

Practica 3.1 - Aplicación ToDo

Sesión 2

Mtro. Anastacio Rodríguez García

UNIVA Campus León

20 de Marzo de 2021

Creamos un proyecto en Android Studio: **ToDo**

Abrimos Android Studio y creamos el proyecto ToDo

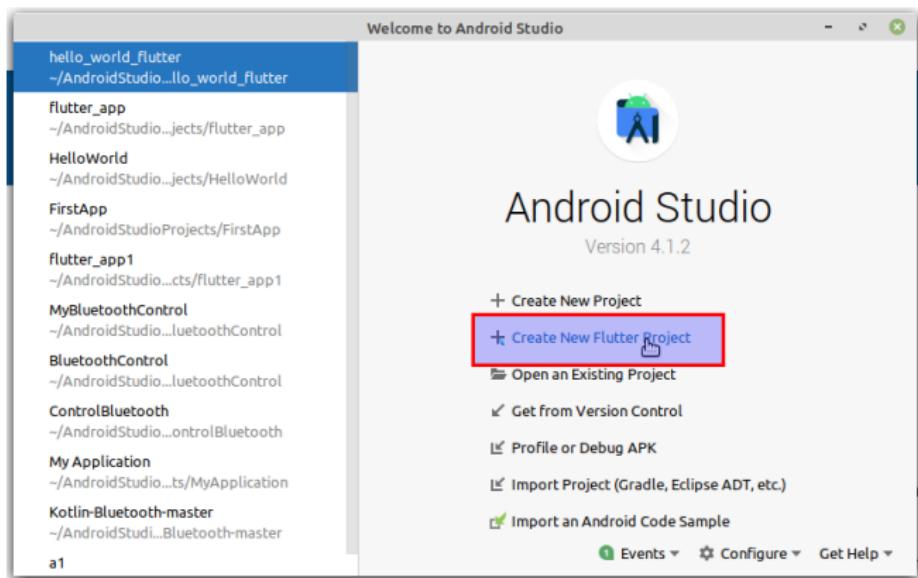


Figura: Creación del proyecto ToDo

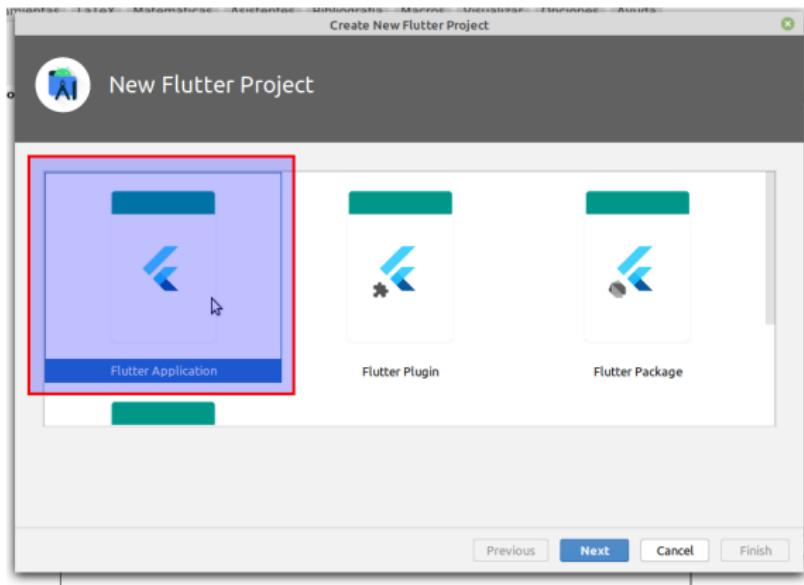


Figura: Elegimos una Flutter Application

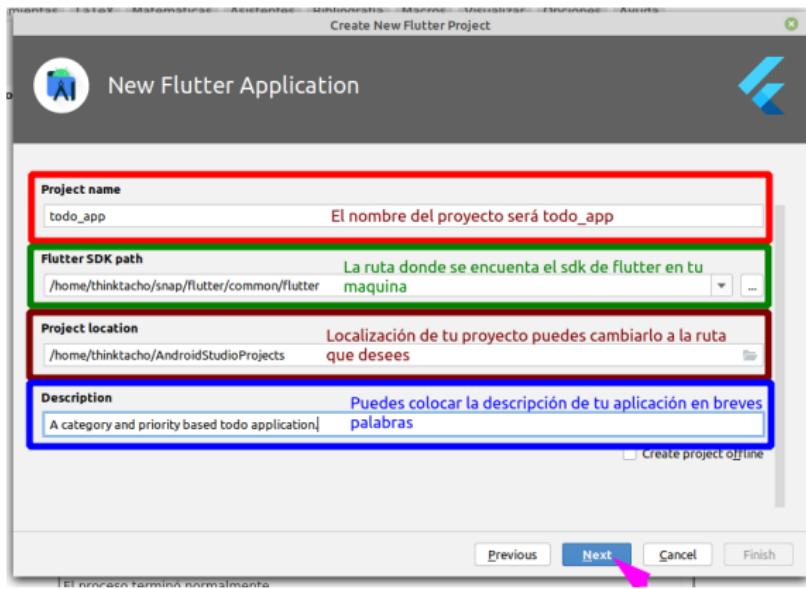


Figura: Coloca el nombre y la descripción del proyecto.

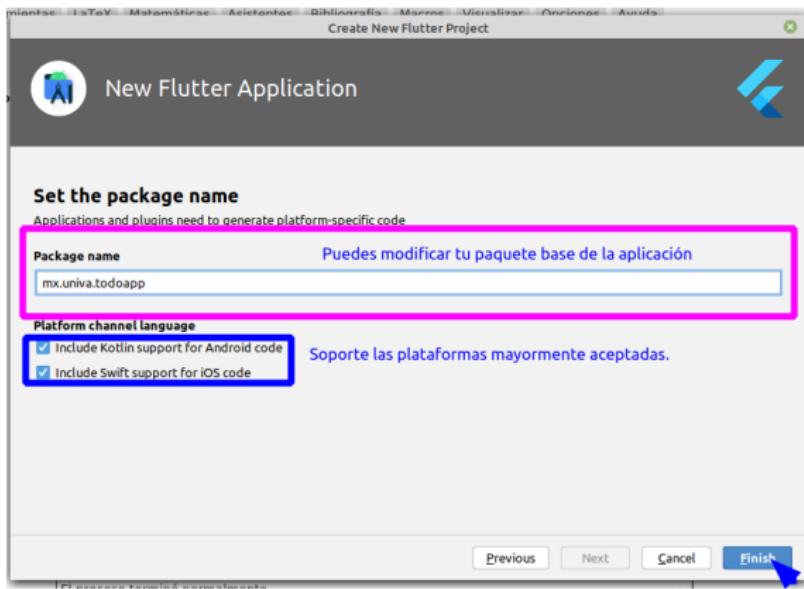


Figura: Finalizamos la creación del proyecto.

En el paquete **lib** crea un nuevo paquete **src**

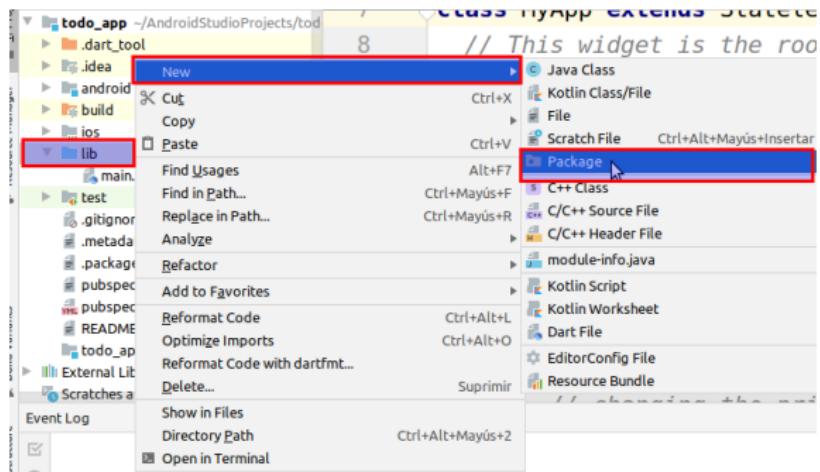


Figura: Crearemos un paquete llamado **src**

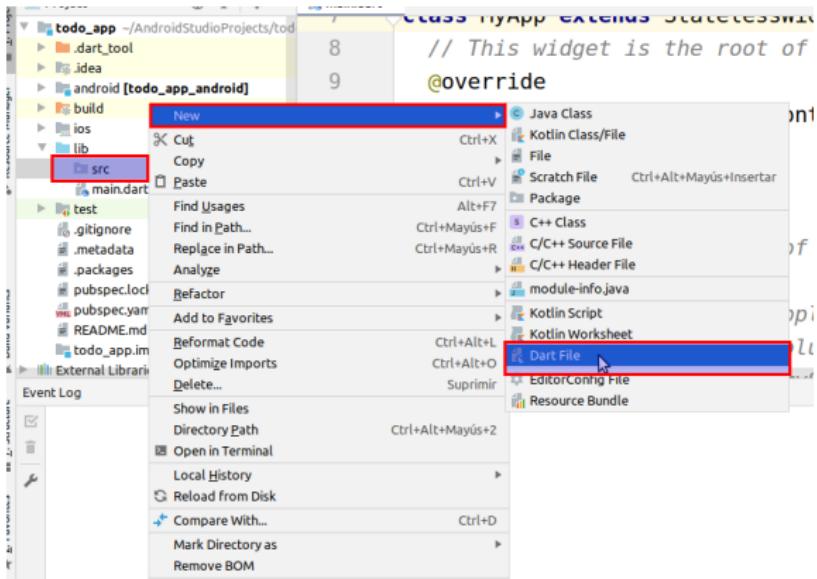


Figura: En el paquete `src` crea un nuevo archivo dart llamado `app.dart`

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the following Dart code:

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

class App extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp();
    }
}
```

The code is highlighted with syntax coloring. The first line, 'import', and the class definition 'class App' are highlighted in green. The word 'MaterialApp' in the return statement is also highlighted in green. A red circular icon with a white arrow is positioned next to the opening brace of the build method.

Figura: Modifica Colocando el nombre del Stateless Widget y y
MaterialApp usando Control Espacio para que haga el import
correspondiente

