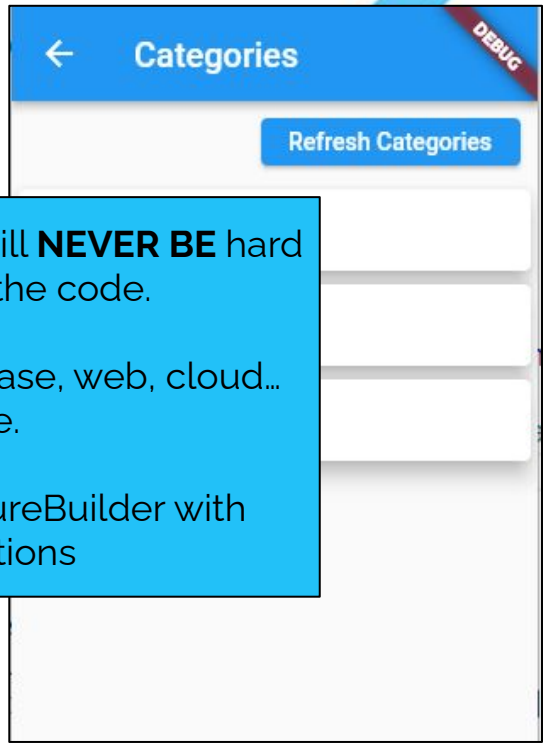
From MVP to Beta Version Case Study

- **Categories Screen**
  - Scaffold:
    - Appbar
    - Column
      - Row
        - Spacer
        - Button
      - Expanded
        - ListView
          - Card

For certain : Categories Data will **NEVER BE** hard coded as variables in the code.

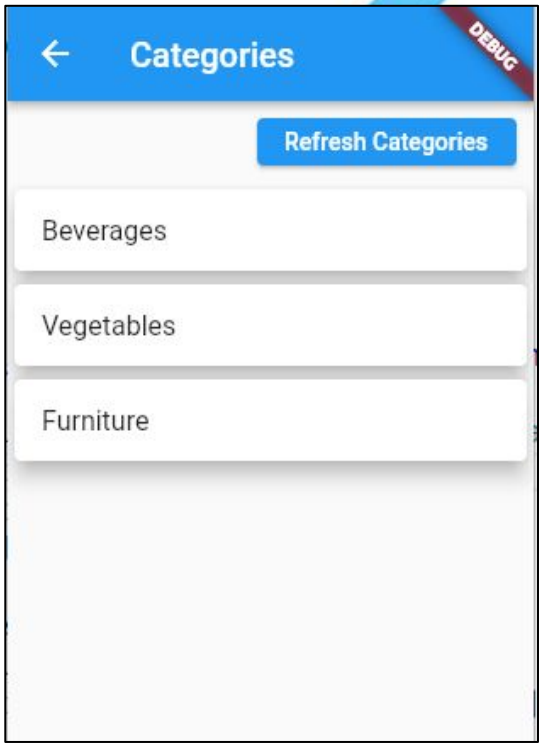
It will be retrieved from database, web, cloud… after sometime.

⇒ Better to always use FutureBuilder with async/await functions

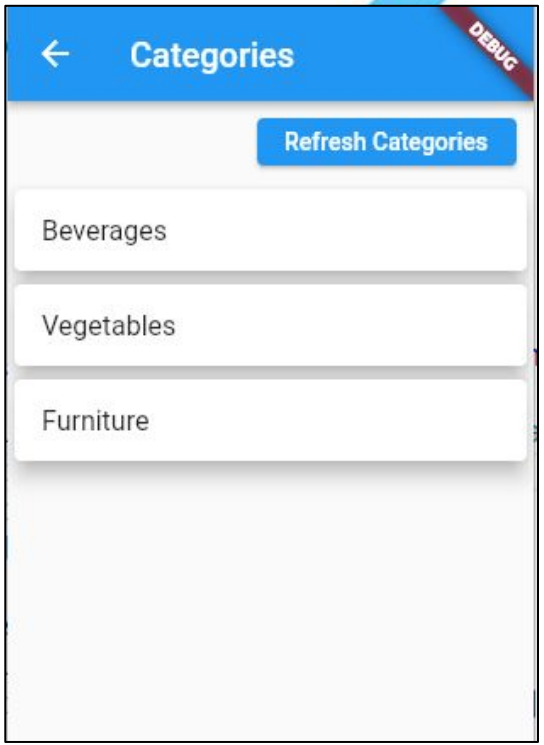


From MVP to Beta Version Case Study

- **Categories Screen : Use of Future Builder**
  - Scaffold:
    - Appbar
    - Column
      - Row
        - Spacer
        - Button
      - Expanded
        - **FutureBuilder**
          - ListView
            - Card

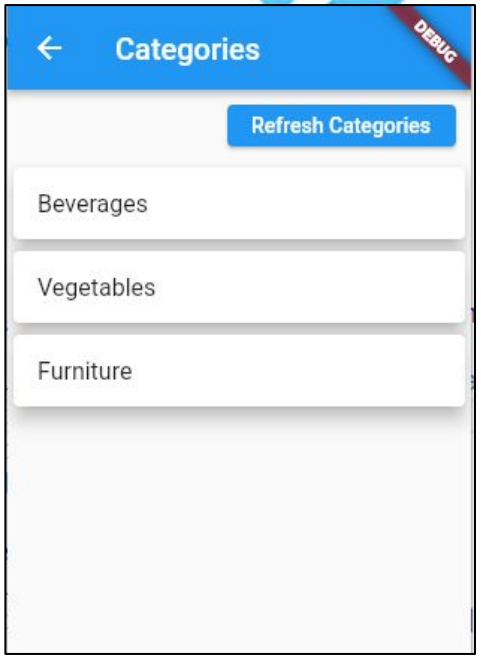


```
@override
Widget build(BuildContext context) {
  Future<List> categories = getListCategories();
  return Scaffold(
      ...
       Expanded(
              child: FutureBuilder(
            future: categories,
            builder: _build_list_categories,
          ))
        ],
      )));
}
Future<List<Map>> getListCategories() async {
  return [{'id': 1, 'name': 'Beverages'},
          {'id': 2, 'name': 'Vegetables'},
          {'id': 3, 'name': 'Furniture'},
          {'id': 4, 'name': 'Food'},   ];
}
Widget  build list categories(BuildContext context, AsyncSnapshot snapshot) {
  if (snapshot.hasData) {
    List<Map> items = snapshot.data as List<Map>;
    return ListView.builder(
      itemCount: items.length,
      itemBuilder: (context, index) {
        return Card(
            elevation: 10,
            child: ListTile(
              title: Text(items[index]['name']),
            ));
      },
    );
  } else if (snapshot.hasError) { return Text("${snapshot.error}"); }
  return CircularProgressIndicator();
}
```



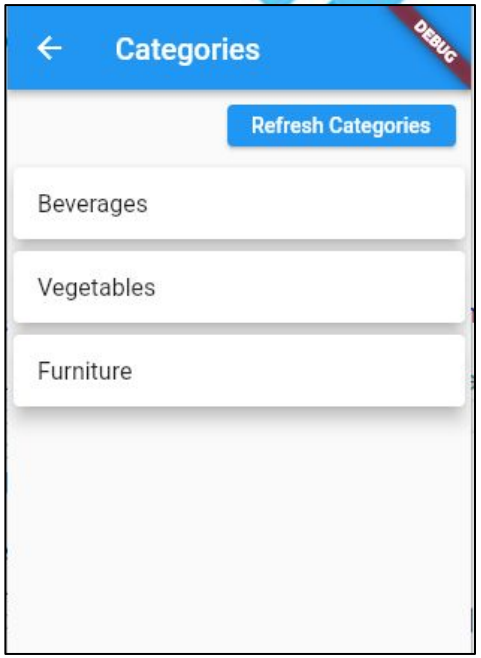
From MVP to Beta Version Case Study

- **Categories Screen**
  - MVP : Categories Data are hardcoded:
    - Shall we keep them hardcoded ?
    - Shall we get them everytime from the web
    - Shall we store them into a database ?
    - Shall we store them into a cloud database ?



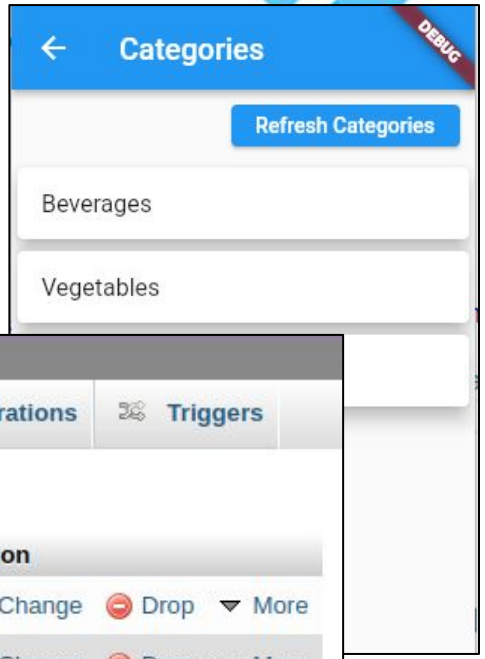
From MVP to Beta Version Case Study

- **Categories Screen**
  - MVP : Categories Data are hardcoded:
    - **We get them from the web from time to time**
    - **We store them into the database +**
    - **Future builder is used**
    - In the context of this app :
      - it is not frequent for categories to change.



From MVP to Beta Version Case Study

- **Categories Screen : Remote Database**
  - Maintain using even phpMyAdmin, no need for for Backend UI at this stage…



Server: 127.0.0.1:3306 » Database: u229483663_productinfoapp » Table: categories

Browse | Structure | SQL | Search | Insert | Export | Import | Operations | Triggers

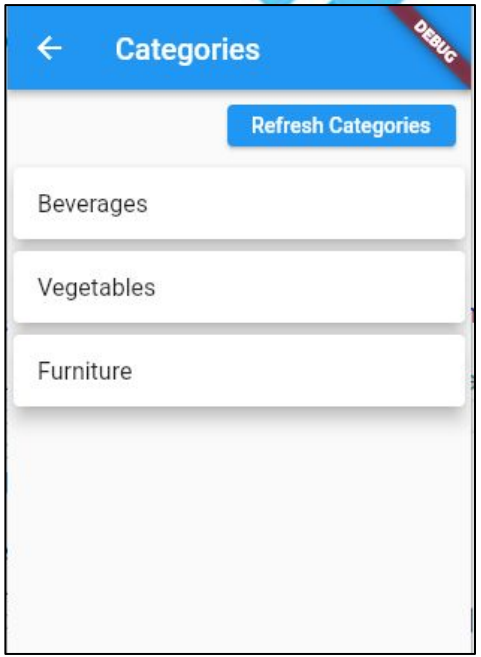
Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	name	varchar(250)	utf8mb4_unicode_ci		No	*None*			Change Drop More

From MVP to Beta Version Case Study

- **Categories Screen : Simple API EndPoint via PHP**
  - API Endpoint :
    - **categories.get**

```php
<?php

    include("init.php");

    switch($vars['action']){

        case "categories.get":{

            $items = $db->query('SELECT * FROM categories')->fetchAll();
            echo json_encode($items);
            exit;

        }break;
    }

?>
```

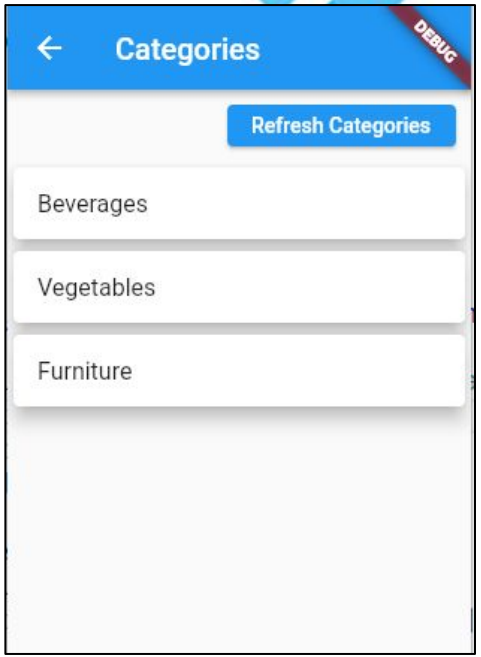


From MVP to Beta Version Case Study

- **Categories Screen : Simple API EndPoint via PHP**

https://productinfoapp.startsoftware.dev/?action=categories.get

[{"id":1,"name":"Food"},{"id":2,"name":"Fruits"},
{"id":3,"name":"Bread"},{"id":4,"name":"Beverages"},
{"id":5,"name":"Laundry"},{"id":6,"name":"Cakes"}]

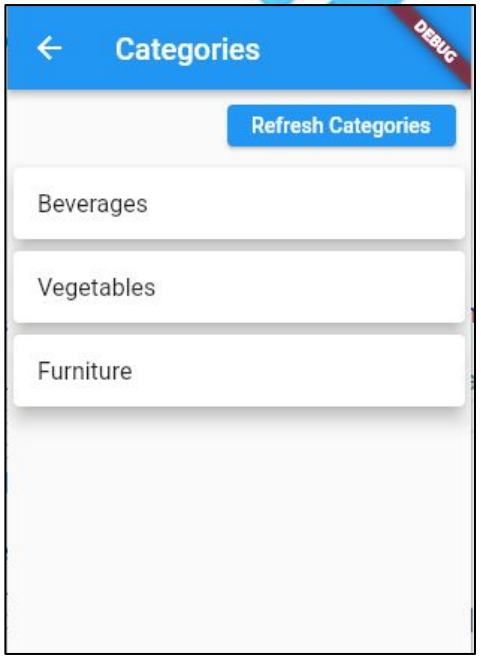


From MVP to Beta Version Case Study

- **Categories Screen : Simple API EndPoint**
  - Maintain using even phpMyAdmin, no need for for E

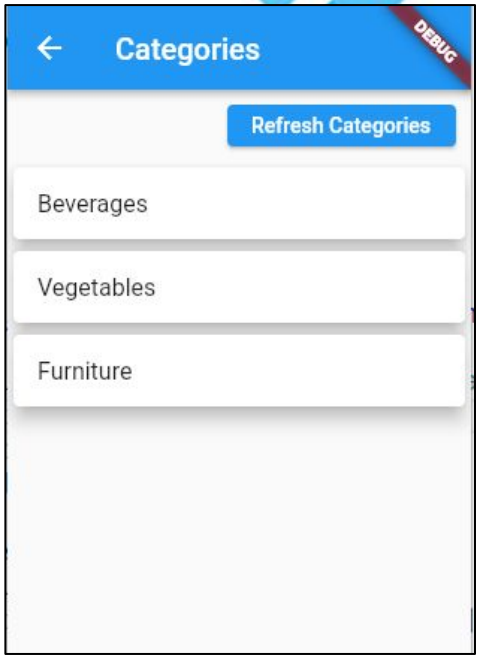
Better to store them locally to avoid calling the server everytime we add a product ?

