

APLICACIONES MOVILES MULTIPLATAFORMA

LABORATORIO N° 13

Uso de modales – TextField – Plugin Shared Preferences



Alumno(s):	Rojas Huayhua Yesica Nancy		Nota	
Grupo:	A	Ciclo:V		

Laboratorio 13: Uso de modales – TextField – Shared Preferences

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Utilizar modales en un proyecto Flutter
- Interactuar con TextField
- Utilizar plugins en Flutter

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

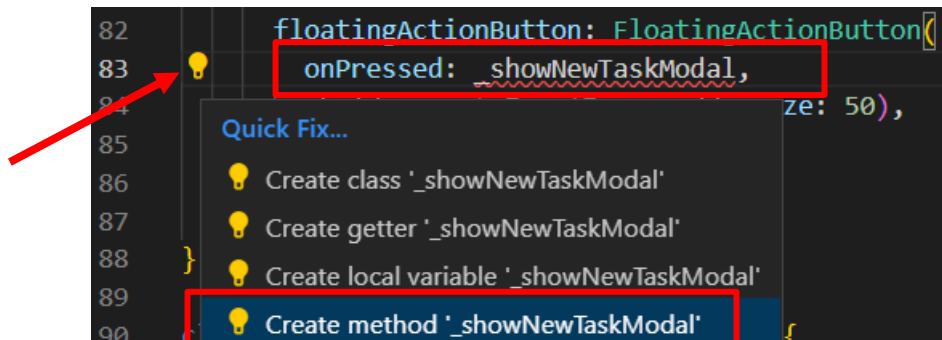
Equipos y Materiales:

- Una computadora con:
 - Windows 7 o superior
 - Conexión a la red del laboratorio
- Máquinas Virtuales
 - Windows 7 Pro 64bits Español – Plantilla

Procedimiento:

1. Agregando modales al proyecto:

- 1.1. En base al proyecto base del laboratorio anterior, vamos a modificar el contenido del archivo 'task_list_page.dart', para que muestre un modal al momento de presionar el botón principal:



- 1.2. Dentro del método generado indicamos lo siguiente:

```
89 void _showNewTaskModal() {
90   showModalBottomSheet(context: context, builder: builder)
91 }
```

- 1.3. Generamos una función anónima que internamente ejecuta la función que queremos ejecutar:

```
89 void _showNewTaskModal(BuildContext context) {
90   showModalBottomSheet(
91     context: context, builder: (_) => const _NewTaskModal();
92   )
93 }
```

```
95 class _NewTaskModal extends StatelessWidget {
96   const _NewTaskModal({super.key});
97
98   @override
99   Widget build(BuildContext context) {
100     return Container(
101       child: Column(
102         children: [
103           H1('Nueva tarea'),
104           TextField(),
105           ElevatedButton(
106             onPressed: () {},
107             child: Text('Guardar'),
108           ) // ElevatedButton
109         ],
110       ) // Column
111     ); // Container
112   }
113 }
```

- 1.4. Queremos que el modal se adapte al tamaño de los objetos que tiene dentro. Para ello, aplicamos lo siguiente:

```

89   void _showNewTaskModal(BuildContext context) {
90     showModalBottomSheet(
91       context: context,
92       isScrollControlled: true,
93       builder: (_) => const _NewTaskModal());
94   }
95 }

```

```

97   class _NewTaskModal extends StatelessWidget {
98     const _NewTaskModal({super.key});
99
100    @override
101    Widget build(BuildContext context) {
102      return Container(
103        child: Column(
104          mainAxisAlignment: MainAxisAlignment.min,

```

1.5. Generamos espacio en el aplicativo de acuerdo a lo siguiente:

```

100    @override
101    Widget build(BuildContext context) {
102      return Container(
103        padding: const EdgeInsets.symmetric(
104          horizontal: 33,
105          vertical: 23
106        ), // EdgeInsets.symmetric
107        child: Column(
108          crossAxisAlignment: CrossAxisAlignment.start,
109          mainAxisAlignment: MainAxisAlignment.min,
110          children: [
111            H1('Nueva tarea'),
112            const SizedBox(height: 26),
113            TextField(),
114            const SizedBox(height: 26),
115            ElevatedButton(
116              onPressed: () {},
117              child: Text('Guardar'),
118            ) // ElevatedButton
119          ],