```
main.dart x app.dart x
1 import 'package:flutter/material.dart';
2 import 'package:todo_app/src/app/app.dart';
3
4 >> void main(){
5     runApp(App());
6 }
```

Usa CONTROL + ESPACIO
para hacer los import automá-
ticamente.

Figura: Borra todo el contenido de **main.dart** y agrega el método principal invocando al método **runApp()** que llame a una instancia de **App()**

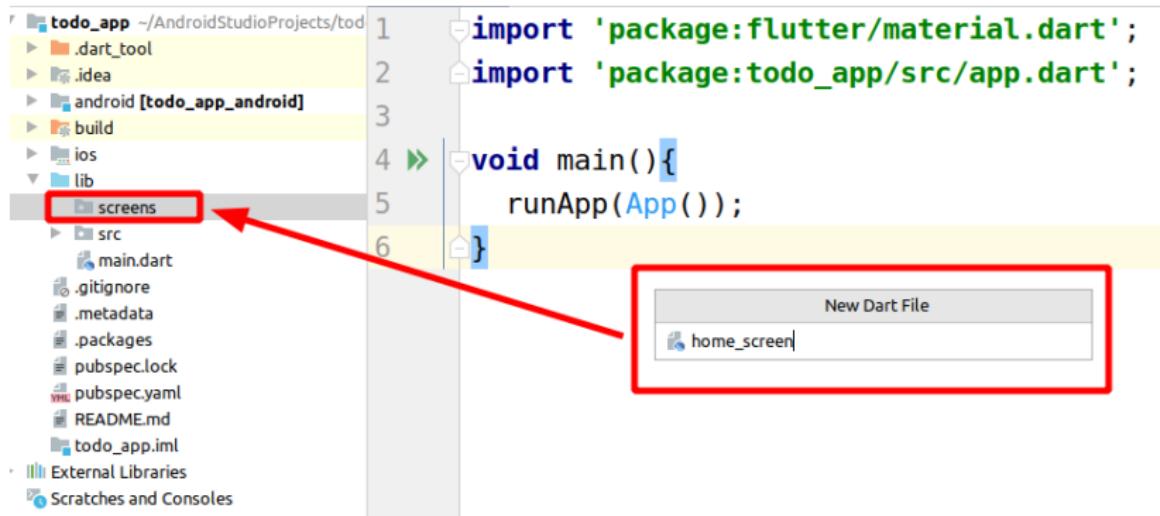


Figura: Crea un paquete llamado **screens** dentro de **lib** y en el agrega un archivo dart llamado **home_screen.dart**

The screenshot shows a code editor interface with three tabs at the top: 'main.dart', 'home_screen.dart' (which is currently selected), and 'app.dart'. In the main editor area, the letter 's' is typed into a text input field. A dropdown menu appears, listing three suggestions: 'stful' (New StatefulWidget), 'stless' (New StatelessWidget), and 'stanim' (New StatefulWidget with Animation). The first suggestion, 'stful', is highlighted with a red border.

```
main.dart × home_screen.dart × app.dart ×  
s|  
stful New StatefulWidget  
stless New StatelessWidget  
stanim New StatefulWidget with Animation
```

Figura: Crea un StatefulWidget utilizando el método abreviado inicia con una S y selecciona

The screenshot shows a code editor with three tabs at the top: main.dart, home_screen.dart, and app.dart. The home_screen.dart tab is active, displaying the following Dart code:

```
1 import 'package:flutter/cupertino.dart';
2 import 'package:flutter/material.dart';
3 class HomeScreen extends StatefulWidget {
4   @override
5   _HomeScreenState createState() => _HomeScreenState();
6 }
7
8 class _HomeScreenState extends State<HomeScreen> {
9   @override
10  Widget build(BuildContext context) {
11    return Scaffold(
12      appBar: AppBar(
13        title: Text("ToDo App"),
14      ), // AppBar
15    ); // Scaffold
16  }
17}
```

Figura: Modifica colocando el nombre **HomeScreen** a la clase, en automático se modificarán los elementos internos del Widget con estado también cambia el container por Scaffold y agrega una barra de app.

The screenshot shows the Android Studio interface with the project 'todo_app' open. The left sidebar displays the project structure, including 'lib', 'src', and various files like 'main.dart', 'home_screen.dart', and 'app.dart'. The main editor window shows the code for 'app.dart':

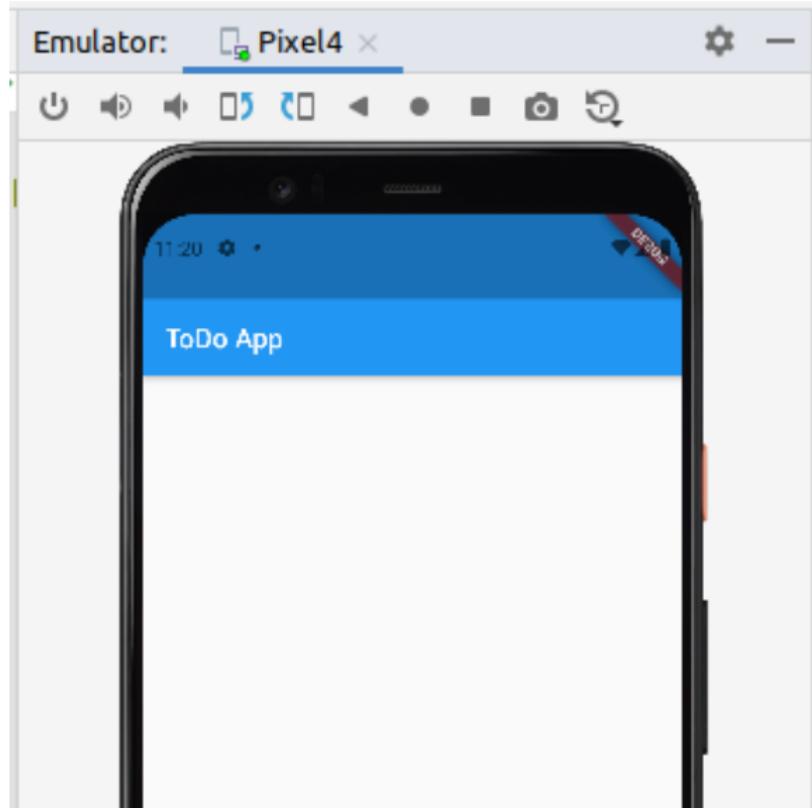
```
import 'package:flutter/material.dart';
import 'package:todo_app/screens/home_screen.dart';

class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomeScreen(),
    ); // MaterialApp
  }
}
```

A red annotation highlights the line 'home: HomeScreen()', with the text 'Coloca como home la pantalla de inicio creada en el Widget con Estado' overlaid in red.

Figura: Agrega al Widget App en el método MaterialApp nuestro nuevo Widget HomeScreen.

Ejecuta tu aplicación en un emulador que hayas creado con anterioridad o en tu propio dispositivo



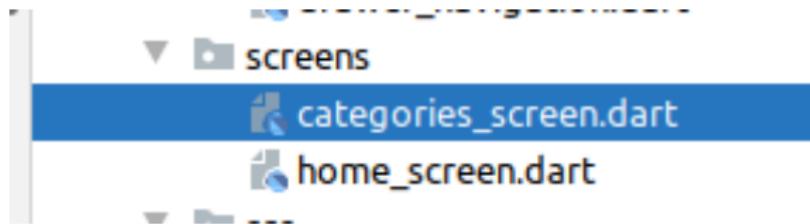


Figura: Creamos un archivo dart llamado **categories_screen.dart** dentro del paquete **screens**

```
1 import 'package:flutter/material.dart';
2
3 class CategoriesScreen extends StatefulWidget {
4   @override
5   _CategoriesScreenState createState() => _CategoriesScreenState();
6 }
7
8 class _CategoriesScreenState extends State<CategoriesScreen> {
9   @override
10  Widget build(BuildContext context) {
11    return Container(
12      child: Center(
13        child: Text("Welcome to Categories screen!"),
14      ), // Center
15    ); // Container
16  }
17 }
```

Figura: Codificamos con el método abreviado **stfui** y colocando el nombre y un texto centrado en el contenido

The screenshot shows the Android Studio interface. On the left, the Project Structure sidebar displays the project tree for 'todo_app'. A red box highlights the 'lib/helpers' folder. A red arrow points from this box to the code editor on the right. The code editor shows the 'lib/main.dart' file with the following code:

```
import 'package:flutter/material.dart'
import 'package:todo_app/screens/home.dart'

class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: drawer_navigation // MaterialApp
    );
  }
}
```

A red box highlights the 'drawer_navigation' part of the 'MaterialApp' line. A context menu is open at this location, showing options like 'New Dart File'. The 'MaterialApp' line is also highlighted with a red box.

Figura: Crea un paquete dentro de **lib** llamado **helpers** y dentro de él el archivo dart **drawer_navigation.dart**

```
main.dart × home_screen.dart × app.dart × drawer_navigation.dart × categories_screen.dart
1 import 'package:flutter/material.dart';
2 import 'package:todo_app/screens/categories_screen.dart';
3 import 'package:todo_app/screens/home_screen.dart';
4 class DrawerNavigation extends StatefulWidget {
5     @override
6     _DrawerNavigationState createState() => _DrawerNavigationState();
7 }
8
9 class _DrawerNavigationState extends State<DrawerNavigation> {
10    @override
11    Widget build(BuildContext context) {
12        return Container(
13            child: Drawer(
14                child: ListView(
15                    children: [
16                        UserAccountsDrawerHeader(
17                            accountName: Text("ToDo App"),
18                            accountEmail: Text("Category & Priority based ToDo App"),
19                            currentAccountPicture: GestureDetector(
20                                child: CircleAvatar(
21                                    backgroundColor: Colors.black54,
22                                    child: Icon(Icons.filter_list,
23                                        color: Colors.white), // Icon
24                                ), // CircleAvatar
25                                ), // GestureDetector
26                                decoration: BoxDecoration(
27                                    color: Colors.red
28                                )
29                ]
30            )
31        );
32    }
33}
```

```
28     ),
29     ),
30     ListTile(
31       title: Text("Home"),
32       leading: Icon(Icons.home),
33       onTap: () {
34         Navigator.of(
35           context).push(new MaterialPageRoute(
36             builder: (context) => new HomeScreen())); // MaterialPageRoute
37       },
38     ), // ListTile
39     ListTile(
40       title: Text("Categories"),
41       leading: Icon(Icons.view_list),
42       onTap: () {
43         Navigator.of(
44           context).push(new MaterialPageRoute(
45             builder: (context) => new CategoriesScreen())); // MaterialPageRoute
46       },
47     ), // ListTile
48   ],
49   ), // ListView
50   ), // Drawer
51 ); // Container
52 }
53 }
```

Ejecutamos y abrimos el drawer

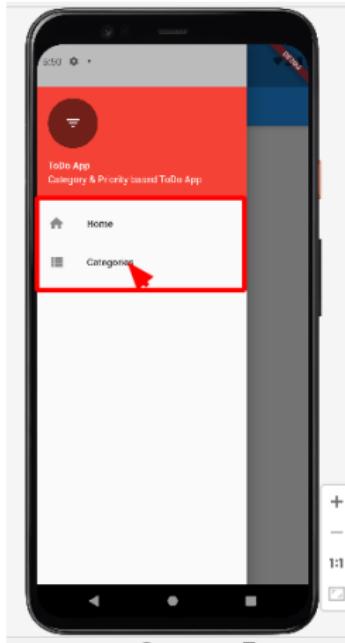


Figura: Abrir el menú lateral y dar clic en categories.

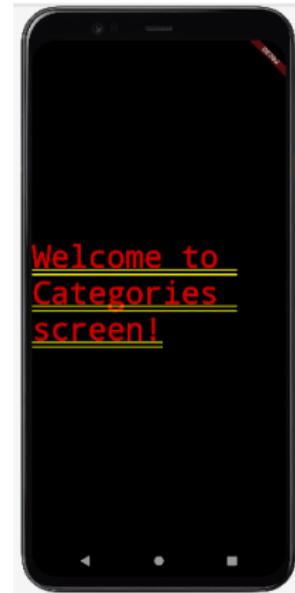
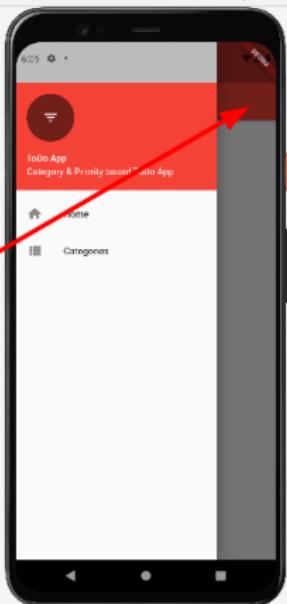


Figura: Se abre la siguiente pantalla, modificaremos enseguida.

The screenshot shows the Android Studio interface. On the left, the code editor displays `CategoriesScreen.dart`. The code defines a `CategoriesScreen` class that extends `StatefulWidget`. It overrides the `createState` method to return a `_CategoriesScreenState` object. The `build` method returns a `Scaffold` widget. Inside the `Scaffold`, there is an `AppBar` with the title "ToDo App" and a `body` containing a `Center` widget with the text "Welcome to Categories screen!". On the right, the emulator window shows a smartphone displaying the application. The top bar is blue with the title "ToDo App". Below it, the main screen has the text "Welcome to Categories screen!". A red arrow points from the highlighted code in the code editor to the `AppBar` in the emulator's screenshot.