Work even in <u>offline</u> conditions



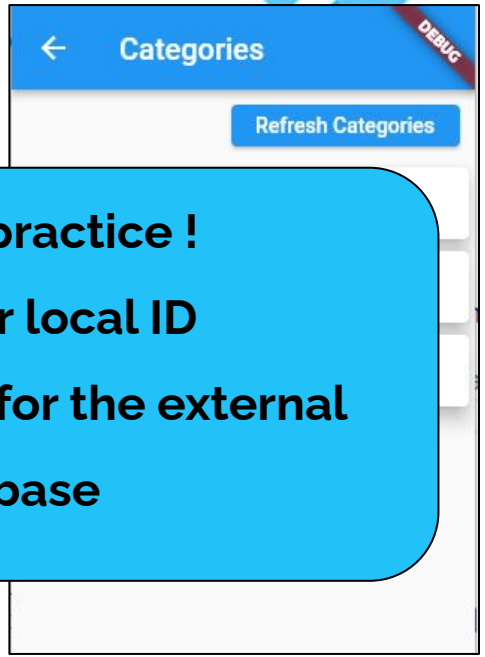
From MVP to Beta Version Case Study

- **Categories Screen : Local Database Schema**
  - categories
    - id : INTEGER
    - name : TEXT



From MVP to Beta Version Case Study

- **Categories Screen : Local Database Schema**
  - categories
    - id : INTEGER
    - remote_id : INTEGER
    - name : TEXT
    - *create_time: TEXT*
    - *update_time : TEXT*



A better practice !

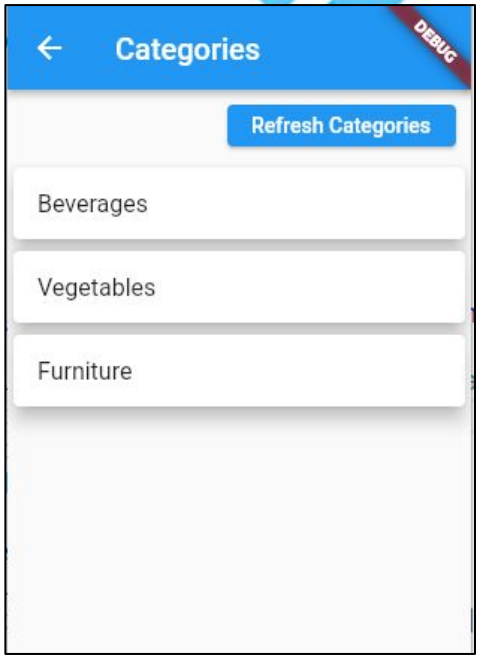
set id : for local ID

and remote_id for the external database

From MVP to Beta Version Case Study

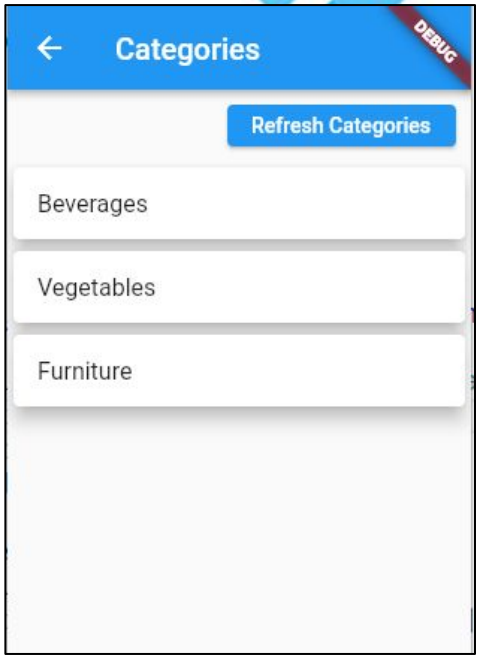
- **Categories Screen : Local Database in Flutter**
  - **Step 1** :
    - Install the sqflite + path plugins

```
flutter pub add sqflite path
```

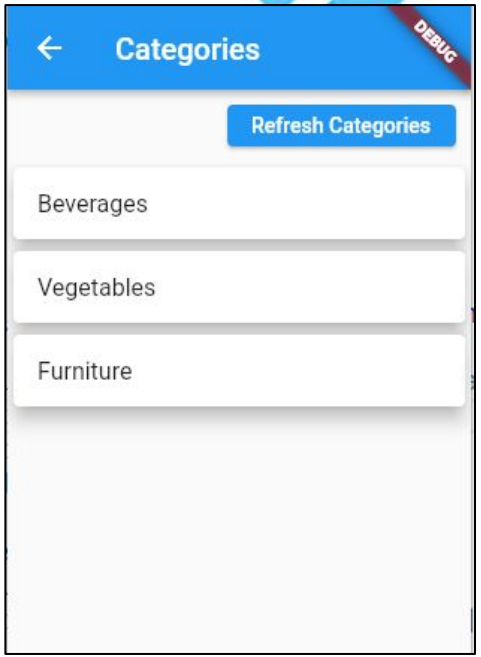


From MVP to Beta Version Case Study

- **Categories Screen : Local Database in Flutter**
  - **Step 2** :
    - Create the DBHelper class

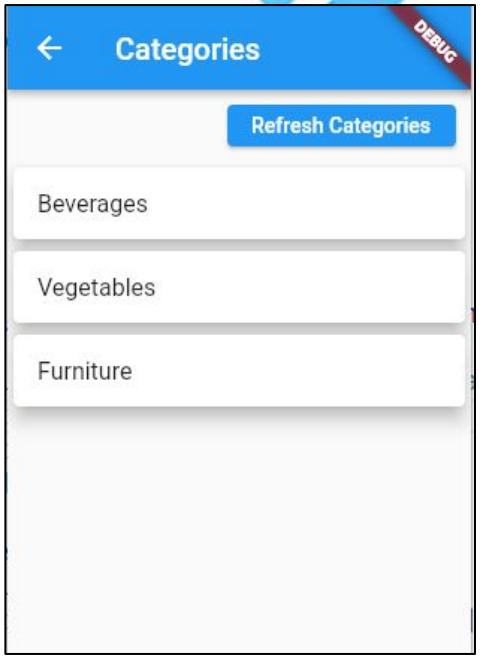


```dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';

class DBHelper {
 static const _database_name = "PRODUCT_INFO_V1.db";
 static const  database_version = 1;
 static var database;

 static Future getDatabase() async {
   if (database != null) {
     return database;
   }
   database = openDatabase(
     join(await getDatabasesPath(), _database_name),
     onCreate: (database, version) {
       database.execute('''
         CREATE TABLE  categories (
             id INTEGER PRIMARY KEY AUTOINCREMENT,
             remote_id INTEGER,
             name TEXT,
             create_date TEXT
           )
       ''');
     },
     version: _database_version,
     onUpgrade: (db, oldVersion, newVersion) {
       //do nothing...
     },
   );
   return database;
 }
}
```

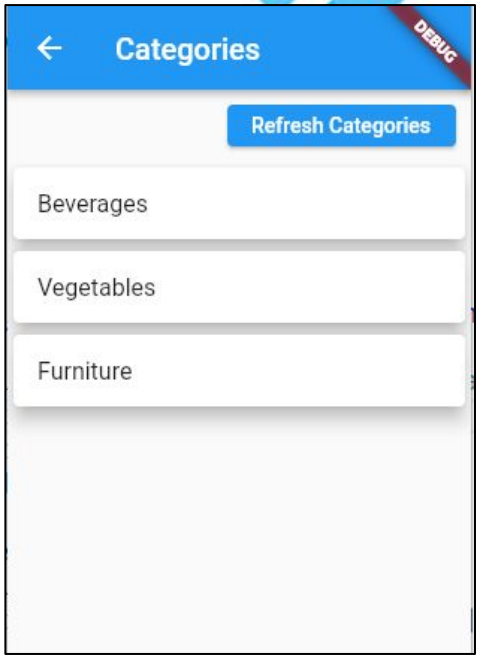


From MVP to Beta Version Case Study

- **Categories Screen : Local Database in Flutter**
  - **Step 3** :
    - Create the SQL utility class for Categories as db_category.dart

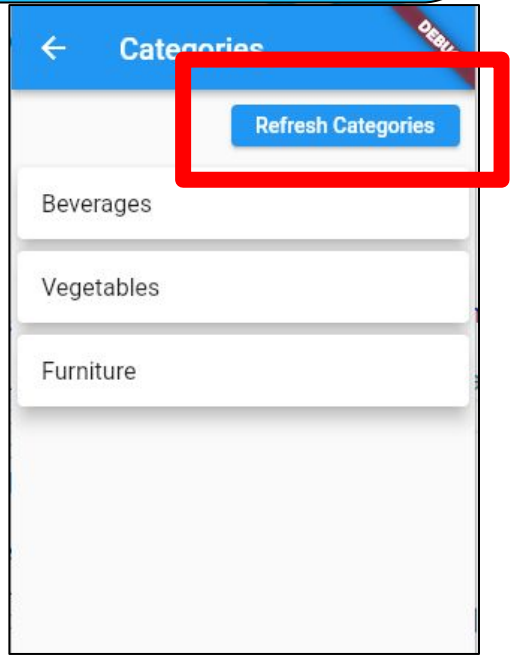


```dart
import 'package:sqflite/sqflite.dart';
import 'DBHelper.dart';

class DBCategory {
 static final tableName = 'categories';

 static Future<List<Map<String, dynamic>>> getAllCategories() async {
   final database = await DBHelper.getDatabase();

   return database.rawQuery('''SELECT id , name, remote_id from ${tableName} ''');
 }
 static Future<bool> syncCategories(
     List<Map<String, dynamic>> remote_data) async {
    //Some code here to sync
   return true;
 }
 static Future<bool> updateRecord(int id, Map<String, dynamic> data) async {
   final database = await DBHelper.getDatabase();
   database.update(tableName, data, where: "id=?", whereArgs: [id]);
   return true;
 }

 static Future<bool> insertRecord(Map<String, dynamic> data) async {
   final database = await DBHelper.getDatabase();
   database.insert(tableName, data,
       conflictAlgorithm: ConflictAlgorithm.replace);
   return true;
 }
 static Future<bool> deleteRecord(int id) async {
   final database = await DBHelper.getDatabase();
   database.rawQuery("""delete from  ${tableName}  where id=?""", [id]);
   return true;
 }
}
```



```dart
import 'package:sqflite/sqflite.dart';
import 'DBHelper.dart';

class DBCategory {
 static final tableName = 'categories';

 static Future<List<Map<String, dynamic>>> getAllCategories() async {
   final database = await DBHelper.getDatabase();

   return database.rawQuery('''SELECT id , name, remote_id from ${tableName}
 }
 static Future<bool> syncCategories(
     List<Map<String, dynamic>> remote_data) async {
    //Some code here to sync
   return true;
 }
 static Future<bool> updateRecord(int id, Map<String, dynamic> data) async {
   final database = await DBHelper.getDatabase();
   database.update(tableName, data, where: "id=?", whereArgs: [id]);
   return true;
 }

 static Future<bool> insertRecord(Map<String, dynamic> data) async {
   final database = await DBHelper.getDatabase();
   database.insert(tableName, data,
       conflictAlgorithm: ConflictAlgorithm.replace);
   return true;
 }
 static Future<bool> deleteRecord(int id) async {
   final database = await DBHelper.getDatabase();
   database.rawQuery("""delete from  ${tableName}  where id=?""", [id]);
   return true;
 }
}
```

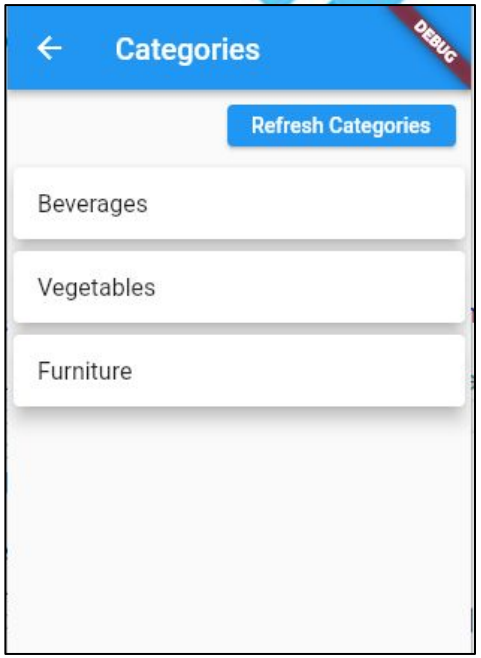
Let's get the data via manual click for now ?



From MVP to Beta Version Case Study

- **Categories Screen : Local Database in Flutter**
  - **Step 4** :
    - Synch Data between Local and Remote
      - onPressed
        - Connect to API EndPoint
        - Decode JSON into Map Object
        - Call the DB Function :

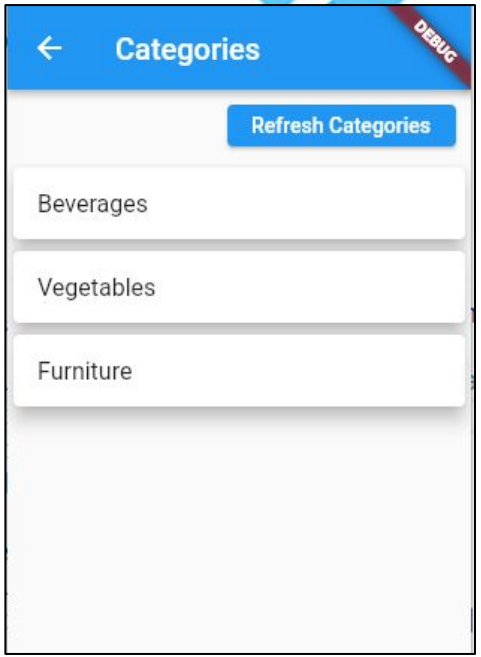
```
static Future<bool> syncCategories(
```



```
  child: ElevatedButton(
         onPressed: () async {
                    await service_sync_categories();
                    setState(() {});
             },
        child: Text("Refresh Categories"))),
```

utils/categories.dart

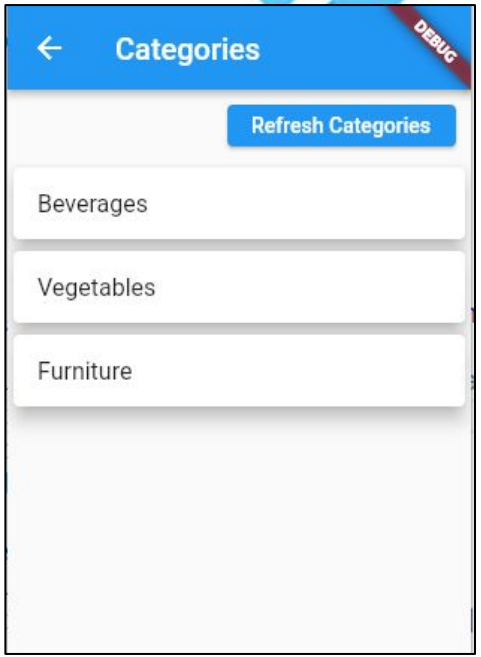
```
Future<bool> service_sync_categories() async {
 print("Running Cron Service to get Categories");
 List? remote_data = await endpoint_api_get_categories();

 if (remote data != null) {
   await DBCategory.syncCategories(remote_data as List<Map<String,
dynamic>>);
   return true;
 }
 return false;
}
```



flutter pub add http

```
import 'package:http/http.dart' as http;
import 'dart:convert';

Future<List<Map<String, dynamic>>?> endpoint_api_get_categories() async
{
 //This can be improved by placing API endpoints into a constant dart
file
 try {
   final response = await http.get(Uri.parse(

'https://productinfoapp.startsoftware.dev/?action=categories.get'));
   print("${response.statusCode}");
   if (response.statusCode == 200) {
     List<Map<String, dynamic>> ret =
         List<Map<String, dynamic>>.from(jsonDecode(response.body));
     return ret;
   }
 } catch (error) {
   print("Error ${error.toString()}");
   return null;
 }
 return null;
}
```

utils/categories.dart



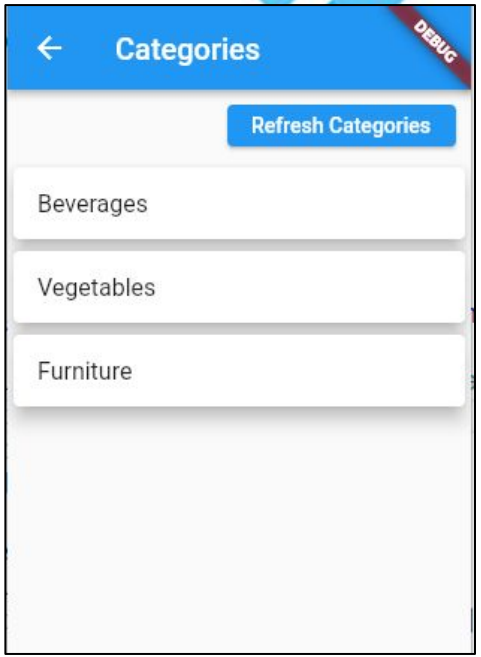
```
static Future<bool> syncCategories(
    List<Map<String, dynamic>> remote_data) async {
  Li
                                                  d']});
  fo
  return true;
}
```

The easy way to sync the local and remote table:

Delete all local lines

Insert them again and use Remote_ID as the ID

Easy, as we don't edit or add categories from the phone.

