Desarrollo de App de comercio electrónico

Dr. Casimiro Gómez González
Departamento de I+D, SMARTEST
correo: casimiro.gomez@smartest.mx
Tel: 222 707 4118

Otoño 2021

Prólogo

El comercio electrónico es una parte importante en el desarrollo empresarial de SMARTEST. Po ello se describe el desarrollo de dicho proyecto para tener las bases de diferentes tipos de aplicaciones a desarrollar en donde se tomará como punto de partida lo presentado en este proyecto.

El autor
Casimiro Gómez González
Departamento de I+D, SMARTEST

Índice general

Capítulo 1

Introducción

En el presente capítulo se realizarán las configuraciones iniciales de un proyecto en flutter.

1.1. Creación del proyecto

Vamos a crear el app de fluter para comercio electrónico de smartest, el proyecto se llama **vecnu** [1]. Para ello en una terminal, con una máquina que previamente tiene flutter instalado, ejecutamos:

```
1 $ mkdir Vecnu
2 $ cd Vecnu
3 $ flutter create vecnu
```

Para editar el proyecto se utiliza ***visual studio code***, por lo cual ejecutamos:

```
1 $ cd vecnu
2 $ code .
```

Anteriormente se instalá a ***visual studio code*** la extensión **flutter dev tools**. Apretamos las teclas **ctr->shift->p** y escribimos **Flutter: Launch emulator** y nos permite seleccionar los emuladores que instalamos desde **android studio**, en nuestro caso el de pixel, quedando como se muestra en la figura 1.1.

Para empezar a trabajar con el proyecto activamos la opción de debug, para ellos nuevamente apretamos las teclas **ctr->shift->p** y escribimos **Debug: Start Debugging**, lo cual nos permite ejecutar nuestro proyecto en el emulador previamente abierto.

[1] Lo cual en el idioma lojban significa vender

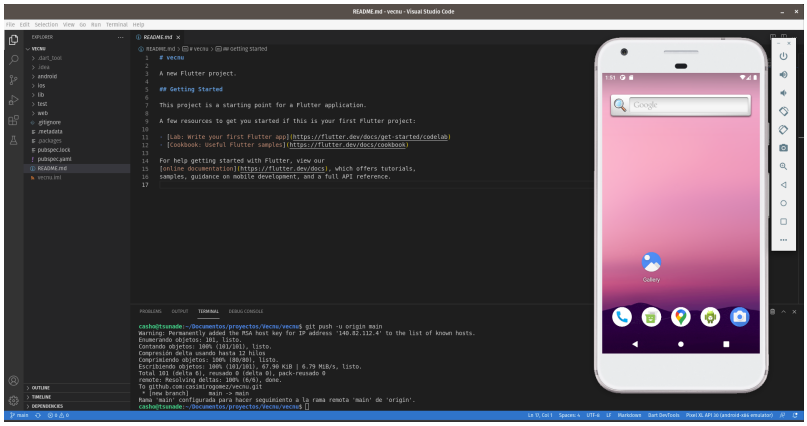


Figura 1.1: Programa creado y editando en ***visual studio code*** y mostrando el emulador

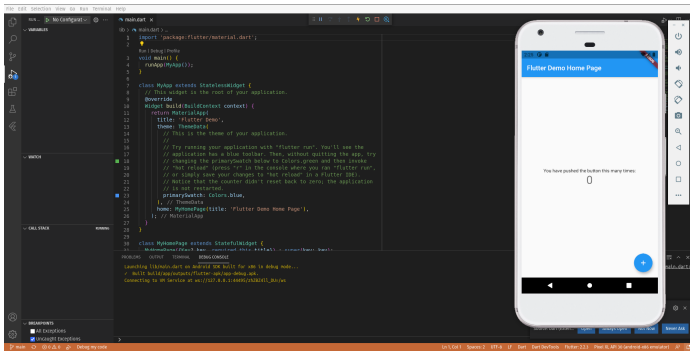


Figura 1.2: Programa ejecutandose en el emulador

Para que el debugger se ejecute correctamente necesitamos tener instalado Java en nuestra distribución de linux.