```
dart × home_screen.dart × app.dart × drawer_navigation.dart × categories_screen.dart × Emulator: Pixel4 ×
1 import 'package:flutter/material.dart';
2
3 class CategoriesScreen extends StatefulWidget {
4   @override
5   _CategoriesScreenState createState() => _CategoriesScreenState();
6 }
7
8 class _CategoriesScreenState extends State<CategoriesScreen> {
9   @override
10  Widget build(BuildContext context) {
11    return Scaffold(
12      appBar: AppBar(
13        title: Text("ToDo App")
14      ), // AppBar
15      body: Center(
16        child: Text("Welcome to Categories screen!"),
17      ), // Center
18    ); // Scaffold
19  }
20
21 }
```

Figura: Cambiamos el container por un Scaffold agregando AppBar y el child por el body del scaffold y vemos el cambio



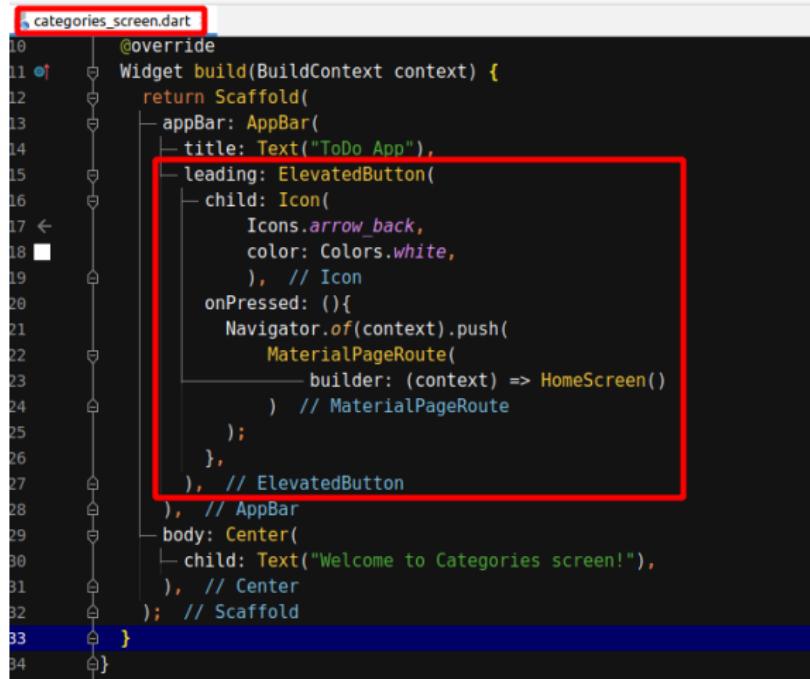
```
dart > home_screen.dart > app.dart > drawer_navigation.dart > categories_screen.dart > Emulator: Pixel4 >
```

```
1 import 'package:flutter/material.dart';
2 import 'package:todo_app/screens/home_screen.dart'
3
4 class App extends StatelessWidget {
5   @override
6   Widget build(BuildContext context) {
7     return MaterialApp(
8       home: HomeScreen(),
9       theme: ThemeData(
10         primarySwatch: Colors.red,
11       ), // ThemeData
12     ); // MaterialApp
13   }
14 }
15
```

MEJORAMOS EL ESTILO DE LA
APLICACIÓN AGREGANDOLE
TEMA.

Figura: Se añade tema y las appbars y otros elementos no serán de distinto estilo.

Esto tiene un problema



```
categories_screen.dart
10  @override
11  Widget build(BuildContext context) {
12    return Scaffold(
13      appBar: AppBar(
14        title: Text("ToDo App"),
15        leading: ElevatedButton(
16          child: Icon(
17            Icons.arrow_back,
18            color: Colors.white,
19          ), // Icon
20          onPressed: () {
21            Navigator.of(context).push(
22              MaterialPageRoute(
23                builder: (context) => HomeScreen()
24              ) // MaterialPageRoute
25            );
26          },
27        ), // ElevatedButton
28      ), // AppBar
29      body: Center(
30        child: Text("Welcome to Categories screen!"),
31      ), // Center
32    ); // Scaffold
33  }
34}
```

Figura: Al momento de regresar a **HomeScreen** se visualiza abierto el menú Drawer, para corregirlo agregamos el código correspondiente en **categories_screen.dart**

The screenshot shows the Android Studio code editor with the file `CategoriesScreenState.dart`. The code defines a `Scaffold` with a `MaterialPageRoute` builder, an `ElevatedButton`, an `AppBar`, and a `Center` body containing a `Text` widget. A `FloatingActionButton` is added to the `body` with an `onPressed` callback that calls `_showFormInDialog`. The `_showFormInDialog` method uses `showDialog` to present an `AlertDialog` titled "Category Form". The emulator on the right shows a smartphone running the app, with a red floating action button in the bottom right corner labeled "+". A yellow box highlights the `FloatingActionButton` code, and another yellow box highlights the `_showFormInDialog` method.

```
9  class _CategoriesScreenState extends State<CategoriesScreen> {
10    ...
11    ...
12    ...
13    ...
14    ...
15    ...
16    ...
17    ...
18    ...
19    ...
20    ...
21    ...
22    ...
23    ...
24    ...
25    ...
26    ...
27    ...
28    ...
29    ...
30    ...
31    ...
32    ...
33    ...
34    ...
35    ...
36    ...
37    ...
38    ...
39    ...
40    ...
41    ...
42    ...
43    ...
44    ...
45    ...
46    ...
47    ...
48    ...
}
```

Figura: Agregamos un botón flotante en `_CategoriesScreenState` que al darle Tap mostará un dialogo.

The screenshot shows the code for a category form dialog in Dart, with specific sections highlighted by yellow boxes. The code defines an AlertDialog with a title, content (containing two text fields), and actions (Save and Cancel buttons). To the right, a mobile application preview on a Pixel 4 emulator shows the 'Category Form' screen with two text input fields and a bottom row of 'Save' and 'Cancel' buttons.