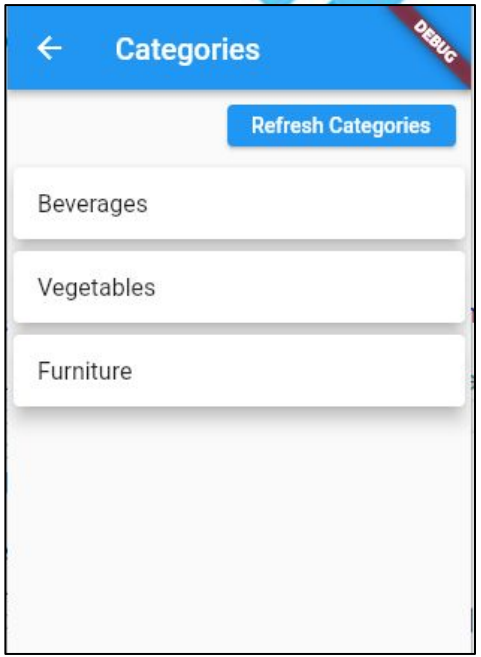


```
static Future<bool> syncCategories(
```

- Getting and Synching List of Categories
  - For category in Remote Categories
    - If inside Local DB:
      - Add Local to Seen_locals
      - Update locally
    - Else
      - INSERT Locally
  - For Locals not in Seen_locals,
    - Remove from DB

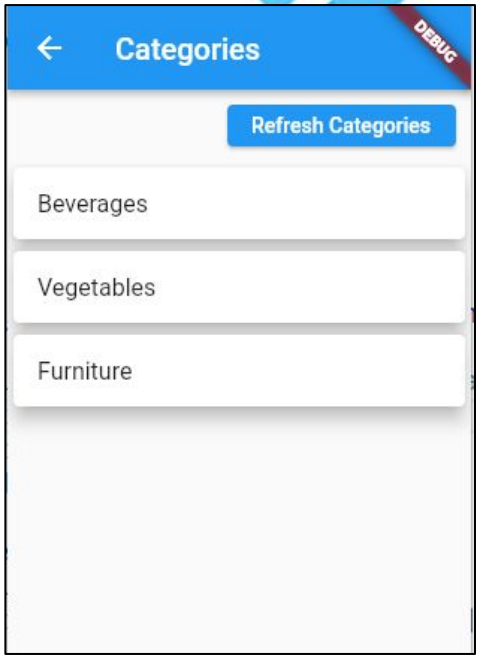
```
}
```

Easy, as we don't edit or add categories from the phone.



```dart
static Future<bool> syncCategories(
    List<Map<String, dynamic>> remote_data) async {
  List local_data = await getAllCategories();
  Map index_remote = {};
  List local_ids = [];
  for (Map item in local_data) {
    index_remote[item['remote_id']] = item['id'];
    local_ids.add(item['id']);
  }

  for (Map item in remote_data) {
    if (index_remote.containsKey(item['id'])) {
      int local_id = index_remote[item['id']];
      await updateRecord(local_id, {'name': item['name']});
      local_ids.remove(local_id);
    } else {
      await insertRecord({'name': item['name'], 'remote_id': item['id']});
    }
  }
  //Remote Local Categories...
  //There is a RISK ? in case items pending with old data?
  for (int local_id in local_ids) await deleteRecord(local_id);
  return true;
}
```

Easy, as we don't edit or add categories from the phone.

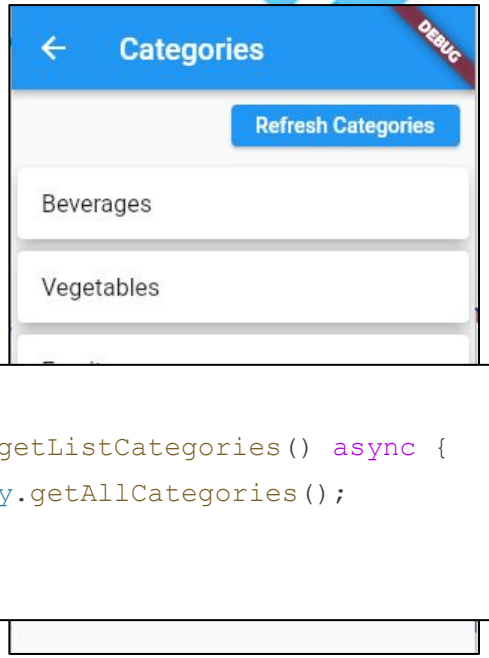


From MVP to Beta Version Case Study

- **Categories Screen : Local Database in Flutter**
  - **Step 4** :
    - Integrate DB Data with the screens:

```
Future<List<Map>> getListCategories() async {
  return [
    {'id': 1, 'name': 'Beverages'},
    {'id': 2, 'name': 'Vegetables'},
    {'id': 3, 'name': 'Furniture'},
    {'id': 4, 'name': 'Food'},
  ];
}
```

```
Future<List<Map>> getListCategories() async {
  return DBCategory.getAllCategories();
}
```



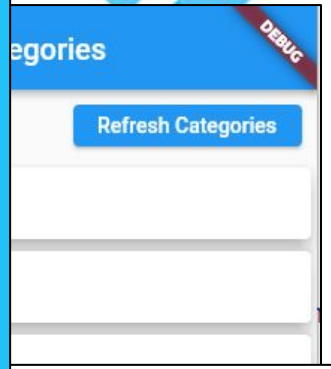
From MVP to Beta Version Case Study

- **Categories S**
  - **Step 4** :
    - Integ

Better to sync not using the “manual button click”,

But using a background service to run every hour ? day ?. ..

```
Future<List<Map>> getList
  return [
    {'id': 1, 'name': 'Bev
    {'id': 2, 'name': 'Veget
    {'id': 3, 'name': 'Furniture'},
    {'id': 4, 'name': 'Food'},
  ];
}
```



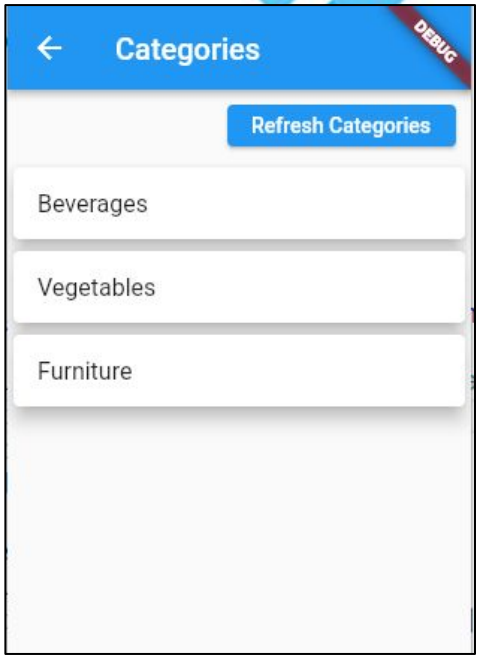

```
ategories() async {
  return DBCategory.getAllCategories();
}
```

From MVP to Beta Version Case Study

- **Categories Screen : Local Database in Flutter**
  - **Step 5** :
    - Cron Service to auto sync categories

flutter pub add cron

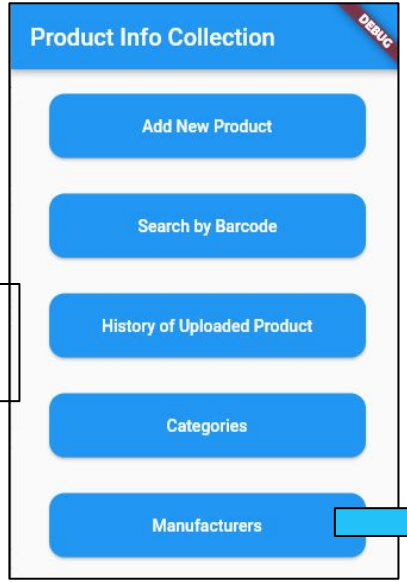
```
void main() {
 WidgetsFlutterBinding.ensureInitialized();

 final cron = Cron();
 cron.schedule(Schedule.parse('*/5 * * * *'), () async {
   service_sync_categories();
 });

 runApp(const MainApp());
}
```

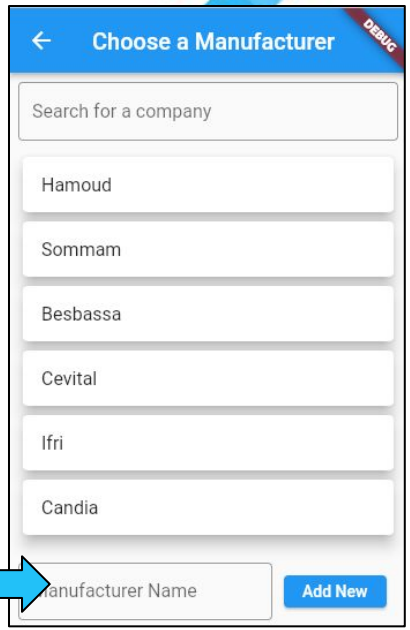


From MVP to Beta Version Case Study

- **Manufacturer Screen**
  - Navigation
    - No arguments is passed

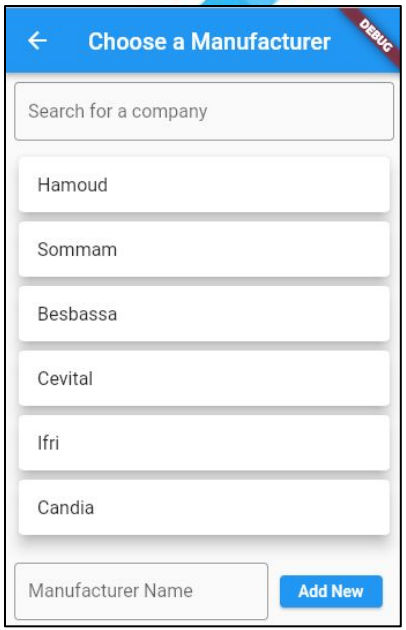
```
Navigator.of(context).pushNamed(Manufacturer.pageRoute);
```





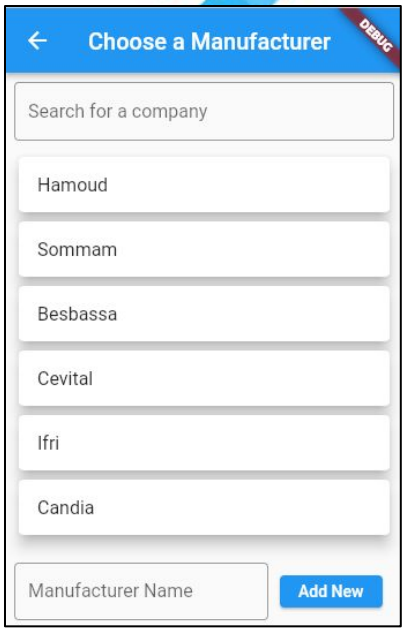
From MVP to Beta Version Case Study

- **Manufacturer Screen**
  - Widgets Tree
    - Column
      - TextField
      - **FutureBuilder**
        - ListView
      - Row
        - TextField
        - Button



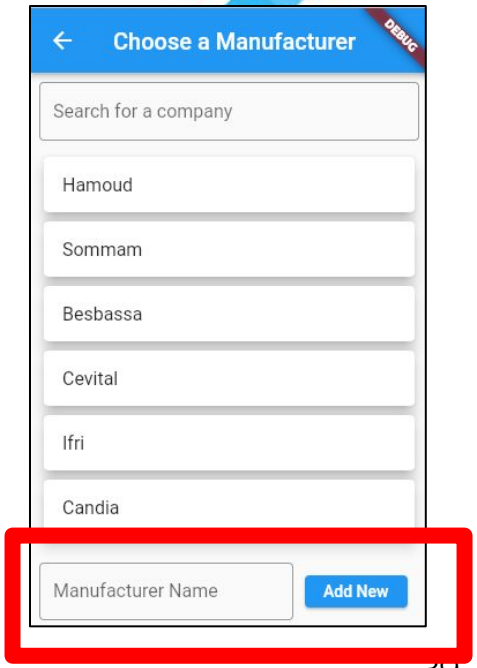
From MVP to Beta Version Case Study

- **Manufacturer Screen**
  - Loading of data :
    - Same as Categories, load from web → Local DB
    - Allow user**S** to add new companies ?
    - Curate and Clean data ?
      - Beyond the scope of this course
      - Hamod vs Hamoud vs حمود vs Hmoud or Even 2 Hamoud (added by two different users)



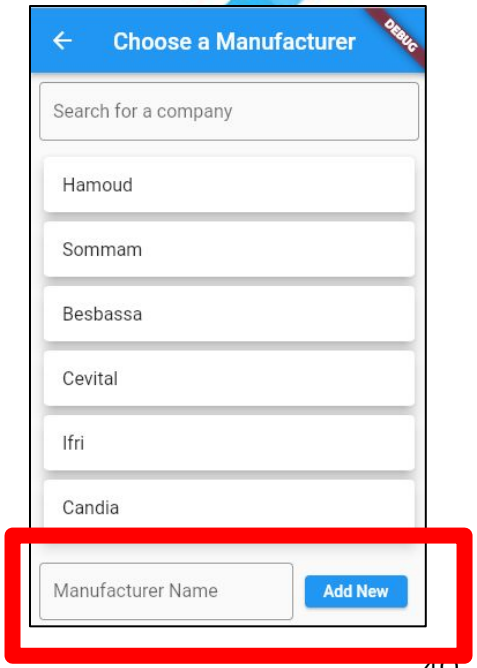
From MVP to Beta Version Case Study

- **Manufacturer Screen : Adding Data**
  - Technical Procedure to Add data:
    - Shall we add it **synchronously** to the remote ?
      - Remember it needs to work in Offline
    - Add it locally only and at a later time when there is an internet connection ⇒
      - **SYNC** but **TWO WAYS**