```

1.6. Cambiamos el borde del contenedor a redondeado. Para tal, realizamos lo siguiente.

```

101    Widget build(BuildContext context) {
102      return Container(
103        padding: const EdgeInsets.symmetric(
104          horizontal: 33,
105          vertical: 23
106        ), // EdgeInsets.symmetric
107        decoration: const BoxDecoration(
108          borderRadius: BorderRadius.vertical(top: Radius.circular(21)),
109          color: Colors.white,
110        ), // BoxDecoration

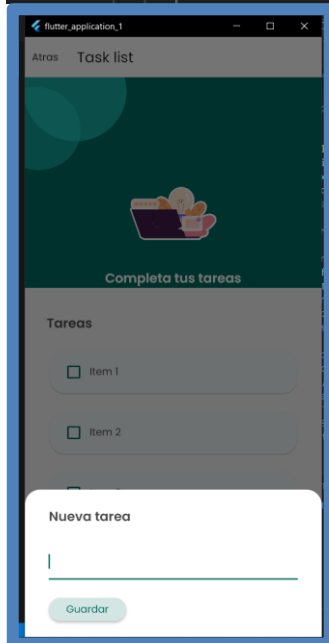
```

1.7. Nos dirigimos al archivo 'app.dart' y modificamos Theme de la siguiente manera:

```

20 textTheme: Theme.of(context).textTheme.apply(
21   fontFamily: 'Poppins',
22   bodyColor: textColor,
23   displayColor: textColor,
24 ),
25 bottomSheetTheme: const BottomSheetThemeData(
26   backgroundColor: Colors.transparent,
27 ), // BottomSheetThemeData
28 useMaterial3: true,

```

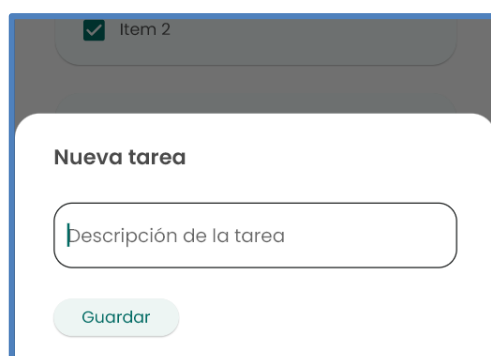


1.8. Modificamos también el TextField de la siguiente manera (en el archivo 'task_list_page.dart'):

```

115 H1('Nueva tarea'),
116 const SizedBox(height: 26),
117 TextField(
118   decoration: InputDecoration(
119     filled: true,
120     fillColor: Colors.white,
121     border: OutlineInputBorder(
122       borderRadius: BorderRadius.circular(16)
123     ), // OutlineInputBorder
124     hintText: 'Descripción de la tarea'
125   ), // InputDecoration
126 ), // TextField

```



- 1.9. Modificamos también el estilo del botón de la siguiente manera, de acuerdo al Theme, para que aplique a todos los botones. Nos ubicamos dentro del archivo 'app.dart' nuevamente y modificamos lo siguiente:

```

25   bottomSheetTheme: const BottomSheetThemeData(
26     backgroundColor: Colors.transparent,
27   ), // BottomSheetThemeData
28   elevatedButtonTheme: ElevatedButtonThemeData(
29     style: ElevatedButton.styleFrom(
30       minimumSize: const Size(
31         double.infinity,
32         54, // Size
33       ),
34       shape: RoundedRectangleBorder(
35         borderRadius: BorderRadius.circular(10)
36       ), // RoundedRectangleBorder
37       textStyle: Theme.of(context).textTheme.bodyMedium!.copyWith(
38         fontSize: 18,
39         fontWeight: FontWeight.w700,
40       )
41     ), // ElevatedButtonThemeData