```
41 _showFormInDialog(BuildContext context){  
42     return showDialog(context: context,  
43         barrierDismissible: true,  
44         builder: (param){  
45             return AlertDialog(  
46                 actions: [  
47                     TextButton(onPressed: (){},  
48                         child: Text("Save")), // TextButton  
49                     TextButton(onPressed: (){},  
50                         child: Text("Cancel")), // TextButton  
51                 ],  
52                 title: Text("Category Form"),  
53                 content: SingleChildScrollView(  
54                     child: Column(  
55                         children: [  
56                             TextField(  
57                                 decoration: InputDecoration(  
58                                     labelText: "Category name",  
59                                     hintText: "Write category name"  
60                                 ), // InputDecoration  
61                             ), // TextField  
62                             TextField(  
63                                 decoration: InputDecoration(  
64                                     labelText: "Category description",  
65                                     hintText: "Write category description"  
66                                 ), // InputDecoration  
67                             ), // TextField  
68                         ],  
69                     ), // Column  
70                 ), // SingleChildScrollView  
71             ); // AlertDialog  
72 } // showFormInDialog
```

Figura: Agregamos campos de texto para el nombre y descripción de la categoría y los respectivos botones de guardar y cancelar

Creamos archivos modelo y servicios para categoría

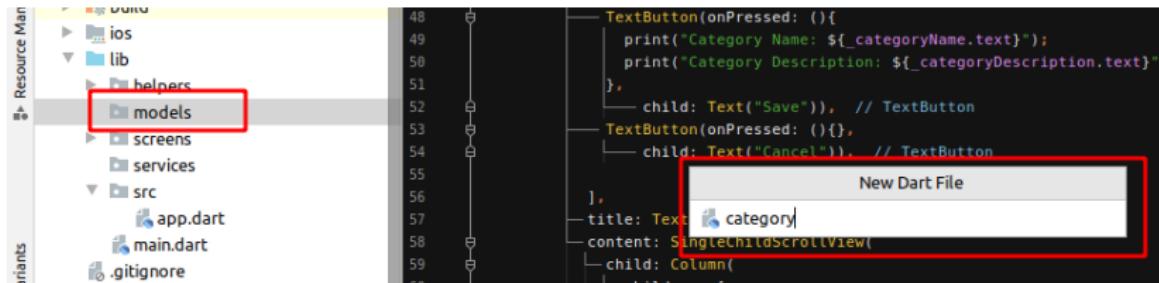


Figura: Creamos un nuevo paquete llamado **models** y añadimos un archivo llamado **category.dart**

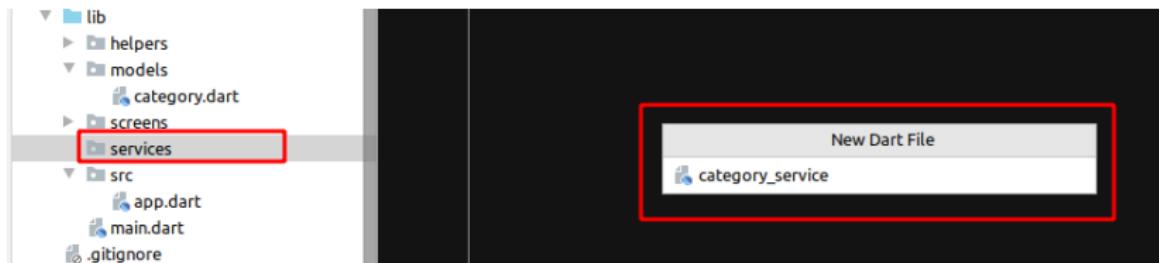
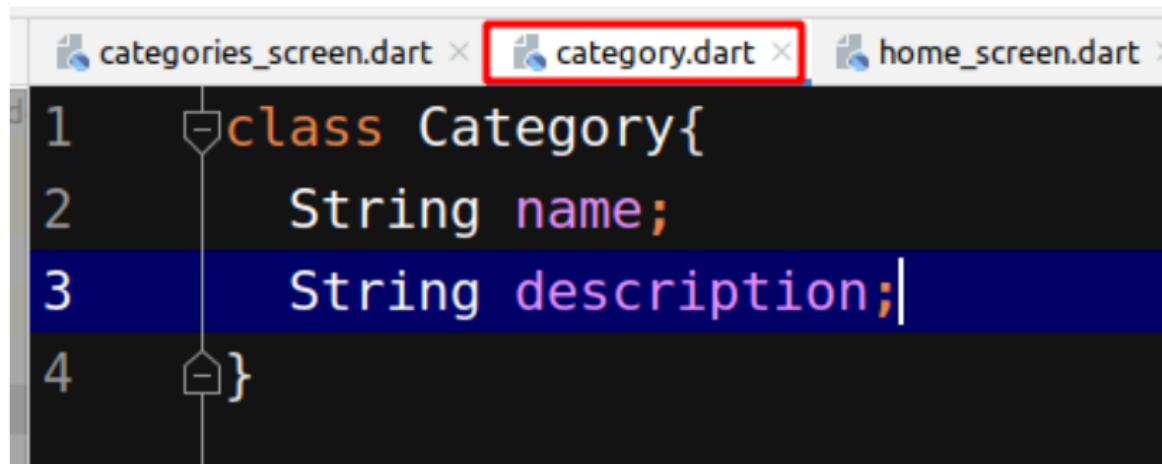


Figura: Agregamos un nuevo paquete llamado **services** y un archivo de servicio para categoría nombrado **category_service.dart**

Codificamos modelo para category

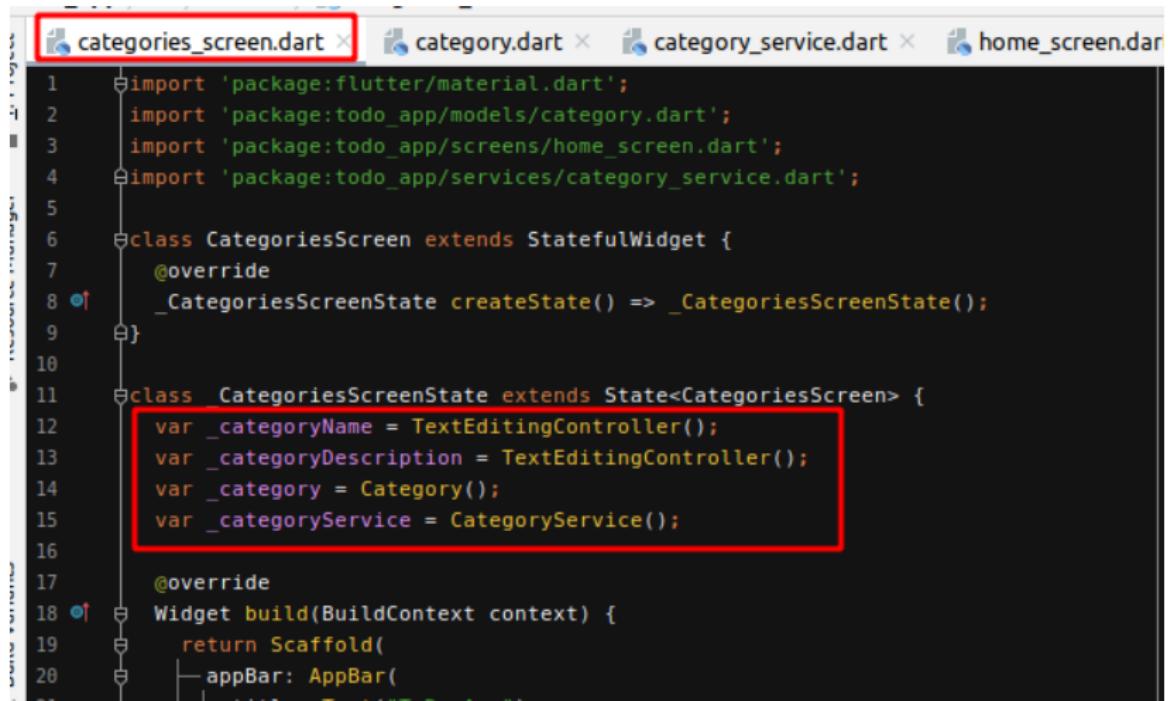


```
categories_screen.dart × category.dart × home_screen.dart >
1 class Category{
2     String name;
3     String description;
4 }
```

Figura: Archivo category.dart con atributos de nombre y descripción

```
ice.dart
|- categories_screen.dart × category.dart × category_service.dart × home_screen.dart ×
1 import 'package:todo_app/models/category.dart'
2
3 class CategoryService{
4     saveCategory(Category category){
5         print(category.name);
6         print(category.description);
7     }
8 }
```

Figura: Codifica la parte inicial de CategoryService creando un método guardar que provisionalmente imprimirá en consola el nombre y descripción del modelo Category.



```
categories_screen.dart category.dart category_service.dart home_screen.dart
1 import 'package:flutter/material.dart';
2 import 'package:todo_app/models/category.dart';
3 import 'package:todo_app/screens/home_screen.dart';
4 import 'package:todo_app/services/category_service.dart';
5
6 class CategoriesScreen extends StatefulWidget {
7   @override
8   _CategoriesScreenState createState() => _CategoriesScreenState();
9 }
10
11 class _CategoriesScreenState extends State<CategoriesScreen> {
12   var _categoryName = TextEditingController();
13   var _categoryDescription = TextEditingController();
14   var _category = Category();
15   var _categoryService = CategoryService();
16
17   @override
18   Widget build(BuildContext context) {
19     return Scaffold(
20       appBar: AppBar(
21         title: Text('Categorías')
22       ),
23       body: Center(
24         child: Column(
25           mainAxisAlignment: MainAxisAlignment.spaceEvenly,
26           children: [
27             TextField(
28               controller: _categoryName,
29               decoration: InputDecoration(
30                 labelText: 'Nombre de la categoría'
31               )
32             ),
33             TextField(
34               controller: _categoryDescription,
35               decoration: InputDecoration(
36                 labelText: 'Descripción de la categoría'
37               )
38             ),
39             ElevatedButton(
40               onPressed: () {
41                 _category.name = _categoryName.text;
42                 _category.description = _categoryDescription.text;
43                 _categoryService.create(_category);
44                 Navigator.pop(context);
45               },
46               child: Text('Crear')
47             )
48           ],
49         )
50       )
51     );
52   }
53 }
```

Figura: Volvemos a **_CategoriesScreenState** y añadimos las instancias que necesitamos para manejar el modelo **Category** y su **servicio** al igual que para obtener datos de los **campos de texto**.

```
categories_screen.dart × category.dart × category_service.dart × home_screen.d
46     }
47     _showFormInDialog(BuildContext context){
48         return showDialog(context: context,
49                         barrierDismissible: true,
50                         builder: (param){
51                             return AlertDialog(
52                                 actions: [
53                                     TextButton(onPressed: (){
54                                         _category.name = _categoryName.text;
55                                         _category.description = _categoryDescription.text;
56                                         _categoryService.saveCategory(_category);
57                                     },
58                                     child: Text("Save")), // TextButton
59                                     TextButton(onPressed: (){}, // TextButton
60                                     child: Text("Cancel"))
61                               ],
62                               title: Text("Category Form"),
63                               content: SingleChildScrollView(
64                                   child: Column(
```

Figura: Dentro del método de `_showFormInDialog` en las acciones modificamos el botón de **save** para que al darle Tap asigne valores al modelo y active el servicio en su método guardar.

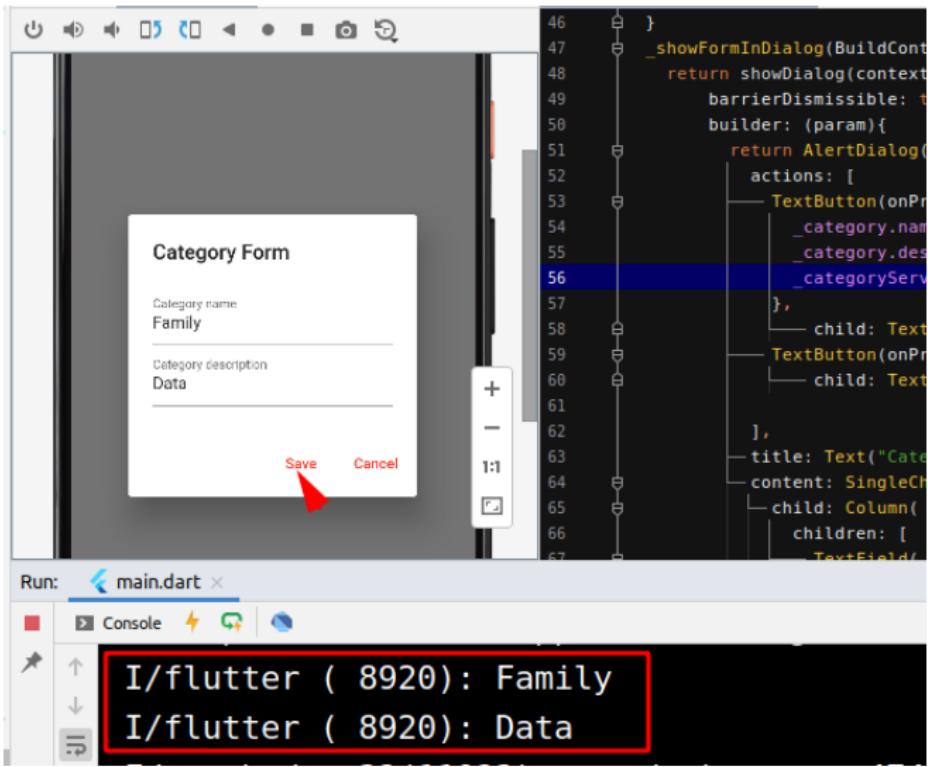


Figura: Usando el servicio y el modelo en el método guardar del dialog.