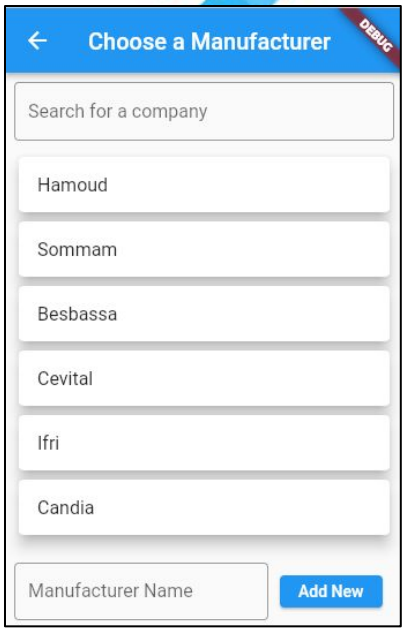
From MVP to Beta Version Case Study

```
ElevatedButton(
     onPressed: () async {
             await DBCompany.insertRecord(
                {'to_add': 1, 'name': _tx_name_add_controller.text});
                    _tx_name_add_controller.text = '';
              setState(() {});
        },
     child: const Text(
                "Add New",
                ),
```



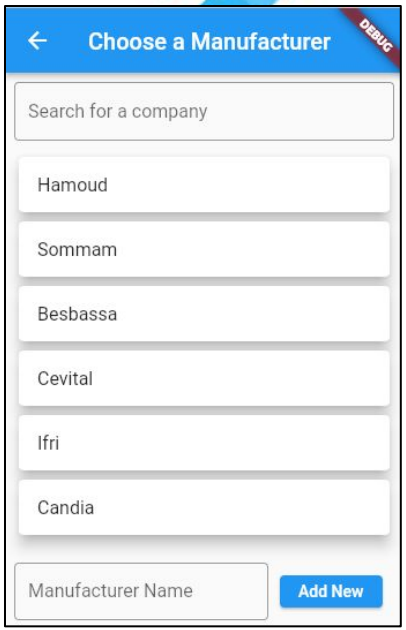
From MVP to Beta Version Case Study

- **Manufacturer Screen : Synching Data**
  - Remote Database Schema
    - **companies:**
      - id
      - name



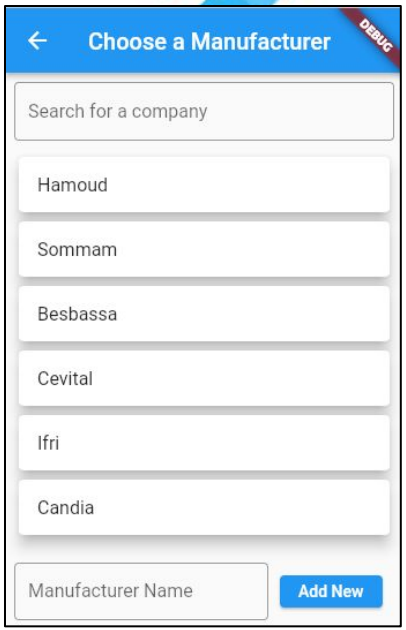
From MVP to Beta Version Case Study

- **Manufacturer Screen : Synching Data**
  - Local Database Schema
    - **companies:**
      - id : INTEGER PRIMARY KEY
      - name : TEXT
      - remote_id : INTEGER
      - to_add : INTEGER
      - ...



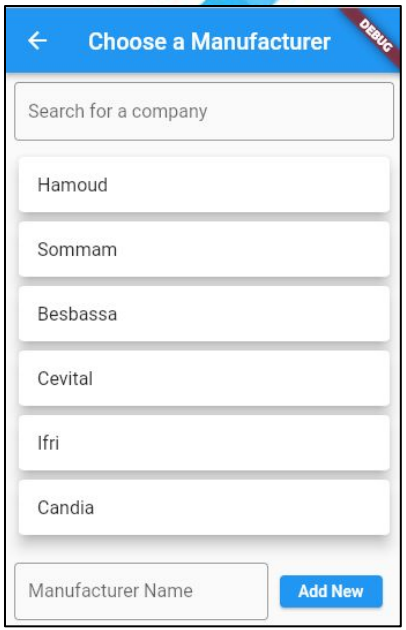
From MVP to Beta Version Case Study

- **Manufacturer Screen : Synching Data**
  - Remote API Endpoints (Let's do simple PHP..)
    - **companies.get**
    - **companies.add**



From MVP to Beta Version Case Study

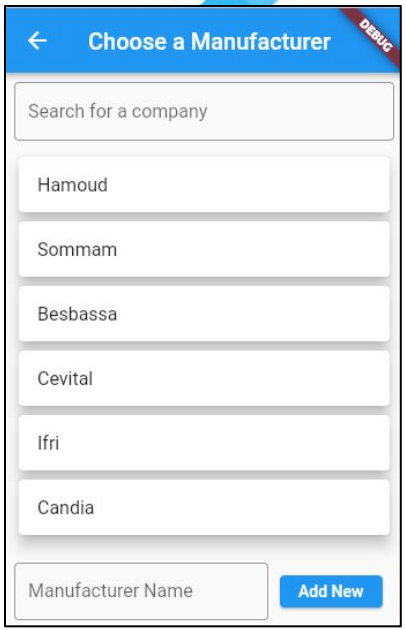
- **Manufacturer Screen : Synching Data**
  - Getting and Synching List of Companies
    - For local Companies flagged : To_Add
      - Send them to the server to add
      - Update Remote ID
    - Continue Syncing the same way as Categories



From MVP to Beta Version Case Study

- **Manufacturer Screen : Synching Data**

```
Future<bool> service_sync_companies() async {
 List<Map<String, dynamic>> data_upload =
     await DBCompany.getCompaniesToUpload();

 ??????????????????? What next ?

 print("Running Cron Service to get Companies");
 List? remote_data = await endpoint_api_get_companies();

 if (remote_data != null) {
   await DBCompany.syncCompanies(remote_data as List<Map<String, dynamic>>);
   return true;
 }
 return false;
}
```



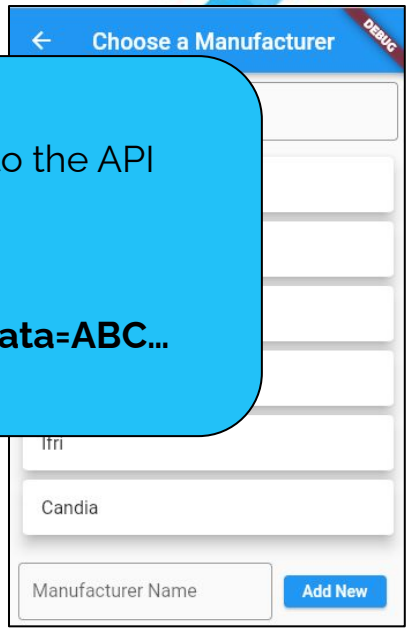
From MVP to Beta Version Case Study

- **Manufacturer Screen : Synching Data**

```
Future<bool> service_sync_companies() asyn
 List<Map<String, dynamic>> data_upload =
     await DBCompany.getCompaniesToUpload(

 ??????????????????? What next ?

 print("Running Cron Service to get Compa
 List? remote_data = await endpoint_api_ge

 if (remote_data != null) {
   await DBCompany.syncCompanies(remote_data as List<Map<String, dynamic>>);
   return true;
 }
 return false;
}
```

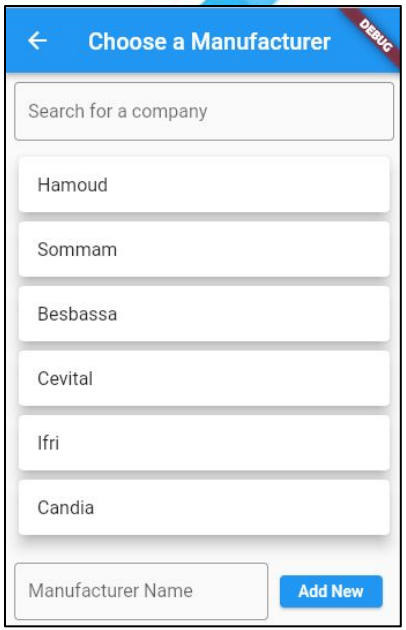
Need to send Data (NOT GET) to the API EndPoint

Shall we use :
url/?action=companies.add&data=ABC...

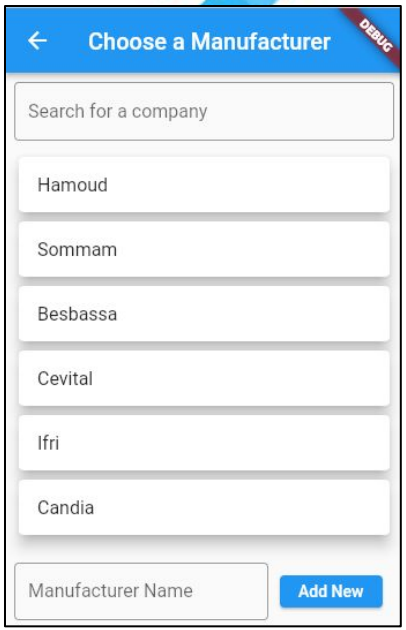


From MVP to Beta Version Case Study

- **Manufacturer Screen : Synching Data**
  - Install **Dio** Plugin
    - flutter pub add dio
  - Include the Package:
    - `import 'package:dio/dio.dart';`
  - Create an Instance of the Dio , inside the main.dart
    - `final dio = Dio();`
    - (Top Level variable or static ..)



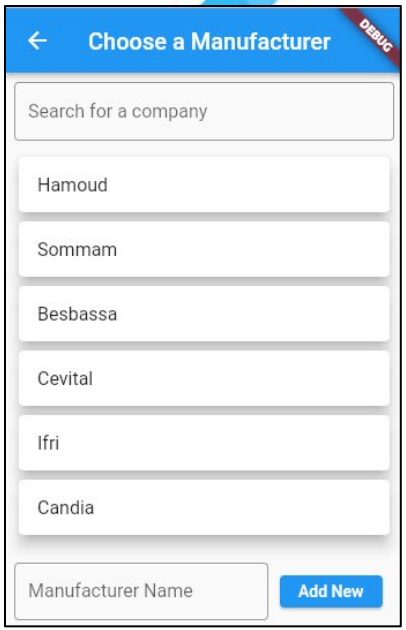
```
Future<bool> service_sync_companies() async {
 List<Map<String, dynamic>> data_upload =
     await DBCompany.getCompaniesToUpload();

 var response = await dio.post(
     'https://productinfoapp.startsoftware.dev/?action=companies.add',
     data: FormData.fromMap({'companies': jsonEncode(data_upload)}));


 if (response == null || response.data == null) return false;

 Map ret_data = jsonDecode(response.toString());
 if ((!ret_data.containsKey('status')) || ret_data['status'] != 'OK')
   return false;

 Map<String, dynamic> mapping = {};
 try {
   mapping = ret_data['mapping'];
 } catch (error) {}

 for (var local_id in mapping.keys) {
   await DBCompany.updateRecord(
       int.parse(local_id), {'remote_id': mapping[local_id], 'to_add': 0});
 }

 print("Running Cron Service to get Companies");
 List? remote_data = await endpoint_api_get_companies();

 if (remote_data != null) {
   await DBCompany.syncCompanies(remote_data as List<Map<String, dynamic>>);
   return true;
 }
 return false;
}
```



From MVP to Beta Version Case Study

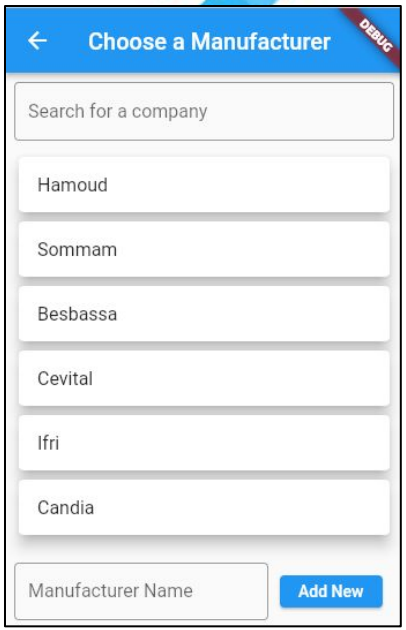
- **Manufacturer Screen : Synching Data**
  - Cron Code :
    - Same as Categories

```
cron.schedule(Schedule.parse('0 */2 * * *'), () async {
  service_sync_companies();
});
```



From MVP to Beta Version Case Study

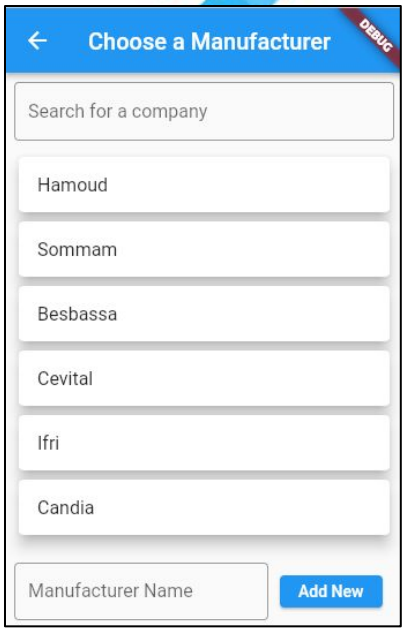
- **Manufacturer Screen : Search locally**
  - Simple Filter…
    - When typing few letters, it would show only relevant results
    - (Let's not bother about **pagination for now or loading more…)**



```
TextFormField(
  decoration: const InputDecoration(
    border: OutlineInputBorder(
        borderSide: BorderSide(color: Colors.teal)),
    labelText: 'Search for a company',
  ),
  keyboardType: TextInputType.text,
  onChanged: (newValue) {
    _tx_search_filter = newValue;
    setState(() {});
  },
),
```

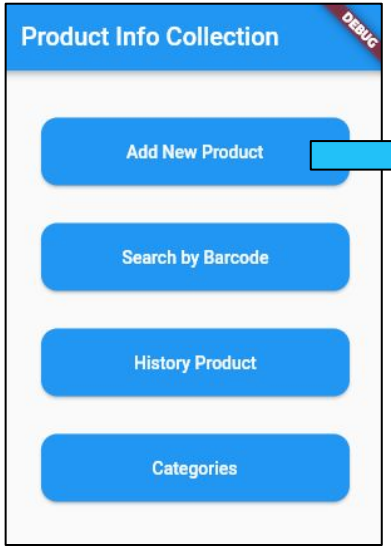
```
String _tx_search_filter = '';
Future<List<Map>> getListCompanies() async {
  return DBCompany.getAllCompaniesByKeyword(_tx_search_filter);
}
```

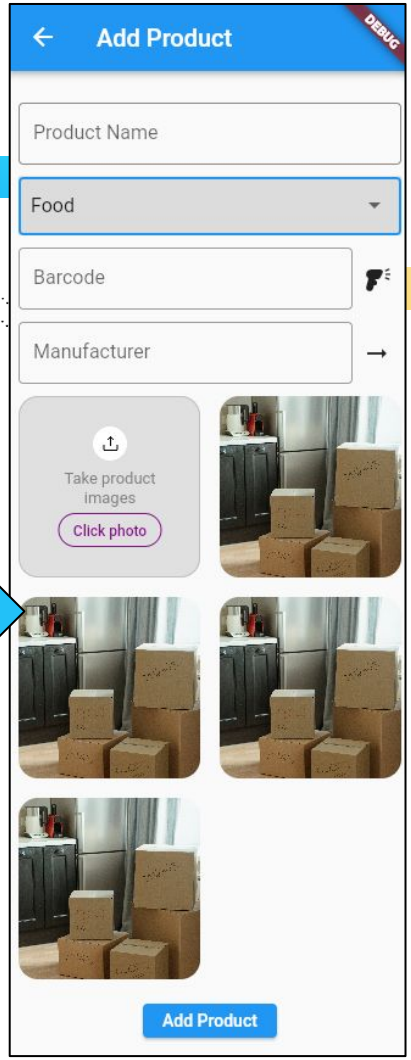
```
static Future<List<Map<String, dynamic>>> getAllCompaniesByKeyword(
    String keyword) async {
  if (keyword.isEmpty || keyword.trim() == '') return getAllCompanies();

  final database = await DBHelper.getDatabase();
  return database.rawQuery('''SELECT
          id ,
          name,
          remote id
        from ${tableName}
        Where LOWER(name) like '%${keyword.toLowerCase()}%'
        order by name ASC
        ''');
}
```



From MVP to Beta Version Case Study

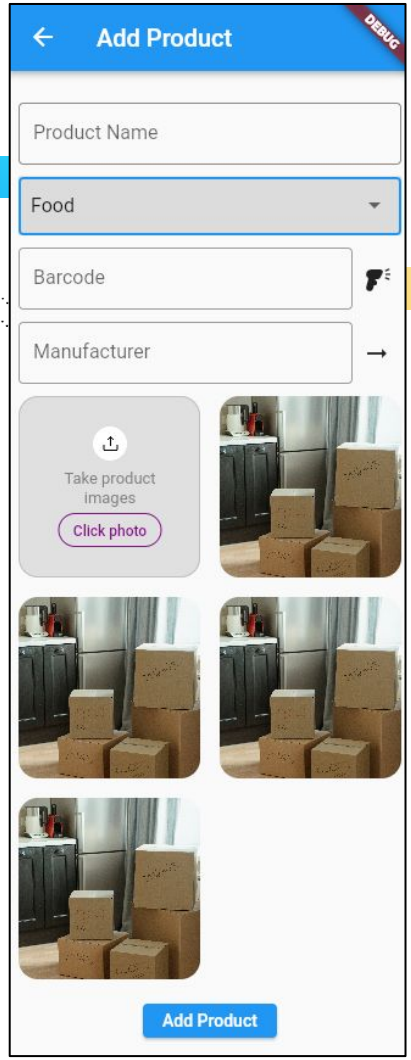
- **Add Product Screen**
  - Flutter Widget Tree : ?