```

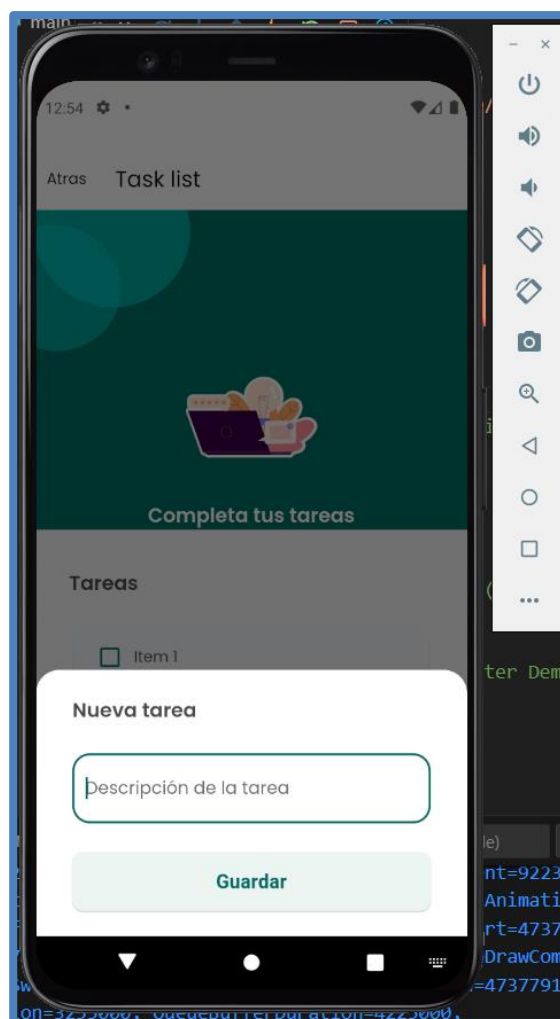
- 1.10. Guardamos los cambios, comprobamos y mostramos los resultados obtenidos a continuación.

El botón flotante (FloatingActionButton) se utiliza como un botón para mostrar un modal que permite agregar una nueva tarea a la lista.

Aquí, onPressed: _showNewTaskModal indica que cuando el botón sea presionado, se llamará a la función _showNewTaskModal.

Esta función usa showModalBottomSheet para mostrar un modal en la parte inferior de la pantalla cuando el botón es presionado. El modal que se muestra es la clase _NewTaskModal, que contiene un formulario para agregar una nueva tarea.

Dentro de _NewTaskModal, tienes un TextField para ingresar la descripción de la tarea y un ElevatedButton que, por el momento, tiene un onPressed: () {}, es decir, no tiene ninguna función asociada.



2. Interacción con TextField:

- 2.1. A continuación, queremos ingresar un mensaje y que se pueda registrar en el aplicativo. Para ello, realizamos las siguientes modificaciones en el archivo 'task_list_page.dart':

```

89   void _showNewTaskModal(BuildContext context) {
90     showModalBottomSheet(
91       context: context,
92       isScrollControlled: true,
93       builder: (_) => _NewTaskModal(),
94     );
95   }
96
97   class _NewTaskModal extends StatelessWidget {
98     _NewTaskModal({super.key});
99
100    final _controller = TextEditingController();
101
102    @override

```

```

118    const SizedBox(height: 26),
119    TextField(
120      controller: _controller,
121      decoration: InputDecoration(
122        filled: true,

```

```

131    ElevatedButton(
132      onPressed: () {
133        if (_controller.text.isNotEmpty) {
134          final task = Task(_controller.text);
135          Navigator.of(context).pop();
136        }
137      },
138      child: Text('Guardar'),
139    ) // ElevatedButton

```

- 2.2. Se puede hacer un callback para informar al listado que hay una nueva tarea. Para ello, modificamos lo siguiente

```

97   class _NewTaskModal extends StatelessWidget {
98     _NewTaskModal({super.key});
99
100    final controller = TextEditingController();
101    final void Function(Task task) onTaskCreated;

```

```

97   class _NewTaskModal extends StatelessWidget {
98     _NewTaskModal({super.key});
99
100    Quick Fix...
101    Add final field formal parameters

```

```

89   void _showNewTaskModal(BuildContext context) {
90     showModalBottomSheet(
91       context: context,
92       isScrollControlled: true,
93       builder: (_) => _NewTaskModal(onTaskCreated: (Task task) { },));
94   }
95 }
96
97 class _NewTaskModal extends StatelessWidget {
98   _NewTaskModal({super.key, required this.onTaskCreated});