```
todo_app pubspec.yaml categories_screen.dart category.dart category_service.dart home_screen.dart drawer_navigation.dart
dependencies:
  flutter:
    sdk: flutter
  sqflite: ^1.3.0
  path_provider: ^2.0.1
  # The following adds the Cupertino Icons font to your application
  # Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.2
Document 1/1 > dependencies:
Terminal: Local
thinktacho@thinktacho:~/AndroidStudioProjects/todo_app$ flutter pub get
Running "flutter pub get" in todo_app...                                6,7s
```

Figura: Abre el archivo **pubspec.yaml** y busca dependencia agrega las dependencias de **sqflite** y **path_provider** y ejecuta en consola el comando **pub get**.

The screenshot shows a file tree on the left and a code editor on the right. The file tree includes 'lib' (with 'helpers', 'models', and 'repositories' folders), 'screens' (with 'categories_screen.dart' and 'home_screen.dart' files), 'services', and 'src' (with 'app.dart' file). The code editor displays a Dart file with the following content:

```
25 |   dependencies: {
26 |     "sqflite": "^1.3.0",
27 |     "path_provider": "^2.0.1"
28 |
29 |
30 |   # The following adds the CupertinoIcons class
31 |   # Use with the CupertinoIcons class
```

A red box highlights the 'repositories' folder in the file tree. Another red box highlights the 'db_connection' file in the code editor's suggestion bar.

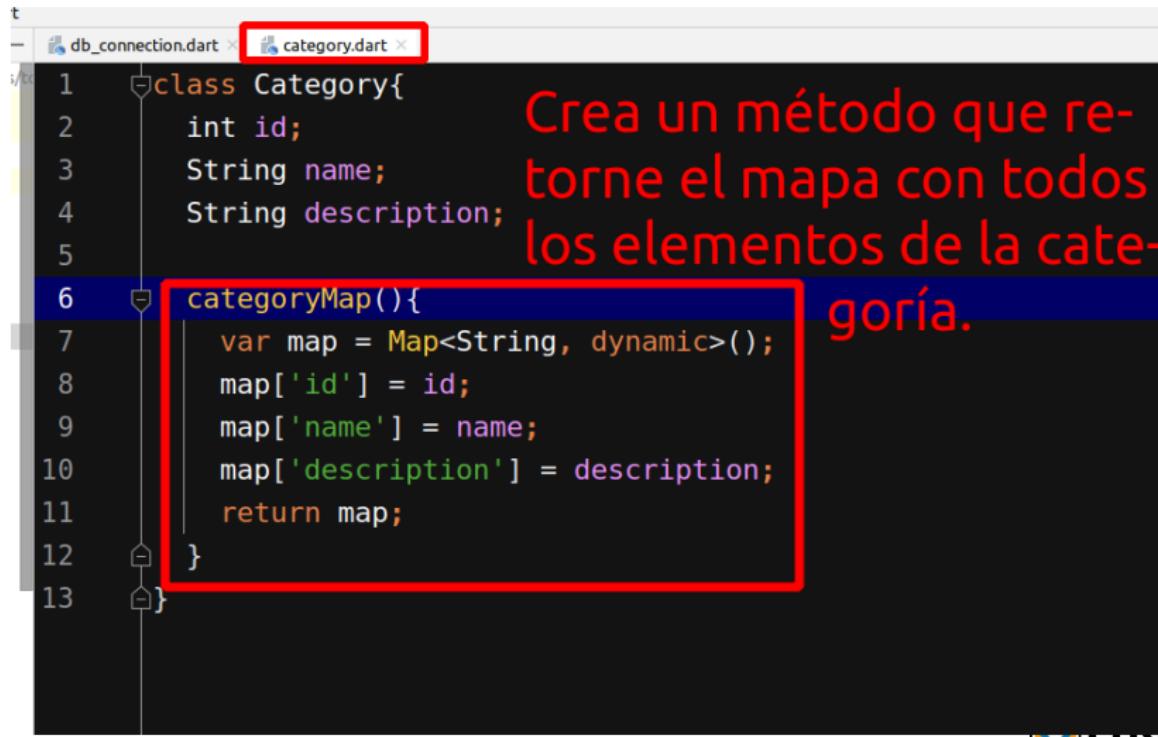
Figura: Crea un nuevo paquete llamado **repositories** en el que agrega un archivo que nombrarás **db_connection.dart**

Codifica el archivo db_connection.dart

```
db_connection.dart
1 import 'package:path/path.dart';
2 import 'package:path_provider/path_provider.dart';      Bibliotecas utilizadas
3 import 'package:sqflite/sqflite.dart';
4
5 class DatabaseConnection {
6     setDatabase() async {
7         var directory = await getApplicationDocumentsDirectory();
8         var path = join(directory.path, "db_todo");
9         var database =
10             await openDatabase(path, version: 1, onCreate: _onCreatingDatabase);
11         return database;
12     }
13
14     _onCreatingDatabase(Database db, int version) async {
15         await db.execute(
16             "CREATE TABLE categories(id INTEGER PRIMARY KEY, name TEXT, description TEXT)");
17         await db.execute(
18             "CREATE TABLE todos(id INTEGER PRIMARY KEY, title TEXT, description TEXT,"
19             " category TEXT, todoDate TEXT, isFinished INTEGER)");
20     }
21 }
```

Figura: Este archivo servirá para crear o acceder a la base de datos creada

Modifica el modelo de categorias y crea el modelo de Todos



The screenshot shows a code editor with two tabs: db_connection.dart and category.dart. The category.dart tab is active and highlighted with a red border. The code defines a Category class with properties id, name, and description, and a categoryMap() method that returns a Map<String, dynamic>. A red box highlights the categoryMap() method.

```
1 class Category{  
2     int id;  
3     String name;  
4     String description;  
5  
6     categoryMap(){  
7         var map = Map<String, dynamic>();  
8         map['id'] = id;  
9         map['name'] = name;  
10        map['description'] = description;  
11        return map;  
12    }  
13}
```

Crea un método que retorne el mapa con todos los elementos de la categoría.

Crea y codifica el archivo **todo.dart** en el paquete **models**

```
db_connection.dart × category.dart × todo.dart ×  
1 class Todo{  
2     int id;  
3     String title;  
4     String description;  
5     String category;  
6     String todoDate;  
7     int isFinished;  
8     todoMap(){  
9         var map = Map<String, dynamic>();  
10        map["id"] = id;  
11        map["title"] = title;  
12        map["description"] = description;  
13        map["category"] = category;  
14        map["todoDate"] = todoDate;  
15        map["isFinished"] = isFinished;  
16        return map;  
17    }  
18 }
```

Clase Todo con 6 propiedades

Mapa que retorna una representación de todas las propiedades de un "Todo"

Figura: Ubicado en la carpeta models

Crea en el paquete **repositories** un archivo llamado **repository.dart**

```
repository.dart
1 import 'package:to_do_app/repositories/db_connection.dart';
2 import 'package:sqflite/sqflite.dart';
3 class Repository{
4     DatabaseConnection _connection;
5     Repository(){
6         _connection = DatabaseConnection();
7     }
8     static Database _database;
9     Future<Database> get database async{
10        if(_database != null){
11            return _database;
12        }
13        _database = await _connection.setDatabase();
14        return _database;
15    }
16    save(table, data) async{
17        var conn = await database;
18        return await conn.insert(table, data);
19    }
}
```

Imports

Creamos un objeto conexión para acceder a la base de datos

En el constructor crea la instancia para poder tener acceso a la DB.

Crea una sola instancia de la base de datos y obtenla en caso de que ya este creada con anterioridad.

Método para guardar un nuevo registro de la tabla que se pasa como argumento.

Figura: Primera parte del archivo **repository.dart**

repository.dart segunda parte en el paquete repositories

```
19
20     getAll(table) async{
21         var conn = await database;
22         return await conn.query(table);
23     }
24     getById(String table, itemId) async {
25         var conn = await database;
26         return await conn.query(table, where: 'id=?', whereArgs: [itemId]);
27     }
28     update(String table, data) async {
29         var conn = await database;
30         return await conn.update(table, data, where: 'id=?', whereArgs: [data['id']]);
31     }
32     delete(String table, itemId) async {
33         var conn = await database;
34         return await conn.rawDelete("DELETE from $table WHERE id = $itemId");
35     }
36     getByColumnName(String table, String columnName, String columnValue) async {
37         var conn = await database;
38         return await conn.query(table, where: "$columnName = ?", whereArgs: [columnValue]);
39     }
40 }
41 
```

Método para obtener todos los registros pertenecientes a una tabla

Método para obtener un elemento a partir del id.

Método para actualizar un registro a partir del id

Método para borrar un registro a partir de un id.

Método para obtener un registro a partir de su valor en alguna columna dada.

Figura: Segunda parte del archivo **repository.dart**

Complementa el archivo **category_service.dart**

```
to_do_app lib services category_service.dart
  db_connection.dart category.dart todo.dart repository.dart category_service.dart