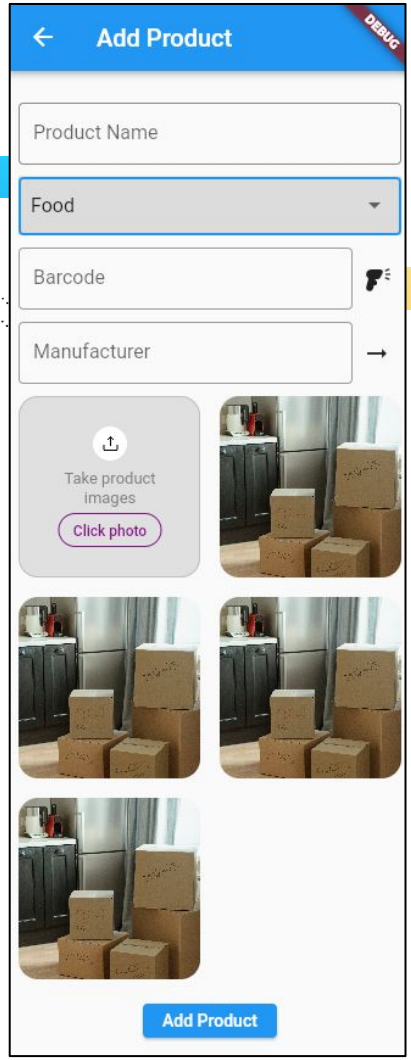
From MVP to Beta Version Case Study

- **Add Product Screen**
  - Flutter Widget Tree :
    - SingleChildScrollView
      - Column
        - TextFormField
        - DropdownButton
        - Row
        - Row
        - GridView
        - Button



From MVP to Beta Version Case Study

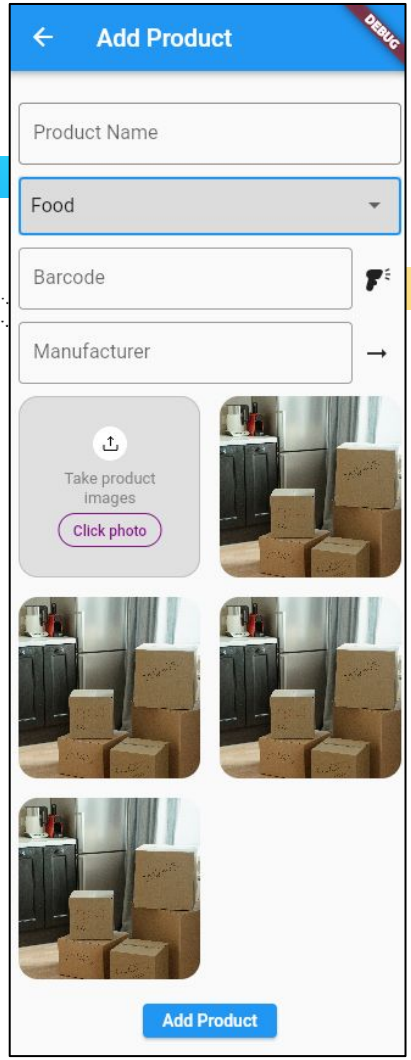
- **Add Product Screen : Type of Product**
  - Loading Data into the Dropdown Widget :
    - Using Future builder



From MVP to Beta Version Case Study

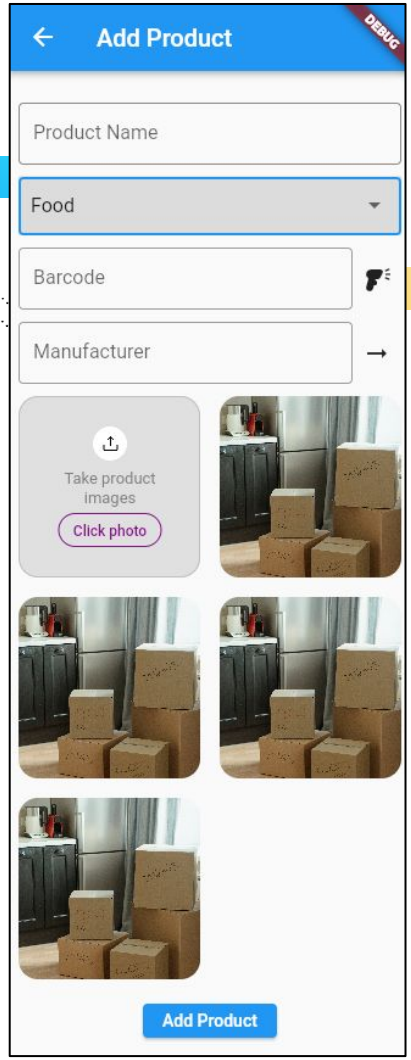
- **Add Product Screen : Barcode**
  - Scanning Barcode using Phone Camera
    - Install Plugin : `flutter_barcode_scanner`
      - flutter pub add `flutter_barcode_scanner`

```
onTap: () async {
   try {
      String barcodeScanRes =
               await FlutterBarcodeScanner.scanBarcode( '#ff6666',
'Cancel', true, ScanMode.BARCODE);
               _tx_barcode_controller.text = barcodeScanRes;
     } on PlatformException {}
```



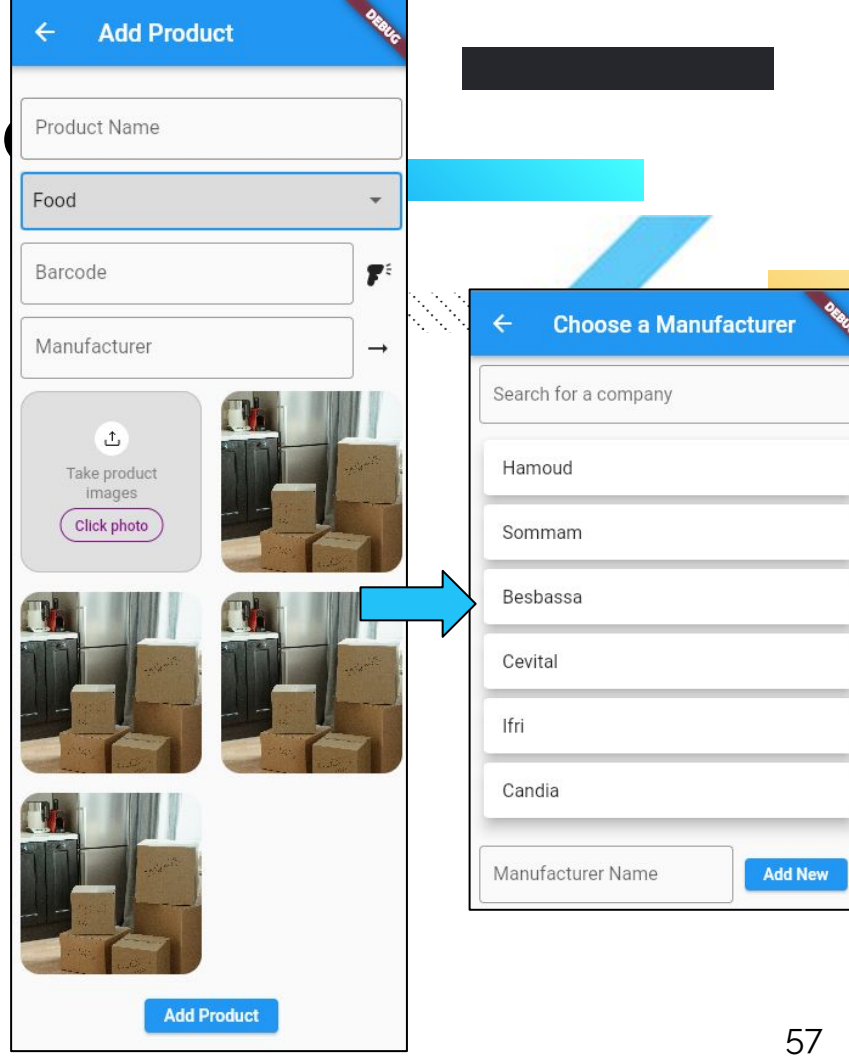
From MVP to Beta Version Case Study

- **Add Product Screen : Manufacturer**
  - Manufacturer :
    - Dropdown Box ?
    - Simple Text Form Input ?
    - ?
  - Same Questions as categories for storing ?
    - How big + How frequent to update ?



From MVP to Beta Versi
Case Study

- **Add Product Screen : Manufacturer**
  - Manufacturer :
    - Simple Text input as readonly
      - Takes to another widget

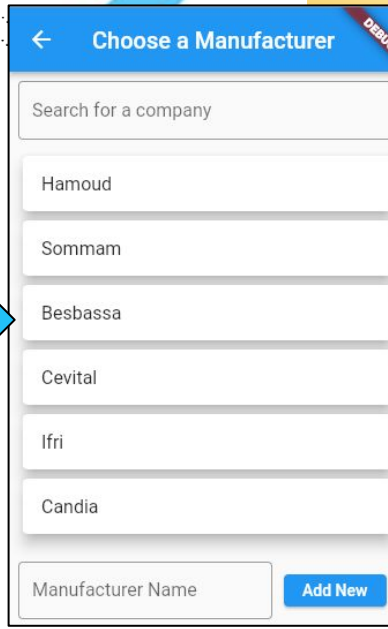


From MVP to Beta Versi
Case Study

- **Add Produ**
  - Manuf
    - Si
      -

How to receive data from the Child Screen to the Parent ?

Let's avoid using static..



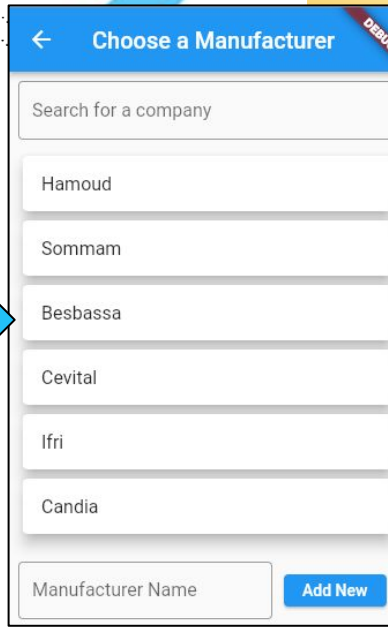
From MVP to Beta Versi
Case Study

- **Add Product Screen : Manufacturer**
  - Navigation ?

```
Navigator.of(context).pushNamed(Manufacturer.pageRoute);
```

  - **vs**

```
onTap: () async {
  Map manufactData = await Navigator.of(context).pushNamed(
      Manufacturer.pageRoute,
      arguments: {'from_product': 1}) as Map;
  _tx_company_controller.text = manufactData['name'];
  _tx_company_value = manufactData;
},
```



```
@override
Widget build(BuildContext context) {
  Future<List> companies = getListCompanies();

  Map args = {};
  var tmp = ModalRoute.of(context)!.settings.arguments;
  if (tmp != null) {
    args = tmp as Map;
  }

  if (args!['from_product'] == 1) {
    from_product = true;
  }
```

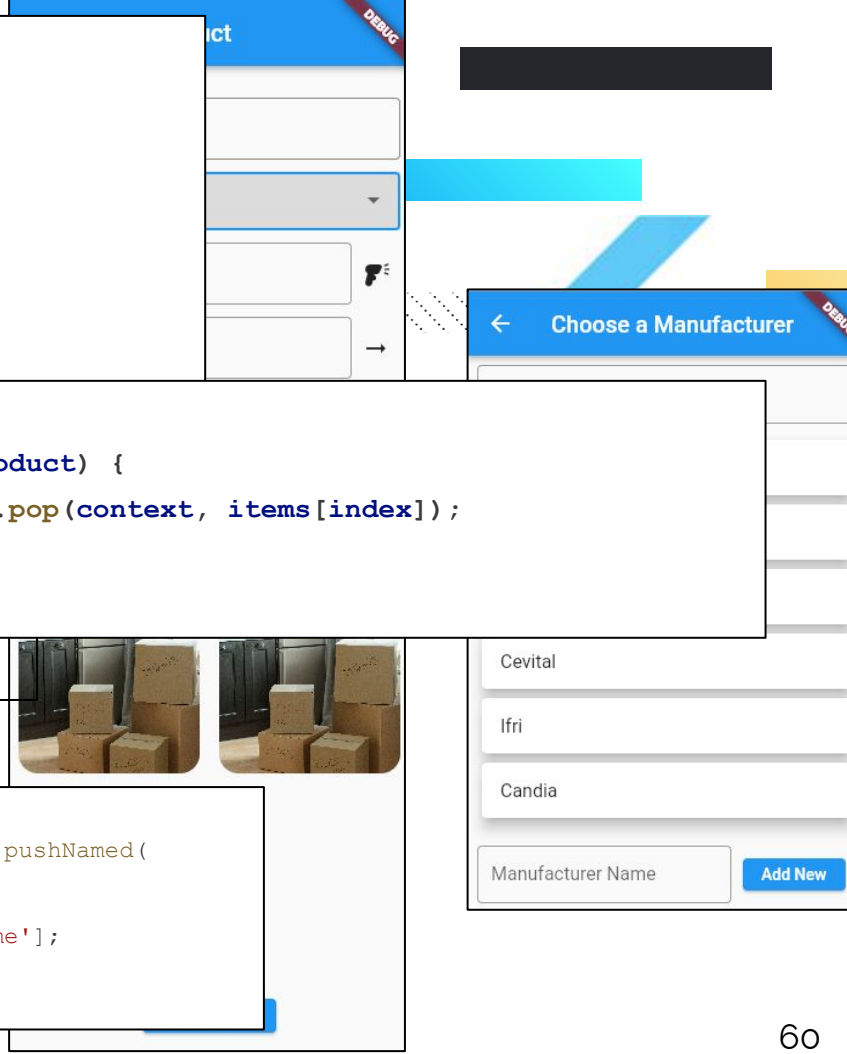
- Navigation ?

```
Navigator.of(context).pushNa
```

```
onTap: () {
  if (from_product) {
    Navigator.pop(context, items[index]);
  }
},
```