1.2. Creando los temas para nuestro proyecto

Para probar como se verán los colores en nuestra aplicación se usa la herramienta **Material Design Color Tool** la cual se encuentra en la dirección *https://material.io /resources /color/* en donde se seleccionan los colores primarios y secundarios para nuestra aplicación. En el caso de SMARTEST en la figura 1.3 se muestra el código de colores.

De la figura 1.3 se toman dos colores el cyan y el gris, y se forman los colores de

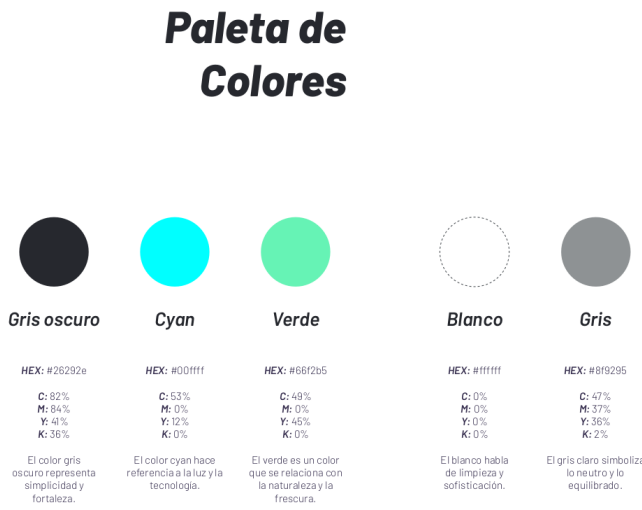


Figura 1.3: Paleta de colores

la aplicación. Una vez decididos los colores de la aplicación y probados en la página. Se procede a programar el app.

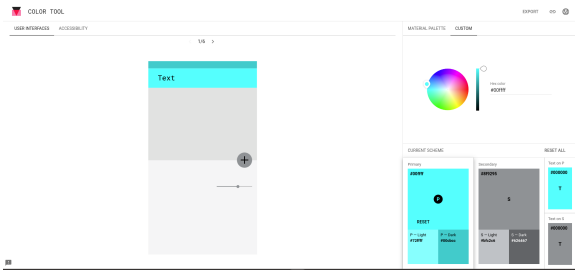


Figura 1.4: Colores primario y secundario de la aplicación

Para ello editamos el archivo **lib/main.dart** quedando de la siguiente forma

```
...
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
          brightness: Brightness.dark,
          primaryColor: Colors.cyan[400],
          accentColor: Colors.grey[200],
          textTheme: TextTheme(
```

```
11              headline5: TextStyle(fontSize: 72.0, fontWeight:
                    FontWeight.bold),
12              headline6: TextStyle(fontSize: 36.0, fontStyle:
                    FontStyle.italic),
13              bodyText1: TextStyle(fontSize: 18.0))),
14      home: MyHomePage(title: 'Flutter Demo Home Page'),
15    );
16  }
17 ...
```

Para que el estilo de textos definidos se apliquen a los textos del app necesitamos modificarlos de la siguiente forma:

```
1  ...
2            children: <Widget>[
3              Text(
4                'You have pushed the button this many times:',
5                style: Theme.of(context).textTheme.bodyText1,
6              ),
7              Text(
8                '$_counter',
9                style: Theme.of(context).textTheme.headline4,
10             ),
11           ],
12 ...
```

1.3. Construyendo la página de registro

Para construir la página de registro, editamos primero el archivo **lib /main.dart** y le pondremos el nombre de nuestro e-commerce.

```
1  ...
2    @override
3    Widget build(BuildContext context) {
4      return MaterialApp(
5        title: 'Vecnu SMARTEST e-commerce',
6        theme: ThemeData(
7            brightness: Brightness.dark,
8            primaryColor: Colors.cyan[400],
9            accentColor: Colors.grey[200],
10           textTheme: TextTheme(
11               headline5: TextStyle(fontSize: 72.0, fontWeight:
                     FontWeight.bold),
12               headline6: TextStyle(fontSize: 36.0, fontStyle:
                     FontStyle.italic),
```

```
              bodyText1: TextStyle(fontSize: 18.0))),
      //home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
  ...
```