```

- 2.3. Luego, antes de cerrar el widget del botón, se quiere llamar al callback generado, modificando lo siguiente:

```

133   const SizedBox(height: 26),
134   ElevatedButton(
135     onPressed: () {
136       if (_controller.text.isNotEmpty) {
137         final task = Task(_controller.text);
138         onTaskCreated(task);
139         Navigator.of(context).pop();
140       }
141     },

```

- 2.4. A continuación, hacemos la gestión de estado para que se añada la nueva tarea en la lista de tareas, de acuerdo a lo siguiente:

```

89   void _showNewTaskModal(BuildContext context) {
90     showModalBottomSheet(
91       context: context,
92       isScrollControlled: true,
93       builder: (_) => _NewTaskModal(onTaskCreated: (Task task) {
94         setState(() {
95           taskList.add(task);
96         });
97       },));

```

- 2.5. Guardamos los cambios, ejecutamos el proyecto, comprobamos y adjuntamos los resultados obtenidos. Realizamos comentarios diversos sobre el funcionamiento del aplicativo hasta el momento.
- 2.6. Eliminamos la lista de tareas que se genera mediante código, para que pueda ingresarse mediante el modal generado. Adjuntamos resultados obtenidos en la ejecución del aplicativo.

creación de nuevas tareas a partir de la entrada de texto proporcionada por el usuario en el modal de creación de tareas (_NewTaskModal).

final _controller = TextEditingController();

Este controlador se utiliza para capturar y controlar el texto ingresado en el TextField. Se asigna al TextField de la siguiente manera:

```

TextField(
  controller: _controller,
  // ...otros atributos de decoración y estilo
)

```

Cuando el usuario ingresa una descripción de tarea en el TextField y presiona el botón "Guardar", se ejecuta el siguiente código:

```

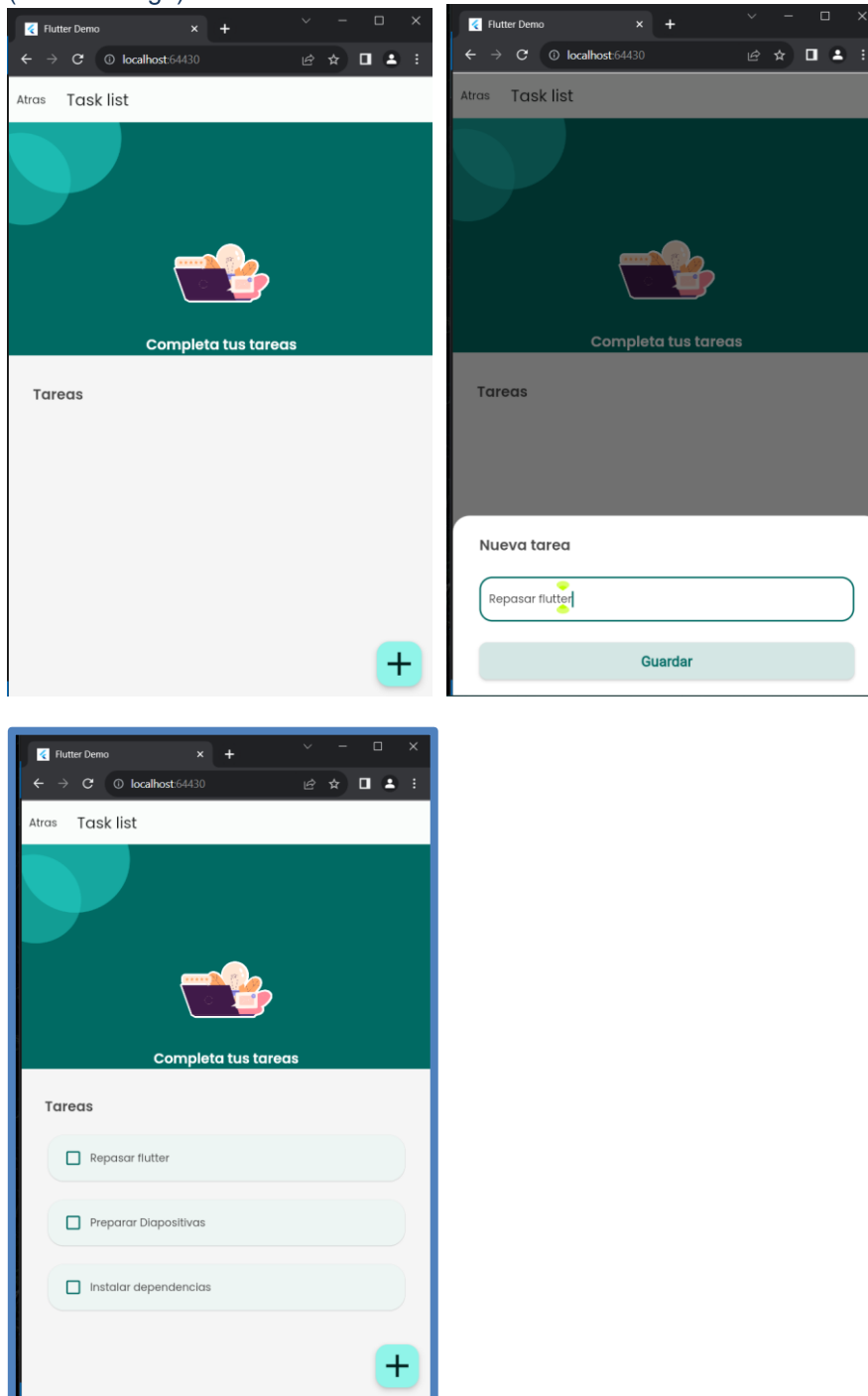
ElevatedButton(
  onPressed: () {
    if (_controller.text.isNotEmpty) {
      final task = Task(_controller.text);