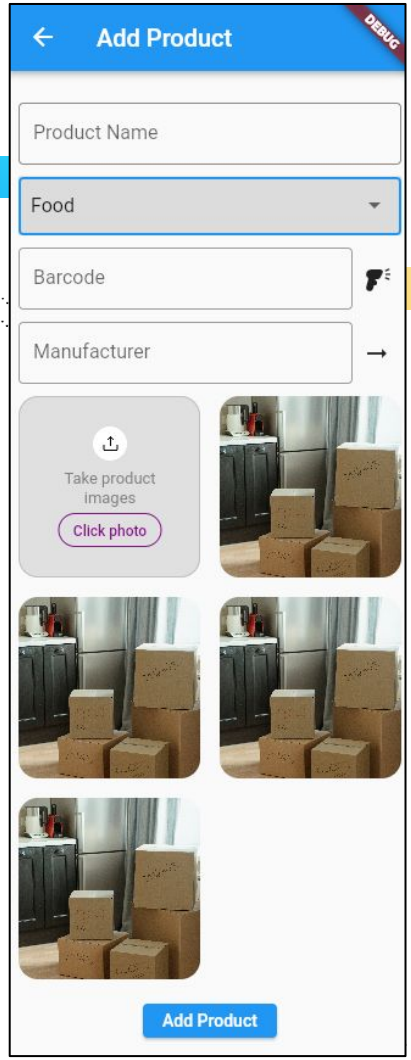
- **vs**

```
onTap: () async {
  Map manufactData = await Navigator.of(context).pushNamed(
      Manufacturer.pageRoute,
      arguments: {'from_product': 1}) as Map;
  _tx_company_controller.text = manufactData['name'];
  _tx_company_value = manufactData;
},
```



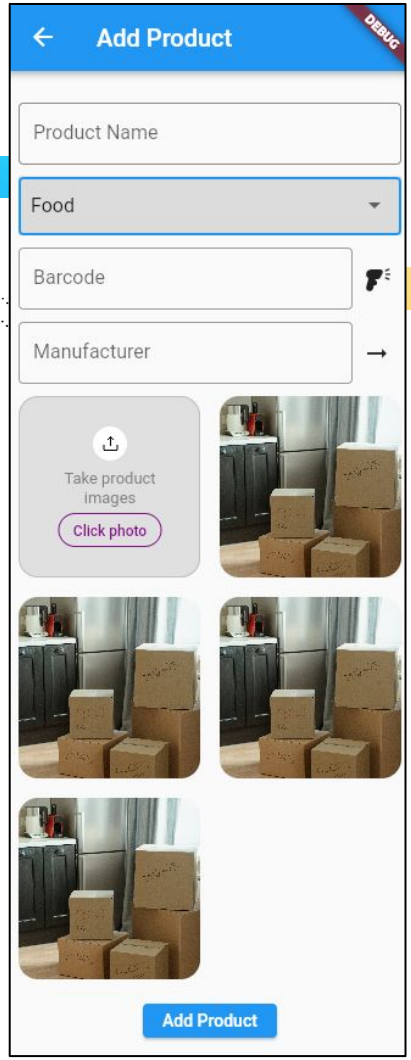
From MVP to Beta Version Case Study

- **Add Product Screen**
  - Images and Opening Camera :
    - Already included in the MVP, need to:
      - Before we click add Product :
        - For simplicity, we store them into a list data structure , not inside a Database Table .
        - Once they click add, we store in DB



From MVP to Beta Version Case Study

- **Add Product Screen : Taking Photos**
  - Images and Opening Camera :
    - Already included in the MVP, need to:
      - Before we click add Product : Shall we:
        - Store them into a list data structure ?
        - Or
        - Database Table



```dart
Future<String?> onImageButtonPressed(
   ImageSource source, BuildContext context) async {
 if (context.mounted) {
   try {
     final XFile? pickedFile = await _picker.pickImage(
       source: source,
       maxWidth: 1200,
       maxHeight: 1200,
       imageQuality: 70,
     );
     return pickedFile!.path;
   } catch (e) {}
   return null;
 }
}
```


```dart
Widget showImagesGrid(List imagesItems, bool readonly, widgetState) {
```

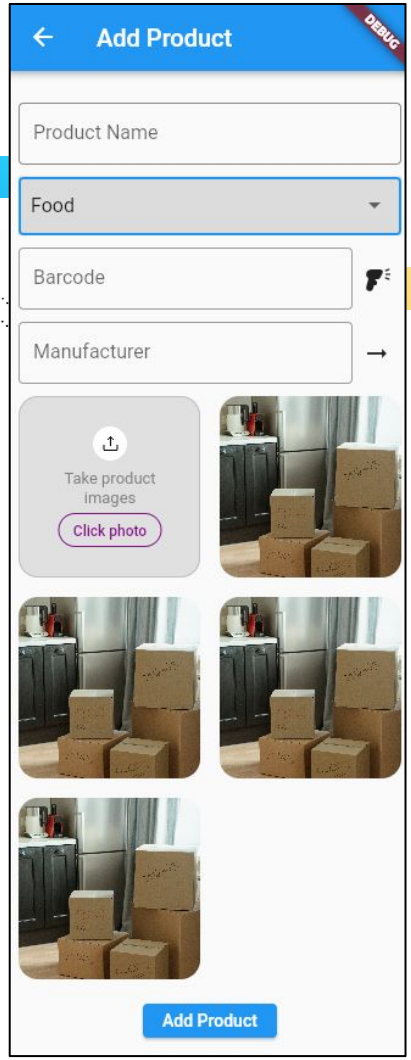
```dart
onPressed: () async {
  String? path = await onImageButtonPressed(
    ImageSource.camera,
    context,
  );
  if (path != null) {
    imagesItems.add({'type': 'file', 'imagePath': path});
    widgetState.setState(() {});
  }
},
```

- Already
  - Befo
    -
    -
    - Database Table

```dart
showImagesGrid(imageList, false, this),
```

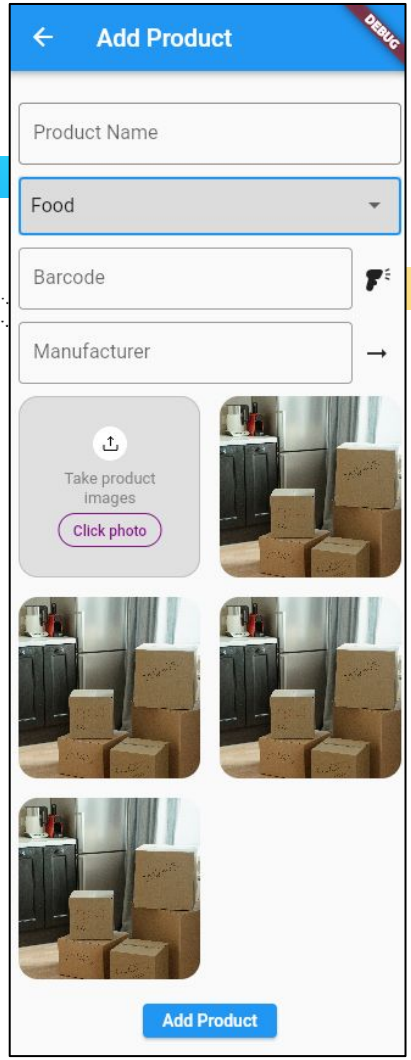
From MVP to Beta Version Case Study

- **Add Product Screen : Clicking the Add Button**
  - How to store into a local database ?
  - How to send product data to remote database ?
    (User can be Offline)
    - Send it with photos ?
    - Send Photos and later send information ?
    - Send Information and later send photos ?
    - What happens to the photo on my phone ?
    - How other people can view the photos on their phone ?



From MVP to Beta Version Case Study

- **Add Product Screen : Local Database Design**
  - Two database tables
    - **products**
      - Id
      - remote_id
      - name
      - barcode
      - category_id
      - company_id
      - to_add
    - **images**
      - id
      - product_id
      - remote_id
      - path
      - to_add



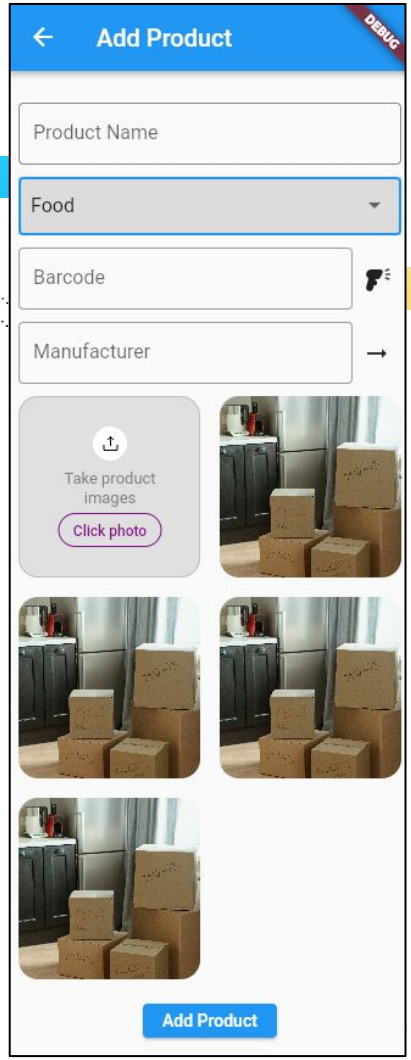
From MVP to Beta Version Case Study

- **Add Product Screen : Inserting into Local DB**

```
static Future<int> insertRecord(Map<String, dynamic> data) async {
  final database = await DBHelper.getDatabase();
  Map<String, dynamic> prodData = {
    'name': data['name'],
    'category_id': await DBCategory.getCategoryByName(data['type']),
    'barcode': data['barcode'],
    'company_id': data['company_data']['id'],
    'to_add': 1,
  };

  int id = await database.insert(tableName, prodData, conflictAlgorithm: ConflictAlgorithm.replace);

  for (Map<String, dynamic> img in data['images']) {
    img['product_id'] = id.toString();
    img['to_add'] = '1';
    await database.insert(tableName_images, img, conflictAlgorithm: ConflictAlgorithm.replace);
  }
  return id;
}
```

From MVP to Beta Version Case Study

- **Add Product Screen : Sending Data**
  - Technical Procedure : Cron Background Service
    - Send information without Images
      - get Remote ID
      - Mark as Synched or Uploaded.
    - Upload Image one by one
      - For each URL uploaded:
        - get ID + Remote URL
      - Delete Image FILE
      - Mark as Synched



From MVP to Beta Version

```
Future<bool> service_sync_products() async {
 List<Map<String, dynamic>> data_upload =
     await DBProduct.getProductsToUpload();

 var response = await dio.post(
     'https://productinfoapp.startsoftware.dev/?action=products.add',
     data: FormData.fromMap({'products': jsonEncode(data_upload)}));


 if (response == null || response.data == null) return false;

 Map ret_data = jsonDecode(response.toString());
 if ((!ret_data.containsKey('status')) || ret_data['status'] != 'OK')
   return false;

 Map<String, dynamic> mapping = {};
 try {
   mapping = ret_data['mapping'];
 } catch (error) {}
 for (var local_id in mapping.keys) {
   await DBProduct.updateRecord(
       int.parse(local_id), {'remote_id': mapping[local_id], 'to_add': 0});
 }

 return true;
}
```

From

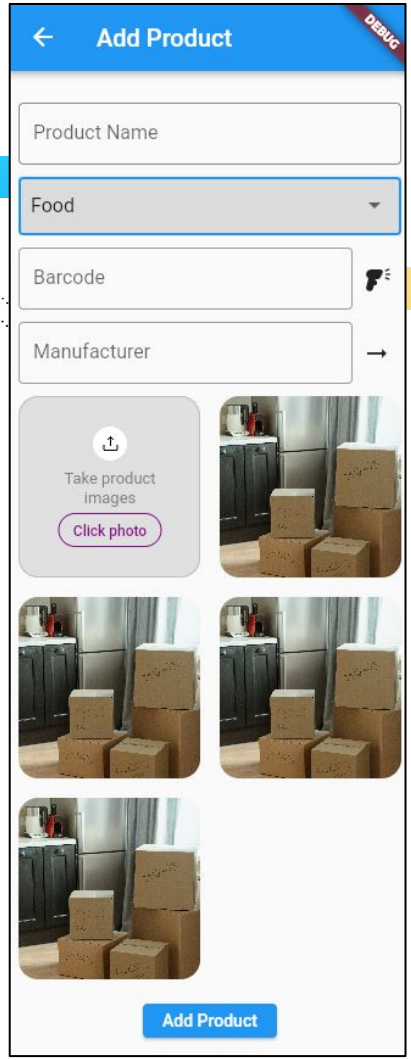
```
Future<bool>
 List<Map<St
     await D

 var respons
     'https:
     data: F


 if (respons

 Map ret_dat
 if ((!ret_d
   return fa

 Map<String,
 try {
   mapping =
 } catch (er
 for (var lo
   await DBP
       int.p
 }

 return true
}
```

```
static Future<List<Map<String, dynamic>>> getProductsToUpload() async {
  final database = await DBHelper.getDatabase();

  return database.rawQuery('''SELECT
          products.id as local_id,
          products.name,
          products.barcode,
          products.remote_id,
          products.to_add as sync,
          categories.remote_id as category_id,
          companies.remote_id as company_id
        from ${tableName}
            left join categories on category_id=categories.id
            left join companies on company_id=companies.id
        where
        products.to_add=1
        order by products.name ASC
        ''');
}
```

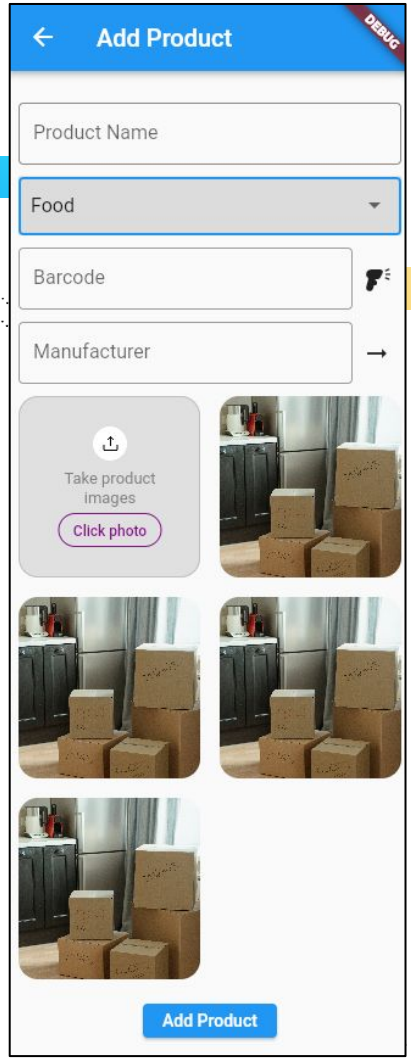
From MVP to Beta Version Case Study

- **Add Product Screen : Sending Data**
  - Technical Procedure : Cron Background Service
    - Send information without Images
      - get Remote ID
      - Mark as Synched or Uploaded.