1 import 'package:to_do_app/models/category.dart';
2 import 'package:to_do_app/repositories/repository.dart';
3 class CategoryService{
4     Repository _repository;
5     CategoryService(){
6         _repository = Repository();
7     }
8     saveCategory(Category category) async{
9         return await _repository.save("categories", category.categoryMap());
10    }
11    getCategories() async{
12        return await _repository.getAll("categories");
13    }
14    getCategoryById(categoryId) async{
15        return await _repository.getById('categories', categoryId);
16    }
17    updateCategory(Category category) async {
18        return await _repository.update('categories', category.categoryMap());
19    }
20    deleteCategory(categoryId) async {
21        return await _repository.delete('categories', categoryId);
22    }
23 }
```

Traemos el modelo de category y los métodos repository para realizar los cambios directos en la DB.

Crea instancia de repository e inicializado.

Guardando categoria

Obtener todas las categorias

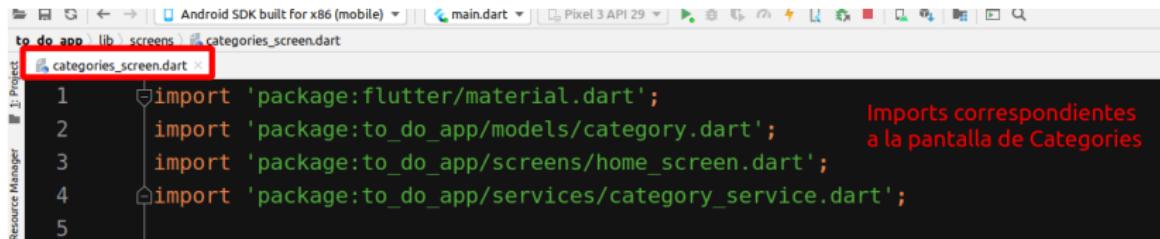
Obtener una categoría por su id

Actualizar una categoría.

Borra una categoría a partir de su id.

Figura: Complementando el archivo **category_service.dart** para tener los métodos necesarios para gestión de categorías.

Complementamos la pantalla `categories_screen.dart`



The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it, the code editor displays the `categories_screen.dart` file under the `lib/screens` directory. The code contains five imports:

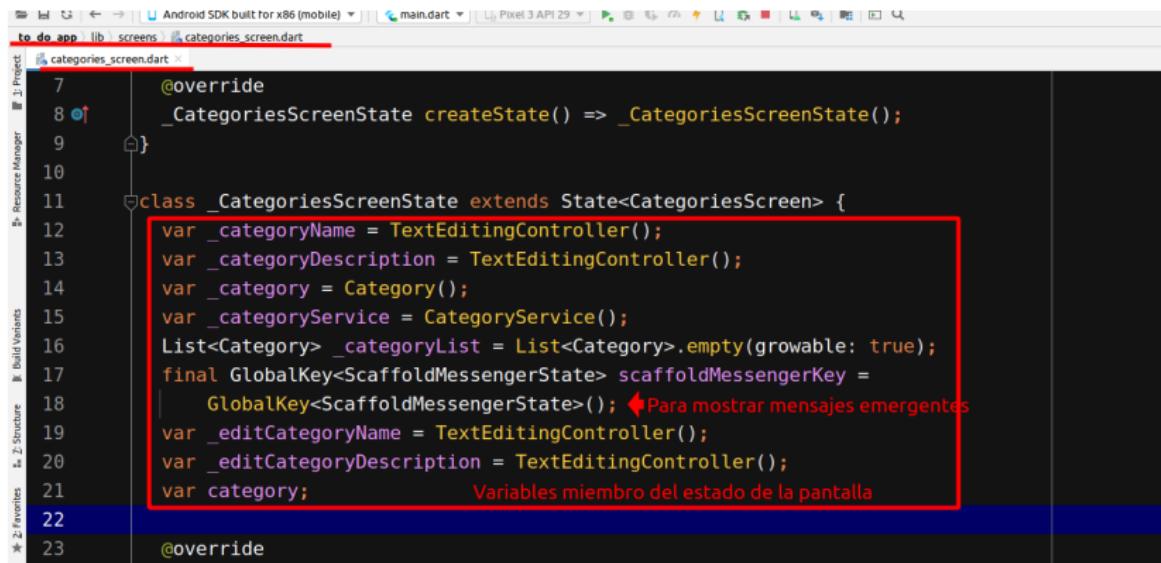
```
1 import 'package:flutter/material.dart';
2 import 'package:to_do_app/models/category.dart';
3 import 'package:to_do_app/screens/home_screen.dart';
4 import 'package:to_do_app/services/category_service.dart';
5
```

A red box highlights the file name in the project navigation bar and the file tab in the code editor. A red arrow points from the text "Imports correspondientes a la pantalla de Categories" to the fourth import statement.

Imports correspondientes
a la pantalla de Categories

Figura: Agrega los imports faltantes para las operaciones de gestión de categorias.

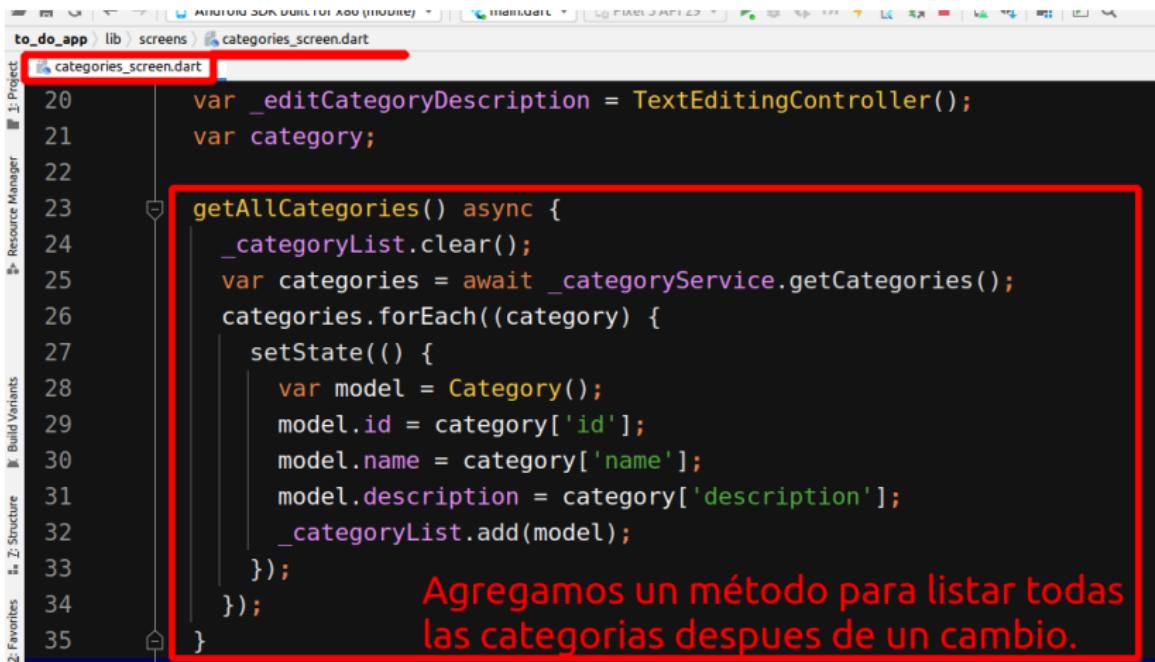
Agregamos las variables miembro faltantes para las operaciones de Categories



The screenshot shows the Android Studio interface with the file `categories_screen.dart` open. The code defines a state class for a screen. A red box highlights the member variables and their descriptions:

```
7 @override
8   _CategoriesScreenState createState() => _CategoriesScreenState();
9 }
10
11 class _CategoriesScreenState extends State<CategoriesScreen> {
12   var _categoryName = TextEditingController();
13   var _categoryDescription = TextEditingController();
14   var _category = Category();
15   var _categoryService = CategoryService();
16   List<Category> _categoryList = List<Category>.empty(growable: true);
17   final GlobalKey<ScaffoldMessengerState> scaffoldMessengerKey =
18     GlobalKey<ScaffoldMessengerState>(); ◆ Para mostrar mensajes emergentes
19   var _editCategoryName = TextEditingController();
20   var _editCategoryDescription = TextEditingController();
21   var category; Variables miembro del estado de la pantalla
22
23 @override
```

Añadimos método para cargar todas las categorías existentes

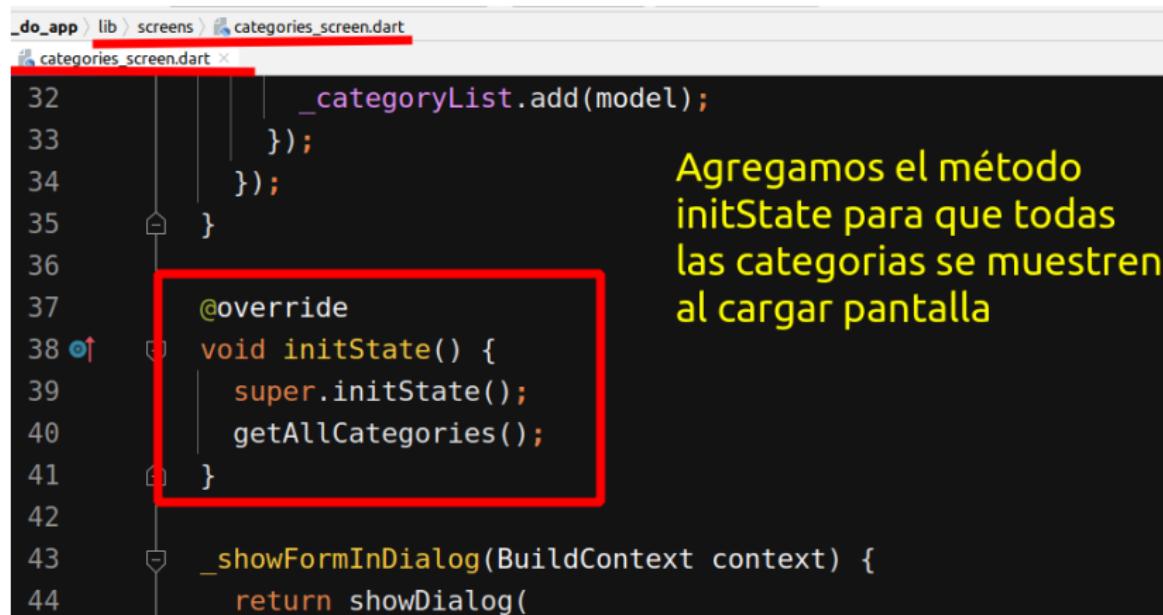


```
20     var _editCategoryDescription = TextEditingController();
21     var category;
22
23     getAllCategories() async {
24         _categoryList.clear();
25         var categories = await _categoryService.getCategories();
26         categories.forEach((category) {
27             setState(() {
28                 var model = Category();
29                 model.id = category['id'];
30                 model.name = category['name'];
31                 model.description = category['description'];
32                 _categoryList.add(model);
33             });
34         });
35     }
```

Agregamos un método para listar todas las categorias despues de un cambio.

Figura: Método que cambia el listado de categorias que puede utilizarse al cargar la pantalla, agregar nuevo, actualizar o borrar un registro de categorias.

Sobrescribimos el método initState con el listado al iniciar pantalla



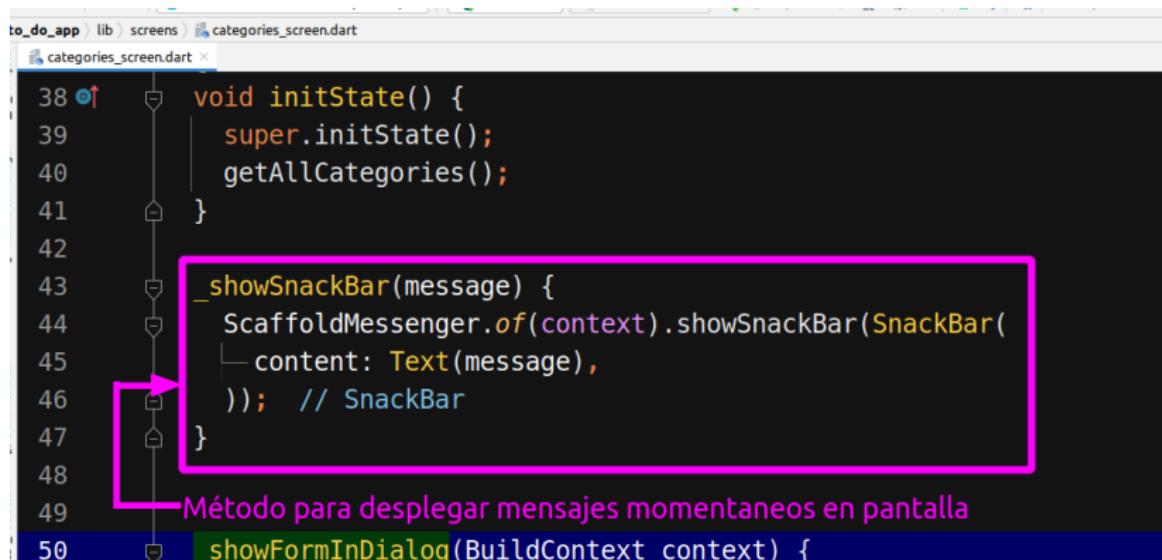
```
_do_app/lib/screens/categories_screen.dart
categories_screen.dart

32           _categoryList.add(model);
33       });
34   });
35 }
36
37 @override
38 void initState() {
39     super.initState();
40     getAllCategories();
41 }
42
43 _showFormInDialog(BuildContext context) {
44     return showDialog(
```

Agregamos el método initState para que todas las categorías se muestren al cargar pantalla

Figura: Método initState sobrescrito

Coloca el método `_showSnackBar(message)`