```



```
onTaskCreated(task);  
Navigator.of(context).pop();  
}  
},  
child: Text('Guardar'),  
)
```

Así, la interacción con el TextField permite al usuario ingresar una descripción de la tarea, la cual se utiliza para crear una nueva instancia de Task cuando se presiona el botón "Guardar". Esta tarea se agrega a la lista de tareas existentes en la página principal (TaskListPage).



3. Uso de plugin Shared Preferences:

- 3.1. Ingresamos a la página <https://pub.dev/>, para tener alcance de los paquetes y plugins de Flutter.
- 3.2. Para empezar, un package es una librería externa que nos provee código Dart (p. ej. un conjunto de widgets), la cual podemos incluir en nuestro proyecto y utilizar ello. Un plugin es una librería que nos permite acceder a la parte nativa de la plataforma a la cual vamos a desarrollar (p. ej. un SDK de mapas de Android, iOS o Mac). En nuestro caso vamos a utilizar un plugin llamado Shared Preferences.
- 3.3. Buscamos este plugin mediante la página compartida. Ingresamos a la documentación del siguiente plugin:

shared_preferences 7569 LIKES 140 PUB POINTS 100% POPULARITY


Flutter plugin for reading and writing simple key-value pairs. Wraps NSUserDefaults on iOS and SharedPreferences on Android.

v 2.2.0 (4 days ago) flutter.dev BSD-3-Clause Flutter Favorite Dart 3 compatible

SDK | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

API results: ▶ [shared_preferences/shared_preferences-library.html](#)

- 3.4. Hacemos clic sobre su nombre para tener alcance de su documentación. A continuación, alcancemos los puntos principales de la documentación.
- 3.5. De acuerdo a la documentación, nos ubicamos a la altura del título hacemos clic en el icono del costado derecho, para copiar el nombre del plugin:

shared_preferences 2.2.0  Copy "shared_preferences: ^2.2.0" to clipboard

Published 4 days ago • flutter.dev Dart 3 compatible

- 3.6. Luego, regresamos a nuestro proyecto en VS Code y abrimos el archivo 'pubsec.yaml'. Modificamos según lo siguiente:

```

30 dependencies:
31   flutter:
32     sdk: flutter
33   shared_preferences: ^2.2.0
34

```

Nota: Colocar el nombre del plugin al mismo nivel de "flutter".