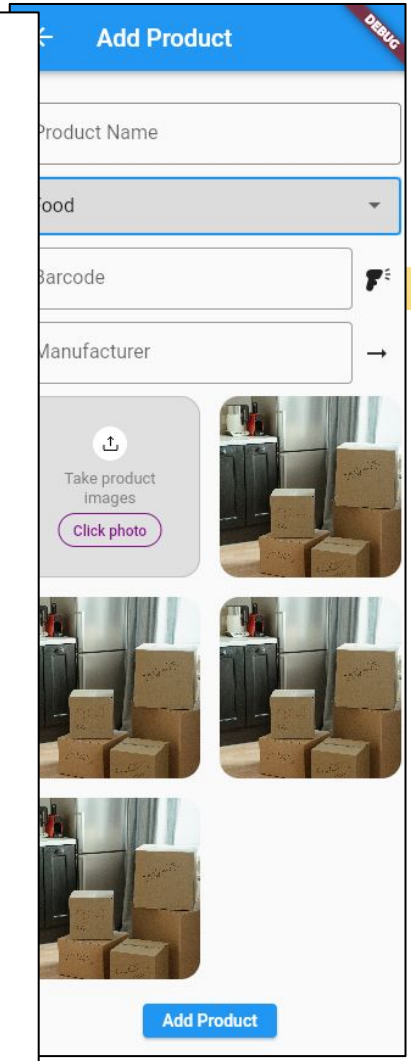
    - **Upload Image one by one**
      - **For each URL uploaded:**
        - **get ID + Remote URL**
      - **Delete Image FILE**
      - **Mark as Synched**



```dart
static Future<Map<String, dynamic>> getSingleImageToUpload() async {
   final database = await DBHelper.getDatabase();

   List<Map<String, dynamic>> res = await database.rawQuery('''SELECT
           images.id,
           images.type,
           images.imagePath,
           products.remote_id as product_id
         from ${tableName}
             left join products on product_id=products.id
         where
           images.to_add=1 and
           products.to_add=0
         order by images.id  ASC
         limit 1
         ''');
   return res[0];
 }
```



```
Future<bool> service_sync_images() async {
 Map<String, dynamic>? data_upload = await DBImage.getSingleImageToUpload();

 if (data_upload == null || data_upload.isEmpty) return true;
 var filename = data_upload['imagePath'].toString().split('/').last;
 late var image_multi_data;
 try {
   image_multi_data = await MultipartFile.fromFile(data_upload['imagePath'],
       filename: filename);
 } catch (error) {
   DBImage.deleteRecord(data_upload['id']);
   return false;
 }
 var response = await dio.post(
     'https://productinfoapp.startsoftware.dev/?action=images.add',
     data: FormData.fromMap({
       'image': jsonEncode(data_upload),
       'file': image_multi_data,
     }));
 if (response == null || response.data == null) return false;
 Map ret_data = jsonDecode(response.toString());
 if ((!ret_data.containsKey('status')) || ret_data['status'] != 'OK') return false;
 Map<String, dynamic> mapping = {};
 try {
   mapping = ret_data['mapping'];
 } catch (error) {}
 for (var local_id in mapping.keys) {
   await DBImage.updateRecord(int.parse(local_id), {
     'remote_id': mapping[local_id]['id'],
     'to_add': 0,
     'type': 'network',
     'imagePath': mapping[local_id]['link']
   });
   File(data_upload['imagePath']).delete();
 }
 return true;
```

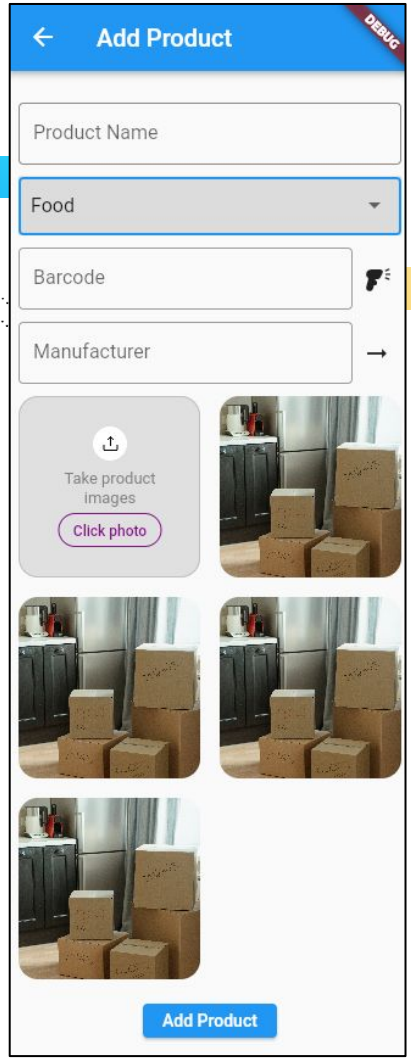


From MVP to Beta Version Case Study


```php
        case "images.add":{
            if (count($_FILES)==0)exit;
            $uploads_dir="img/";
            $url_link="https://productinfoapp.startsoftware.dev/img/";
            $filename=$_FILES["file"]['name'];

            if(!move_uploaded_file($_FILES["file"]["tmp_name"], "$uploads_dir/$filename"))exit;

            $line=json_decode($vars['image']);
            $mapping=[];
            if ($line){
                $db->query("INSERT INTO images (name,product_id) VALUES (?,?)",$filename,$line->product_id);
                $mapping[$line->id]=Array('id'=>$db->lastInsertID(),'link'=>$url_link."/".$filename);
                $ret['status']='OK';
                $ret['mapping']=$mapping;
                echo json_encode($ret);
            }
            exit;
        }break;
```

From MVP to Beta Version Case Study

- **Add Product Screen : Sending Data**
  - API EndPoint : Simple PHP
    - Send information and get ID
    - Upload Image one by one
      - For each URL uploaded , get ID
      - Link product ID to the ImageID

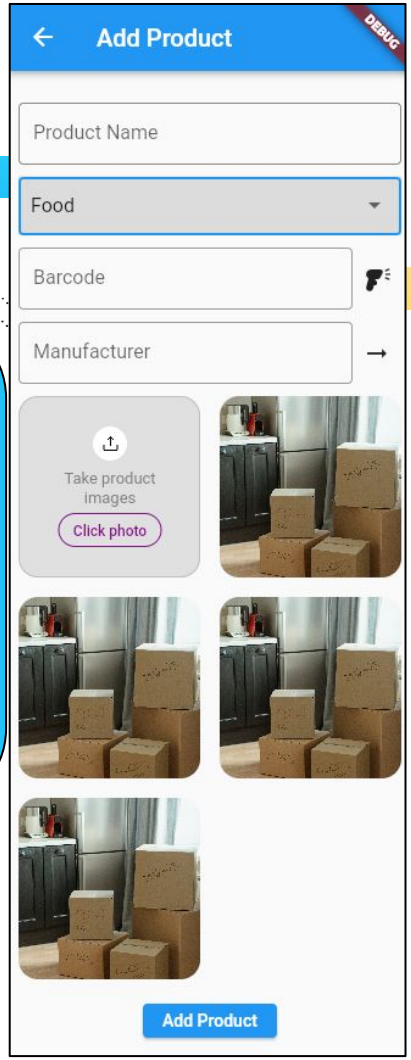


From MVP to Beta Version Case Study

- Ad
  -

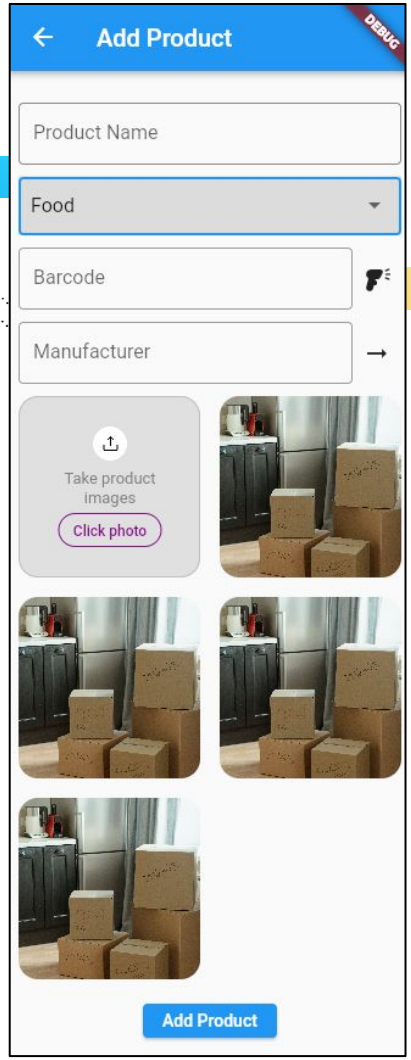
For simplicity, the same server is used for uploading Images and Text Data

In Practice, you create a storage EndPoint for handling binary data away from light or critical endPoints



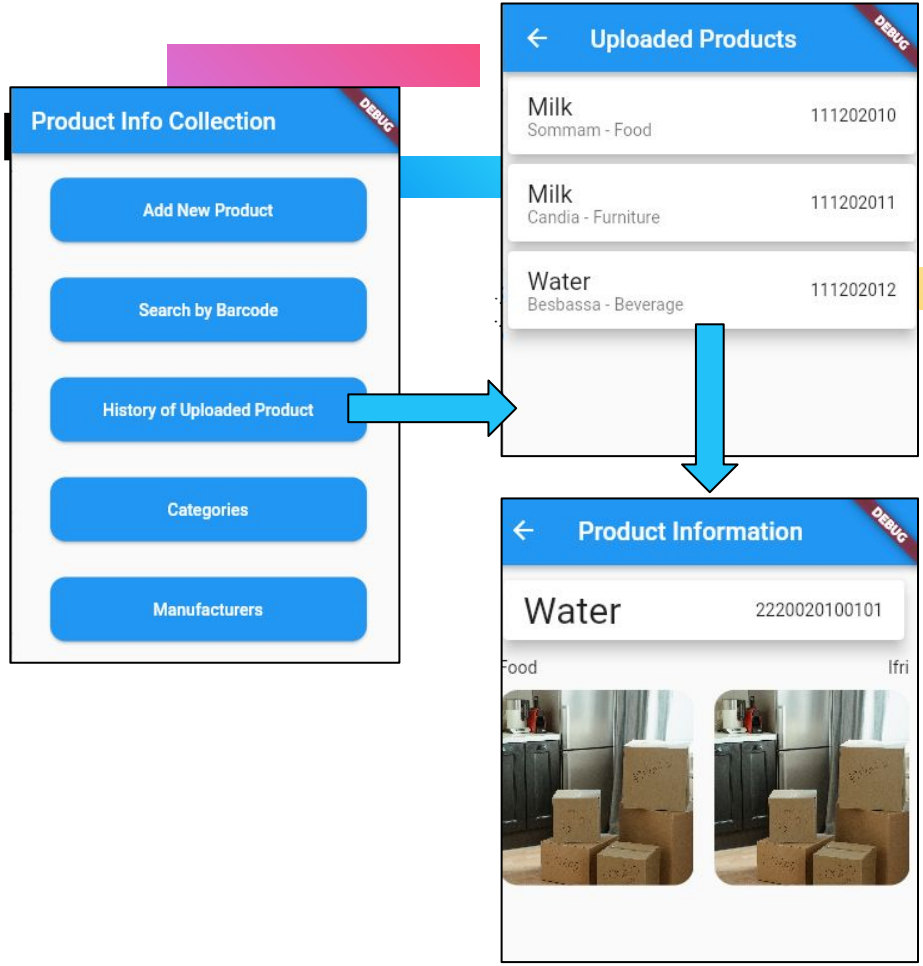
From MVP to Beta Version Case Study

- **Add Product Screen : Sending Data**
  - API EndPoint : Advanced to use S3 Storage
    - Send information and get ID
    - Upload Image one by one
      - For each URL uploaded , get ID
      - Link product ID to the ImageID