```
to_do_app > lib > screens > categories_screen.dart
categories_screen.dart

38 void initState() {
39   super.initState();
40   getAllCategories();
41 }
42
43 _showSnackBar(message) {
44   ScaffoldMessenger.of(context).showSnackBar(SnackBar(
45     content: Text(message),
46   )); // SnackBar
47 }
48
49 showFormInDialog(BuildContext context) {
```

Método para desplegar mensajes momentáneos en pantalla

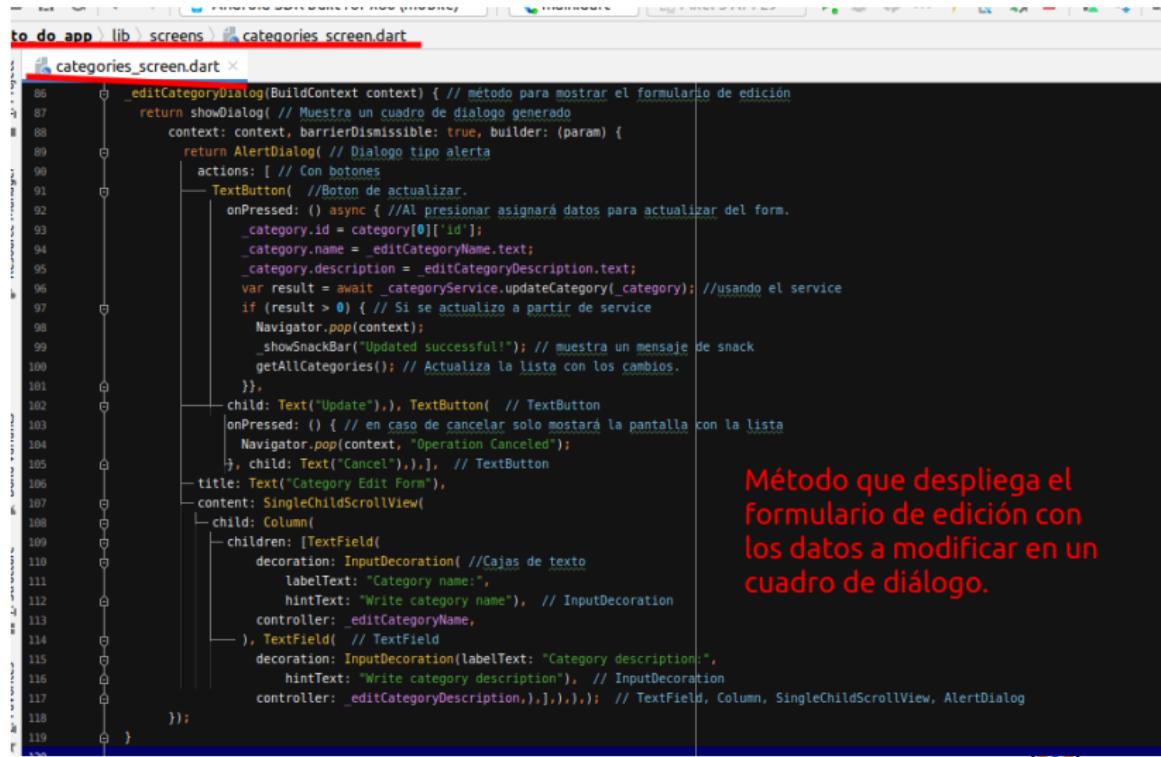
Figura: Método para desplegar mensajes llamados Snack para desplegar errores o operaciones correctamente realizadas con categorías

Complemeta el método para mostrar el formulario de registro de categorías

```
categories_screen.dart
50 _showFormInDialog(BuildContext context) { //Muestra el formulario para guardar
51     return showDialog( //retorna un dialog
52         context: context, //El elemento actual
53         barrierDismissible: true, //Que se pueda salir al dar tap fuera de su área
54         builder: (param) { //método que lo genera
55             return AlertDialog( //Estilo tipo alerta
56                 actions: [TextButton( //Botones
57                     onPressed: () async {
58                         _category.name = _categoryName.text; //asigne al modelo lo del form
59                         _category.description = _categoryDescription.text;
60                         var result = await _categoryService.saveCategory(_category); //Usando el servicio de category para persistencia
61                         if (result > 0) {
62                             Navigator.pop(context); //Sal del cuadro de dialogo.
63                             showSnackBar("Saved successful!"); //Muestra el mensaje en la barra de snack
64                             getAllCategories(); //Actualiza el listado de categorias
65                             child: Text("Save"), //Texto que se despliega para el botón
66                         ), TextButton( //TextButton
67                             onPressed: () { //Al cancelar cierra solo el cuadro de dialogo.
68                                 Navigator.pop(context);
69                             },
70                             child: Text("Cancel")),
71                     title: Text("Category Form"), //Título del cuadro de dialogo
72                     content: SingleChildScrollView(
73                         child: Column( //Muestra en una columna cajas de texto para registrar una categoría
74                             children: [TextField(
75                                 decoration: InputDecoration(
76                                     labelText: "Category name",
77                                     hintText: "Write category name"),
78                                 controller: _categoryName,
79                                 ), TextField(decoration: InputDecoration( //TextField
80                                     labelText: "Category description",
81                                     hintText: "Write category description"),
82                                 controller: _categoryDescription,)],),),
83                 );
84             );
85         });
86     }
87 }
```

Método para mostrar el dialogo del formulario para registrar una categoría.

Agrega el formulario de edición en el método `_editCategoryDialog`

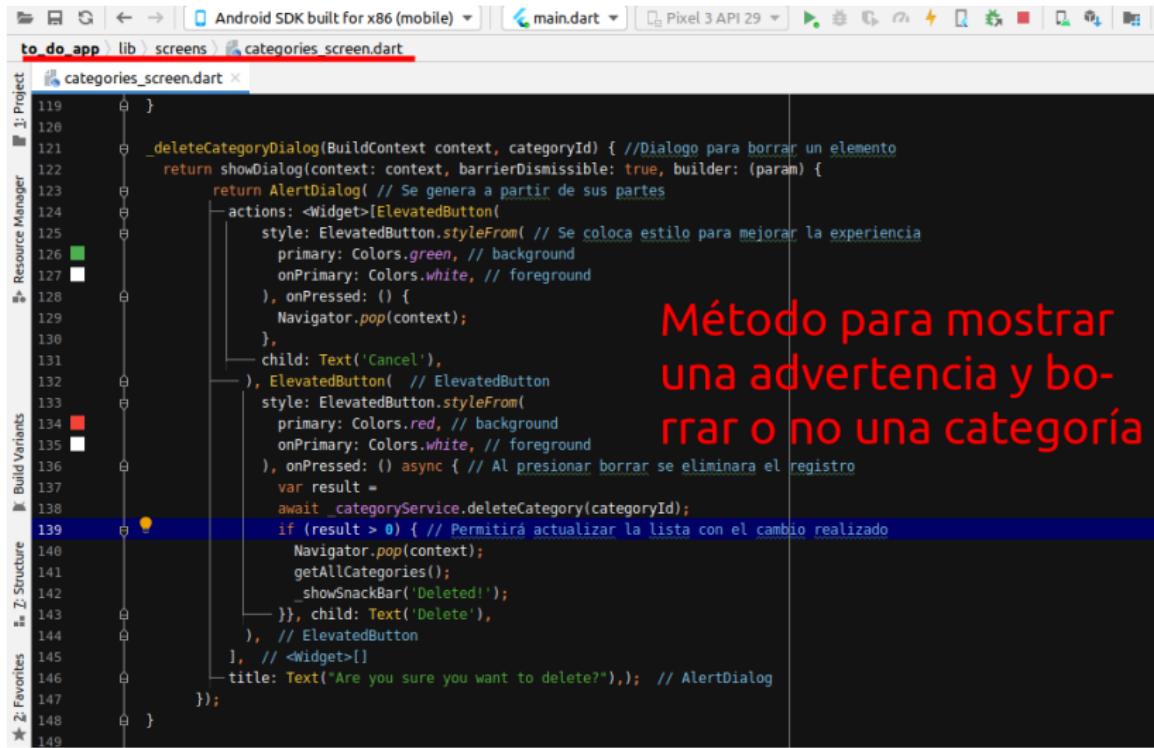


```
to do app lib \ screens \ categories screen.dart
categories_screen.dart

86     _editCategoryDialog(BuildContext context) { // método para mostrar el formulario de edición
87         return showDialog( // Muestra un cuadro de dialogo generado
88             context: context, barrierDismissible: true, builder: (param) {
89                 return AlertDialog( // Dialogo tipo alerta
90                     actions: [ // Con botones
91                         TextButton( //Boton de actualizar,
92                             onPressed: () async { //Al presionar asignará datos para actualizar del form.
93                                 _category.id = category[0]['id'];
94                                 _category.name = _editCategoryName.text;
95                                 _category.description = _editCategoryDescription.text;
96                                 var result = await _categoryService.updateCategory(_category); //usando el service
97                                 if (result > 0) { // Si se actualizo a partir de service
98                                     Navigator.pop(context);
99                                     showSnackBar("Updated successful!"); // muestra un mensaje de snack
100                                     getAllCategories(); // Actualiza la lista con los cambios.
101                                 },
102                                 child: Text("Update"),), TextButton( // TextButton
103                                     onPressed: () { // en caso de cancelar solo mostará la pantalla con la lista
104                                         Navigator.pop(context, "Operation Canceled");
105                                     }, child: Text("Cancel"),), // TextButton
106                     title: Text("Category Edit Form"),
107                     content: SingleChildScrollView(
108                         child: Column(
109                             children: [TextField(
110                                 decoration: InputDecoration( //Cajas de texto
111                                     labelText: "Category name:",
112                                     hintText: "Write category name"), // InputDecoration
113                                     controller: _editCategoryName,
114                                 ), TextField( // TextField
115                                     decoration: InputDecoration(labelText: "Category description:",
116                                     hintText: "Write category description"), // InputDecoration
117                                     controller: _editCategoryDescription,)],), // TextField, Column, SingleChildScrollView, AlertDialog
118                     );
119                 });
120             });
121         );
122     }
123 }
```

Método que despliega el formulario de edición con los datos a modificar en un cuadro de diálogo.

Añade el método para borrar definitivamente un registro

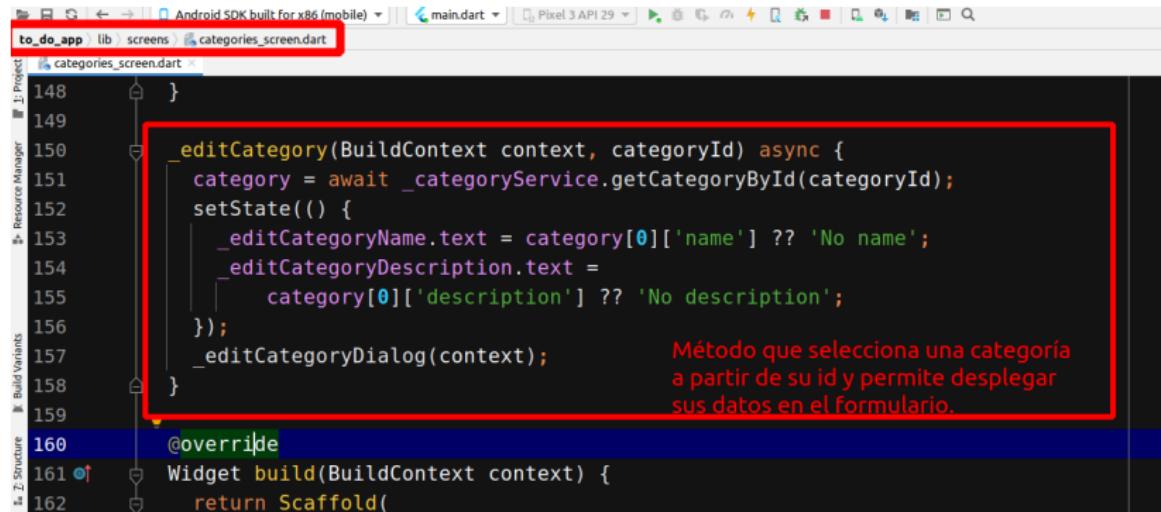


The screenshot shows the Android Studio interface with the project navigation bar at the top. The main editor window displays the code for `categories_screen.dart`. The code implements a dialog for deleting a category. It uses `AlertDialog` with two `ElevatedButton` actions: one for canceling and one for confirming deletion. The confirmation action triggers a `Navigator.pop` to dismiss the dialog and then performs a `deleteCategory` call on the `_categoryService`. If successful, it updates the list and shows a `Deleted!` message. The code is annotated with comments explaining its purpose.