- 3.7. Guardar los cambios realizados. Debe empezar una instalación mediante 'flutter pub get'. Esperar a que culmine el proceso para continuar.
- 3.8. Luego, nos dirigimos al archivo 'task_list_page.dart' y modificamos según lo siguiente:

```


14 class _TaskListPageState extends State<TaskListPage> {
15
16   final taskList = <Task>[];
17
18   @override
19   Future<void> initState() async {
20     final prefs = await SharedPreferences.getInstance();
21     super.initState();
22   }

```

- 3.9. Vamos a continuar guardando nuestro listado de tareas mediante el plugin instalado. Para ello, seguimos los siguientes pasos:

3.9.1. Notemos que ya se ha añadido el plugin al comienzo del archivo:

```
4 import 'package:flutter/material.dart';
5 import 'package:shared_preferences/shared_preferences.dart';
6
```



3.9.2. Modificamos lo siguiente para guardar el listado en el plugin. El plugin tiene diversos métodos Set (SetString, SetStringList, SetBool, etc.) Para ello, no se puede guardar un objeto Task directamente ya que debemos **serializar** primero ese objeto. Esto significa que vamos a convertir ese objeto a un array de bytes. En nuestro caso vamos a convertir el objeto Task a una cadena de texto. La forma más práctica de hacer ello es pasar el objeto a una estructura JSON, para luego transformar el mapa que se indique para que se obtenga una cadena de texto. Para ello, aplicamos lo siguiente:

```
94 void _showNewTaskModal(BuildContext context) {
95   showModalBottomSheet(
96     context: context,
97     isScrollControlled: true,
98     builder: ( ) => NewTaskModal(onTaskCreated: (Task task) {
99       setState(() async {
100         taskList.add(task);
101         final prefs = await SharedPreferences.getInstance();
102         final map = {
103           'title': task.title,
104           'done': task.done
105         };
106         prefs.setStringList('tasks', [
107           jsonEncode(map),
108         ]);
109       });
110     });
111 }
```

3.9.3. Vamos a crear dos métodos dentro de la clase 'Task', para convertir el objeto Task a una estructura de tipo mapa, que es lo requerido para la serialización mediante JSON.

```
lib > app > model > task.dart > ...
1 class Task {
2
3   Task(this.title, {this.done = false});
4
5   final String title;
6   bool done;
7
8   Map<String, dynamic> toJson() {
9     return {
10       'title': title,
11       'done': done
12     };
13   }
14
15 }
```

3.9.4. Modificamos el contenido del archivo 'task_list_page.dart':

```

94  void _showNewTaskModal(BuildContext context) {
95      showModalBottomSheet(
96          context: context,
97          isScrollControlled: true,
98          builder: (_) => _NewTaskModal(onTaskCreated: (Task task) {
99              setState(() async {
100                  taskList.add(task);
101                  final prefs = await SharedPreferences.getInstance();
102                  prefs.setStringList('tasks', [
103                      jsonEncode(task.toJson()),
104                  ]);
105              });
106          });
107      }

```

3.9.5. Pero no queremos que se realice este proceso por cada objeto, sino de acuerdo a la lista de tareas. Siendo así, utilizamos la función 'map', mediante la cual se convierte un listado de objetos de Task a un objeto de cadenas de texto, el cual se va a serializar. Para ello, aplicamos lo siguiente:

```

98      builder: (_) => _NewTaskModal(onTaskCreated: (Task task) {
99          setState(() async {
100              taskList.add(task);
101              final prefs = await SharedPreferences.getInstance();
102              prefs.setStringList(
103                  'tasks', taskList.map((e) => jsonEncode(e.toJson())).toList());
104          });
105      });

```

3.9.6. Para la deserialización se debe generar un constructor en la clase 'Task', como se define a continuación:

```

lib > app > model > task.dart > Task
1  class Task {
2
3      Task(this.title, {this.done = false});
4
5      Task.fromJson(Map<String, dynamic> json){
6          title = json['title'];
7          done = json['done'];
8      }
9      late final String title;
10     late bool done;
11

```

3.9.7. Finalmente, modificamos el archivo 'task_list_page.dart'

```

102     prefs.setStringList(
103         'tasks', taskList.map((e) => jsonEncode(e.toJson())).toList());
104
105     final taskStrings = prefs.getStringList('tasks');
106     final newTaskList = taskStrings?.map((e) => Task(jsonDecode(e))).toList();
107
108     });

```

3.9.8. Guardamos los cambios, comprobamos y adjuntamos los resultados obtenidos. Realizamos comentarios sobre el funcionamiento del aplicativo.