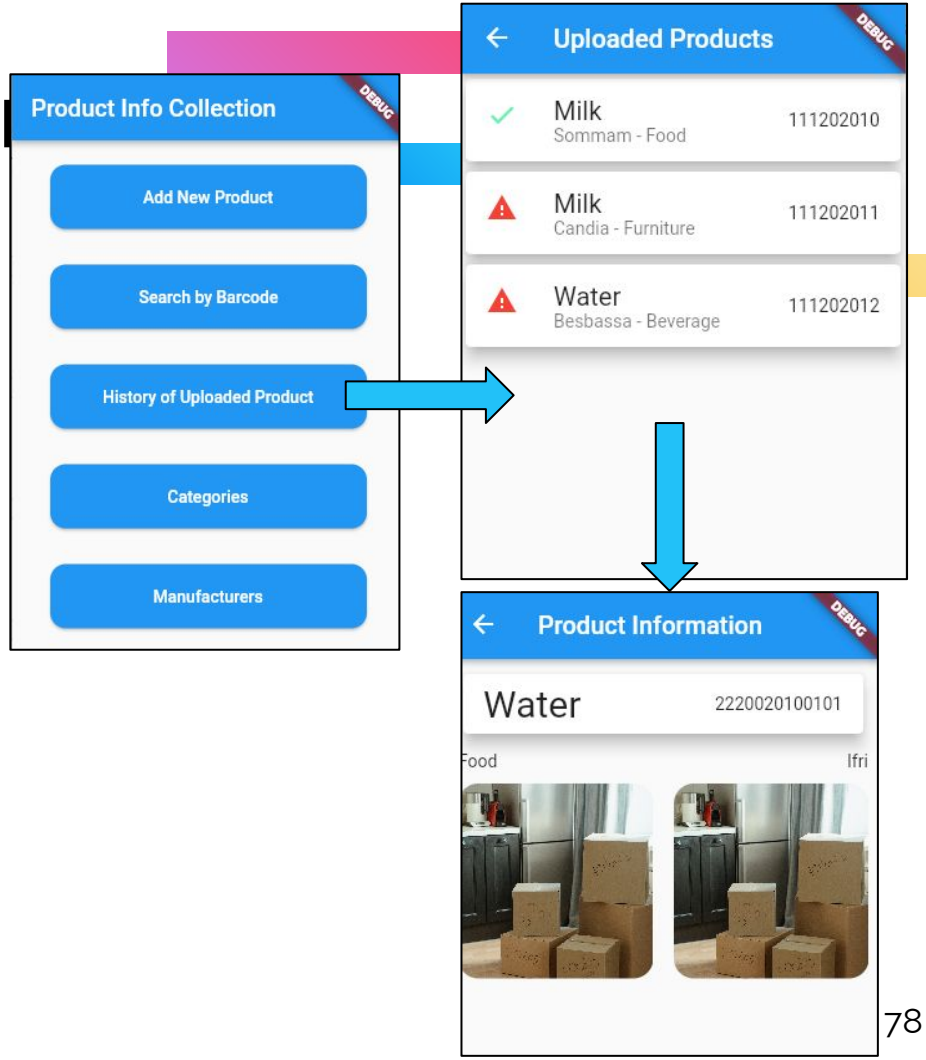
From MVP to Beta Ve
Case Study

- **History & Product Info Screens**
  - Navigations



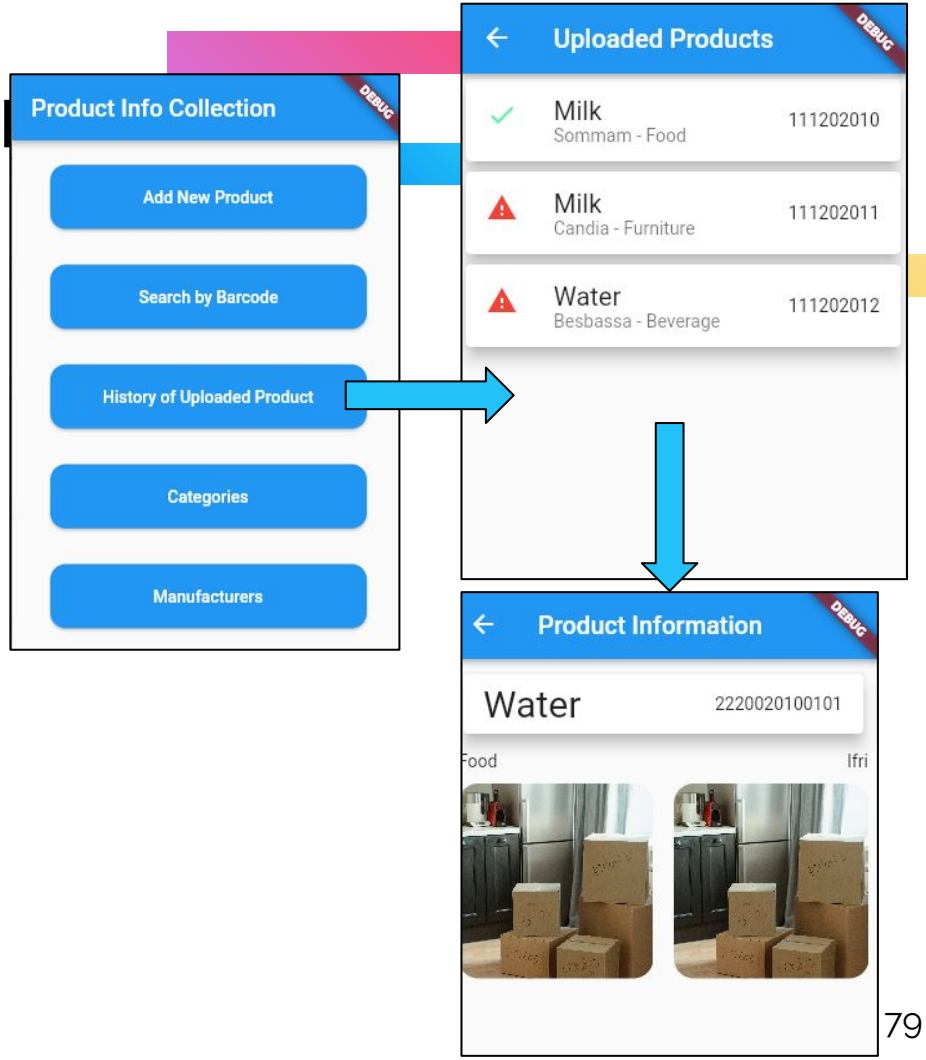
From MVP to Beta Ve
Case Study

- **History & Product Info Screens**
  - Navigations



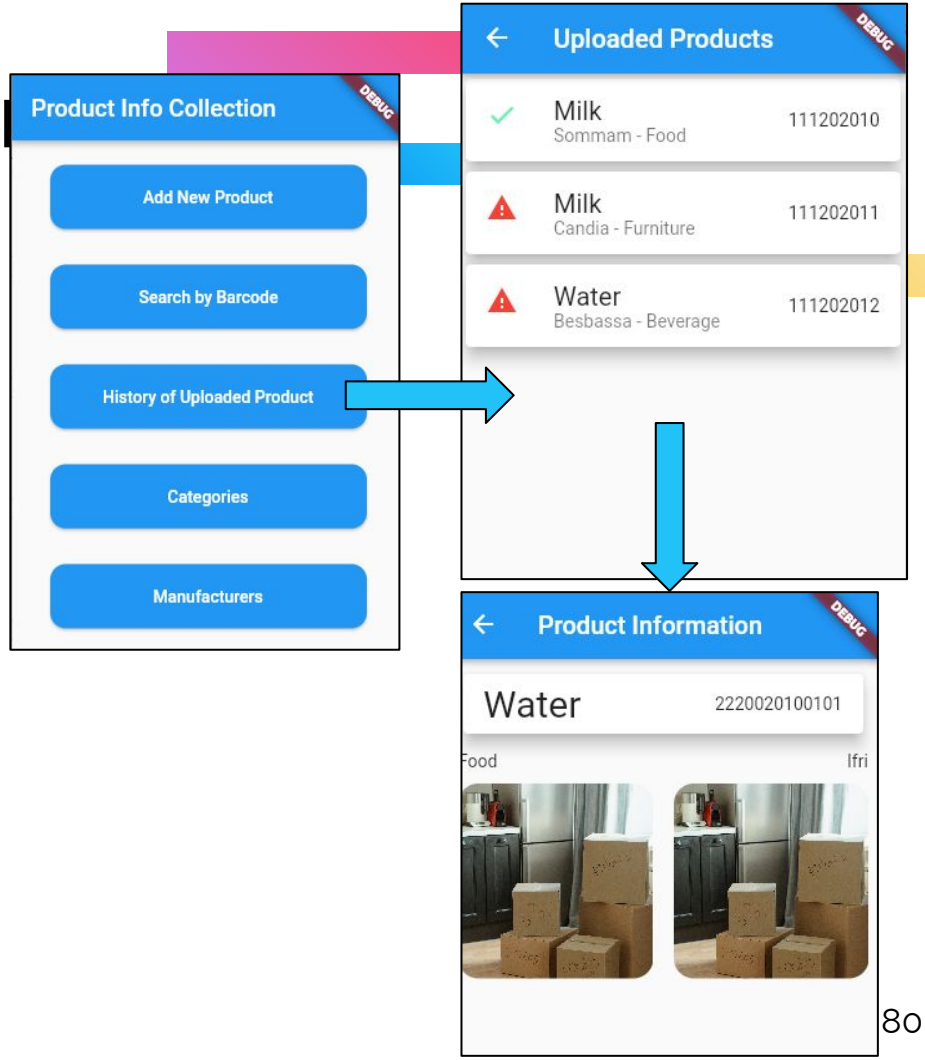
From MVP to Beta Ve
Case Study

- **History & Product Info Screens**
  - Passing data +
  - Use of Future Builder
  - Loading from Database



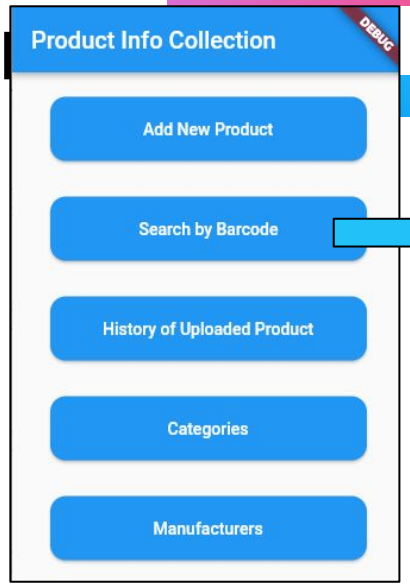
From MVP to Beta Ve
Case Study

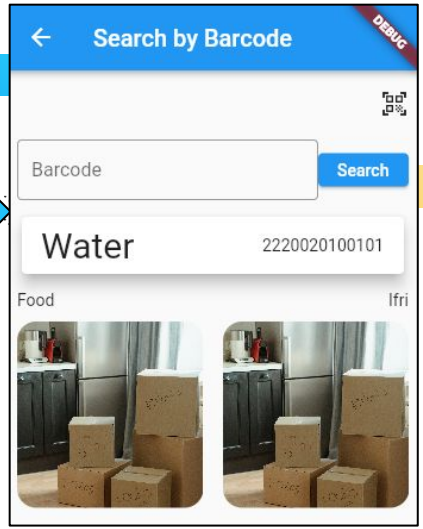
- **History & Product Info Screens**
  - Passing data +
  - Use of Future Builder
  - Loading from Database



From MVP to Beta Ve
Case Study

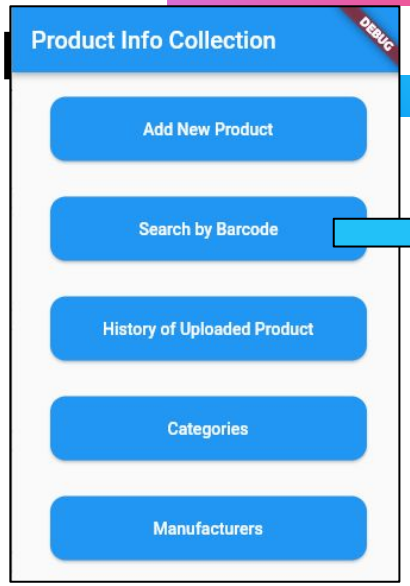
- **Search by Barcode Screen**
  - Tree of Widgets
  -

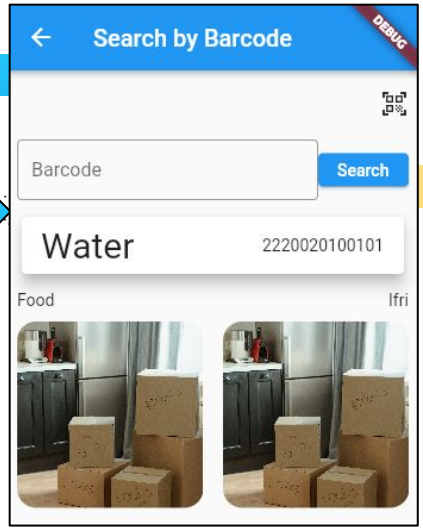




From MVP to Beta Ve
Case Study

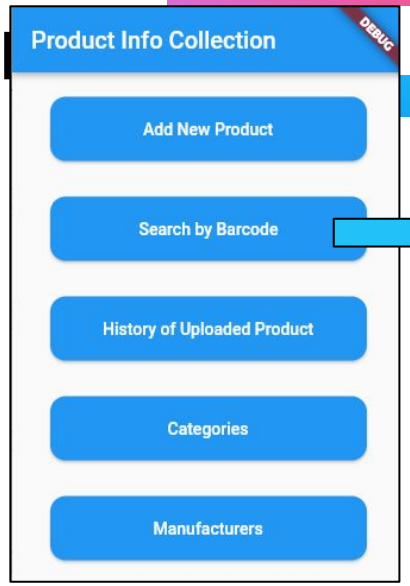
- **Search by Barcode Screen**
  - Searching..
  -

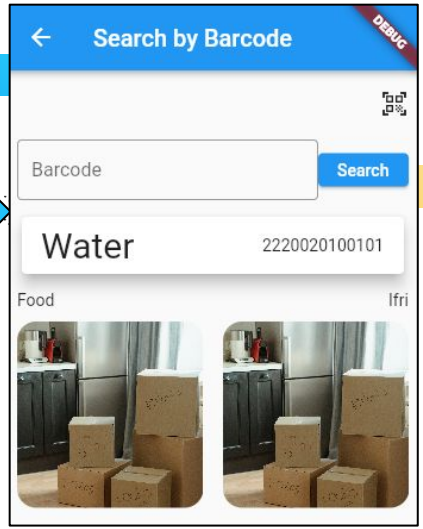




From MVP to Beta Ve
Case Study

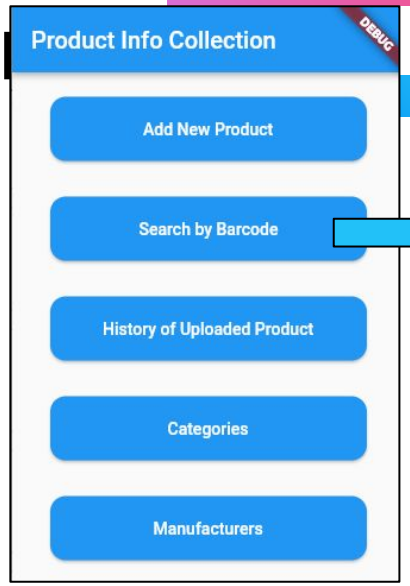
- **Search by Barcode Screen**
  - Use of Future Builder
  -

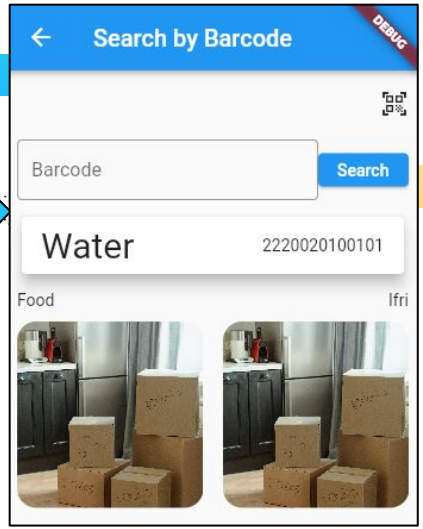




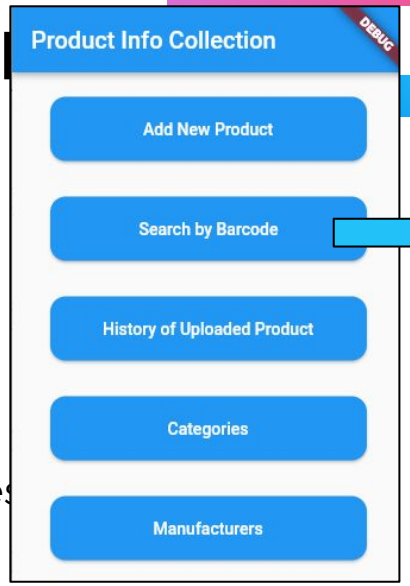
From MVP to Beta Ve[redacted] Case Study

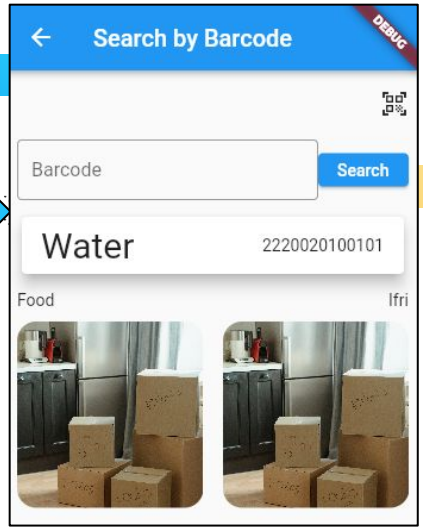
- **Search by Barcode Screen**
  - Use of Future Builder
    - Connect to API…





From MVP to Beta Ve
Case Study





- **Any bugs ? issues ?**
  - Adding product, taking photos and suddenly you hit back button: Images will be lost or ?
  - Pagination of larger lists,
  - You suddenly change the API Endpoint to a different server ? Recompile and Redestribute the app again ?

Lecture Demo Apps

- MVP Only
  - https://www.dropbox.com/scl/fo/dti2n1ncf1cxqc8ydf9qq/h?rlkey=tkjg3porqvpr2ot2lkhb26mbi&dl=0&authuser=0
- App with the functionality of Refreshing Categories Data
  - https://www.dropbox.com/scl/fo/att3n1j53om4drcb7szjm/h?rlkey=221cfa5pt1e6ck52kik2116ce&dl=0
- App with all functionalities (With more bugs)
  - https://www.dropbox.com/scl/fo/r7kl32el9kshmqrh6vvha/h?rlkey=6r5ar2ig25y5iyzgjclp8du3q&dl=0

Resources

- https://pub.dev/packages/dio
- https://pub.dev/packages/flutter_barcode_scanner
- https://www.youtube.com/watch?v=u52TWx410U4&ab_channel=Droidmonk