```
119     }
120
121     _deleteCategoryDialog(BuildContext context, categoryId) { // Dialogo para borrar un elemento
122         return showDialog(context: context, barrierDismissible: true, builder: (param) {
123             return AlertDialog( // Se genera a partir de sus partes
124                 actions: <Widget>[ElevatedButton(
125                     style: ElevatedButton.styleFrom( // Se coloca estilo para mejorar la experiencia
126                         primary: Colors.green, // background
127                         onPrimary: Colors.white, // foreground
128                         onPressed: () {
129                             Navigator.pop(context);
130                         },
131                         child: Text('Cancel'),
132                     ), ElevatedButton( // ElevatedButton
133                         style: ElevatedButton.styleFrom(
134                             primary: Colors.red, // background
135                             onPrimary: Colors.white, // foreground
136                             onPressed: () async { // Al presionar borrar se eliminará el registro
137                                 var result =
138                                     await _categoryService.deleteCategory(categoryId);
139                                 if (result > 0) { // Permitirá actualizar la lista con el cambio realizado
140                                     Navigator.pop(context);
141                                     getAllCategories();
142                                     showSnackBar('Deleted!');
143                                 }, child: Text('Delete'),
144                             ),
145                         ],
146                         title: Text("Are you sure you want to delete?"));
147                     );
148                 );
149             });
150         }
151     }
```

Método para mostrar una advertencia y borrar o no una categoría

Coloca el método `_editCategory` para cargar los datos a modificar de una categoría por su id



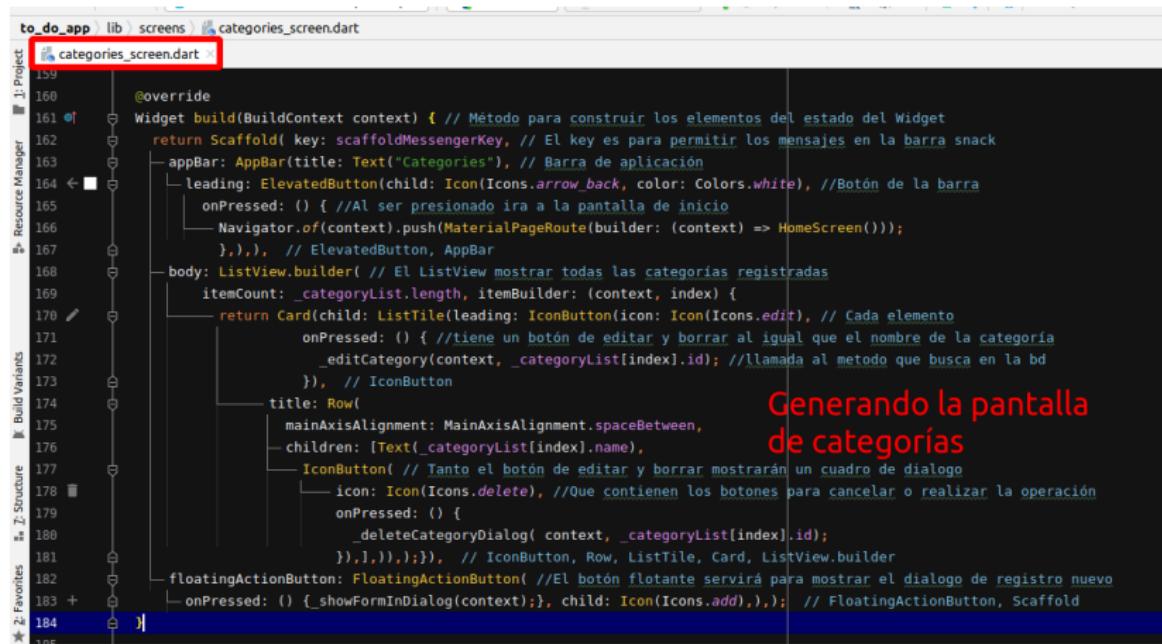
The screenshot shows the Android Studio interface with the file `categories_screen.dart` open. The code is as follows:

```
148 }
149
150     editCategory(BuildContext context, categoryId) async {
151         category = await _categoryService.getCategoryById(categoryId);
152         setState(() {
153             _editCategoryName.text = category[0]['name'] ?? 'No name';
154             _editCategoryDescription.text =
155                 category[0]['description'] ?? 'No description';
156         });
157         _editCategoryDialog(context);
158     }
159
160     @override
161     Widget build(BuildContext context) {
162         return Scaffold(
```

A red box highlights the method `editCategory`. A callout bubble to the right of the code provides a descriptive annotation:

Método que selecciona una categoría a partir de su id y permite desplegar sus datos en el formulario.

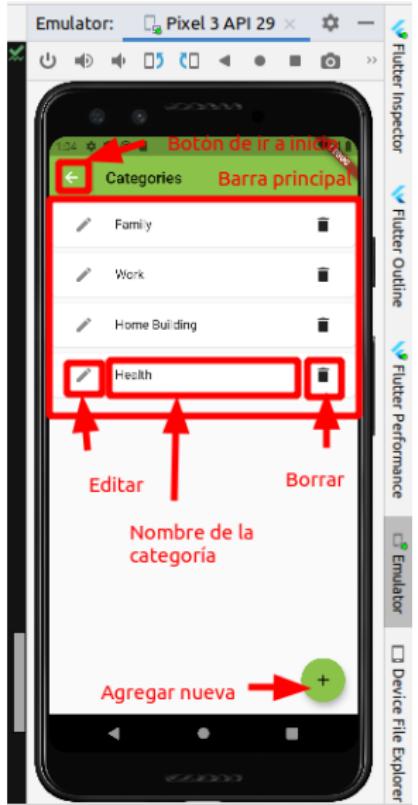
Complemeta el método build para que la pantalla se genere con appBar, ListView y el botón flotante



```
to_do_app/lib/screens/categories_screen.dart
159
160
161 @override
162 Widget build(BuildContext context) { // Método para construir los elementos del estado del Widget
163   return Scaffold( key: scaffoldMessengerKey, // El key es para permitir los mensajes en la barra snack
164     appBar: AppBar(title: Text("Categories"), // Barra de aplicación
165       leading: ElevatedButton(child: Icon(Icons.arrow_back, color: Colors.white), //Botón de la barra
166         onPressed: () { //Al ser presionado ira a la pantalla de inicio
167           Navigator.of(context).push(MaterialPageRoute(builder: (context) => HomeScreen()));
168         },), // ElevatedButton, AppBar
169     body: ListView.builder( // El ListView mostrar todas las categorías registradas
170       itemCount: _categoryList.length, itemBuilder: (context, index) {
171         return Card(child: ListTile(leading: IconButton(icon: Icon(Icons.edit), // Cada elemento
172           onPressed: () { //Tiene un botón de editar y borrar al igual que el nombre de la categoría
173             _editCategory(context, _categoryList[index].id); //llamada al método que busca en la bd
174           },), // IconButton
175           title: Row(
176             mainAxisSize: MainAxisSize.spaceBetween,
177             children: [Text(_categoryList[index].name),
178               IconButton( // Tanto el botón de editar y borrar mostrarán un cuadro de dialogo
179                 icon: Icon(Icons.delete), //Que contienen los botones para cancelar o realizar la operación
180                 onPressed: () {
181                   _deleteCategoryDialog( context, _categoryList[index].id);
182                 },),],), // IconButton, Row, ListTile, Card, ListView.builder
183     floatingActionButton: FloatingActionButton( //El botón flotante servirá para mostrar el dialogo de registro nuevo
184       onPressed: () {_showFormInDialog(context);}, child: Icon(Icons.add,),), // FloatingActionButton, Scaffold
185     );
186   );
187 }
```

Generando la pantalla
de categorías

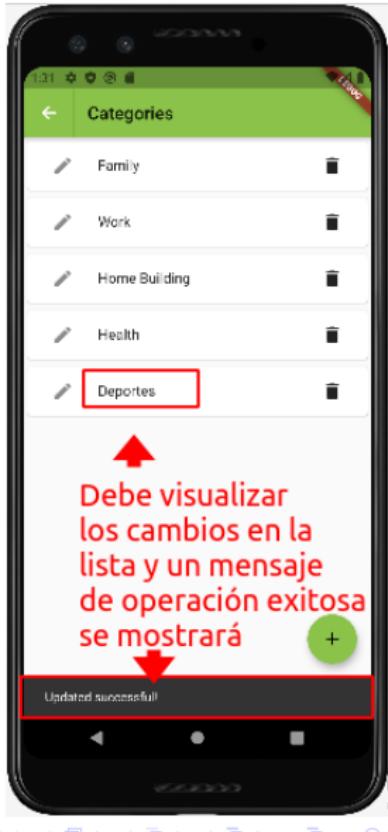
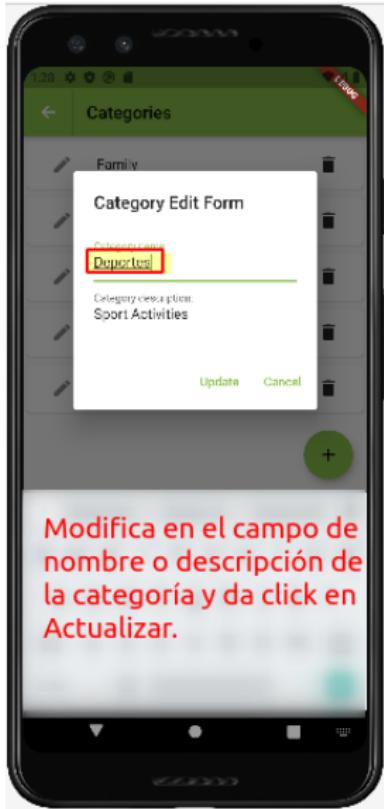
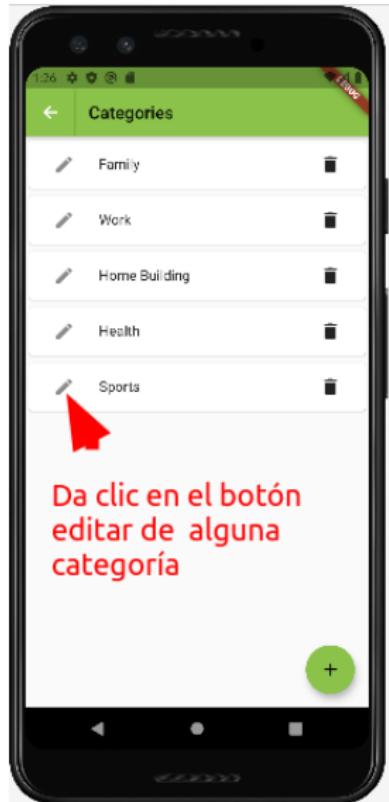
Ejecuta la aplicación y visualiza elementos de la pantalla de categorias



Prueba registrar nueva categoría



Prueba editar categoría



Prueba eliminar categoría