Estamos utilizando SharedPreferences para almacenar y recuperar una lista de tareas (Task)

En la función _showNewTaskModal, cuando se crea una nueva tarea y se presiona el botón "Guardar" en el modal de nueva tarea.

```
void _showNewTaskModal(BuildContext context) {
  showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    builder: (_) => _NewTaskModal(
      onTaskCreated: (Task task) {
        setState(() async {
          taskList.add(task);
          final prefs = await SharedPreferences.getInstance();
          prefs.setStringList('tasks',
            taskList.map((e) => jsonEncode(e.toJson())).toList());

          final taskStrings = prefs.getStringList('tasks');
          final newTaskList =
            taskStrings?.map((e) => Task(jsonDecode(e))).toList();
        });
      },
    ), // _NewTaskModal
);
}
```

`prefs.setStringList('tasks', taskList.map((e) => jsonEncode(e.toJson())).toList());`; guarda la lista de tareas en `SharedPreferences`. Se convierte cada tarea a formato JSON usando `jsonEncode` antes de guardarla como una lista de cadenas (`StringList`).

`final taskStrings = prefs.getStringList('tasks');`; recupera la lista de tareas guardadas como una lista de cadenas (`StringList`).

`final newTaskList = taskStrings?.map((e) => Task(jsonDecode(e))).toList();` intenta decodificar cada cadena de tarea JSON a objetos `Task` y crea una nueva lista de tareas. Sin embargo, esta lista no está siendo utilizada para actualizar el estado de la página.

Tarea:

- De acuerdo a la página: <https://pub.dev/> (o las referencias que se vea por conveniente), generar un nuevo proyecto Flutter que considere el uso de un plugin, diferente al visto en el ejemplo anterior. Adjuntar evidencias de desarrollo y resultados obtenidos.

utilizando el plugin `url_launcher`

```
dependencies:
  flutter:
    sdk: flutter
  shared_preferences: ^2.2.0
  url_launcher: ^6.0.9
```

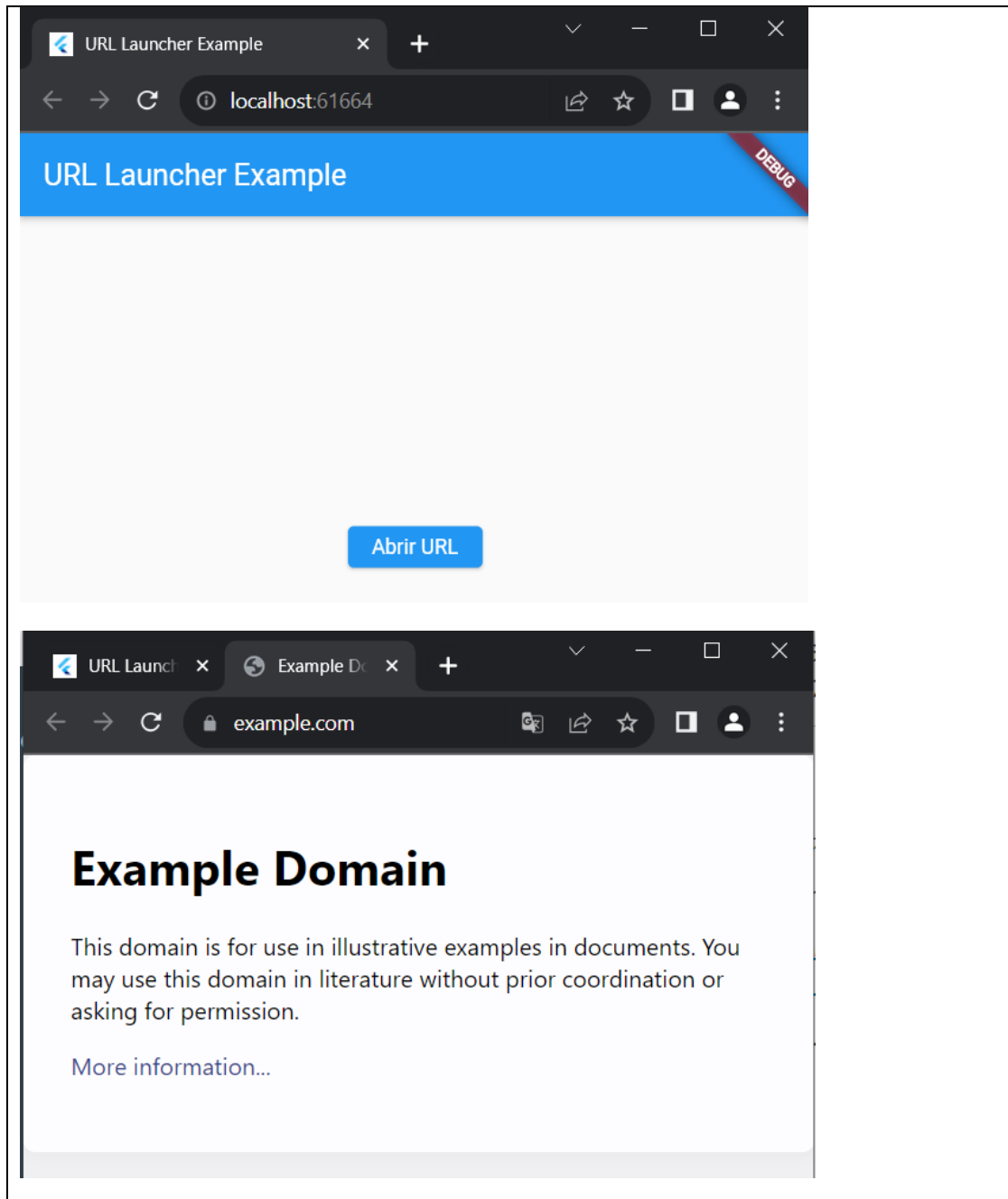
```
import 'package:flutter/material.dart';
import 'package:url_launcher/url_launcher.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
```

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    title: 'URL Launcher Example',  
    theme: ThemeData(  
      primarySwatch: Colors.blue,  
    ),  
    home: MyHomePage(),  
  );  
}  
}  
  
class MyHomePage extends StatelessWidget {  
  // URL que se abrirá al presionar el botón  
  final String _url = 'https://example.com';  
  
  // Función para abrir la URL  
  Future<void> _launchURL() async {  
    if (await canLaunch(_url)) {  
      await launch(_url);  
    } else {  
      throw 'Could not launch $_url';  
    }  
  }  
}  
  
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      title: Text('URL Launcher Example'),  
    ),  
    body: Center(  
      child: ElevatedButton(  
        onPressed: _launchURL, // Llama a la función al presionar el  
        botón  
        child: Text('Abrir URL'),  
      ),  
    ),  
  );  
}
```

Este ejemplo crea una aplicación con un botón que, al ser presionado, abrirá la URL proporcionada en el navegador predeterminado del dispositivo.



Conclusiones:

Los modales en Flutter se utilizan para mostrar contenido sobre la interfaz principal de la aplicación. Pueden ser modales a pantalla completa o modales emergentes que se superponen sobre la pantalla actual. Estos modales son útiles para solicitar información adicional al usuario, confirmar acciones o presentar contenido importante sin abandonar la vista actual.

La interacción entre un TextField y un modal generalmente implica la apertura del modal para capturar información adicional o realizar acciones específicas. El TextField puede ser el desencadenante para abrir el modal, como en el caso de solicitar información adicional al usuario. Una vez que se completa la interacción en el modal, los datos pueden actualizarse y reflejarse en el TextField o en otros componentes de la interfaz.

El plugin SharedPreferences en Flutter se utiliza para almacenar datos de forma persistente en el dispositivo del usuario. Permite guardar y recuperar datos simples como enteros, cadenas, booleanos, listas y mapas de forma asíncrona.

Es esencial gestionar correctamente el estado al interactuar con modales y campos de entrada como TextField. También se deben tener en cuenta las actualizaciones en la interfaz principal después de que el usuario interactúe con el modal. La comunicación entre el modal y los widgets principales se realiza a través de funciones de retorno de llamada o métodos que permiten la actualización del estado y la presentación de datos actualizados en la interfaz.