En el pedazo de programa de **lib /main.dart** mostrado anteriormente se realizaron dos modificaciones la primera corresponde al nombre de la aplicación la cual se llamará **Vecnu SMARTEST e-commerce** y la segunda es que la linea **home:** se convierte en comentario. La razón para hacer esto es porque vamos a borrar la clase **MyHomePage** del **lib /main.dart** y crearemos un nuevo folder en el directorio **lib** el cual llamaremos **paginas** y dentro de este directorio crearemos un programa llamado **lib /paginas /pagina_registro.dart**

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

class PaginaRegistro extends StatefulWidget {
  @override
  PaginaRegistroEstado createState() => PaginaRegistroEstado();
}

class PaginaRegistroEstado extends State<PaginaRegistro> {
  @override
  Widget build(BuildContext context) {
    return Text("Hola␣Mundo");
  }
}
```

Para poder ejecutar el programa creado activamos home: y agragamos la clase **PaginaRegistro()** anexando asimismo la libreria al programa, quedando de la siguiente manera el programa **lib /main.dart**.

```
import 'package:flutter/material.dart';
import 'package:vecnu/paginas/pagina_registro.dart';

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Vecnu␣SMARTEST␣e-commerce',
      theme: ThemeData(
```

```
15            brightness: Brightness.dark,
16            primaryColor: Colors.cyan[400],
17            accentColor: Colors.grey[200],
18            textTheme: TextTheme(
19                headline5: TextStyle(fontSize: 72.0, fontWeight:
                      FontWeight.bold),
20                headline6: TextStyle(fontSize: 36.0, fontStyle:
                      FontStyle.italic),
21                bodyText1: TextStyle(fontSize: 18.0))),
22        home: PaginaRegistro(),
23      );
24    }
25  }
```

Lo cual nos produce una salida que se muestra en la figura 1.5.

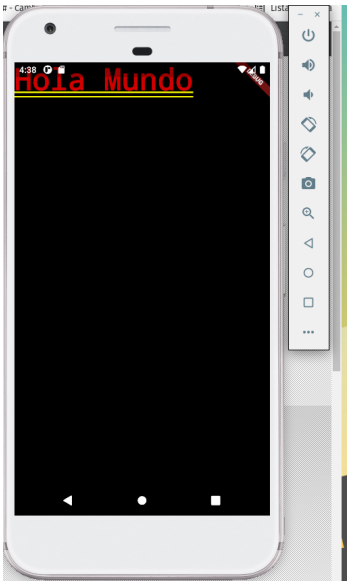


Figura 1.5: Salida del programa Hola Mundo

1.3.1. Creando el Scaffold de la página de registro

La clase de la página de registro (**lib /paginas /pagina_registro.dart**) la editamos y sustituimos el hola mundo por un Scaffold que tiene la estructura que deseamos.

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

```
2
3  class PaginaRegistro extends StatefulWidget {
4    @override
5    PaginaRegistroEstado createState() => PaginaRegistroEstado();
6  }
7
8  class PaginaRegistroEstado extends State<PaginaRegistro> {
9    Widget _mostrarTitulo() {
10     return Text(
11       'Registro',
12       style: Theme.of(context).textTheme.headline5,
13     );
14   }
15
16   Widget _mostrarEntradaUsuario() {
17     return Padding(
18         padding: EdgeInsets.only(top: 20.0),
19         child: TextFormField(
20             decoration: InputDecoration(
21                 border: OutlineInputBorder(),
22                 labelText: 'Usuario',
23                 hintText: 'Introduce␣usuario,␣tamano␣min␣6',
24                 icon: Icon(Icons.face, color: Colors.grey))));
25   }
26
27   Widget _mostrarEntradaCorreo() {
28     return Padding(
29         padding: EdgeInsets.only(top: 20.0),
30         child: TextFormField(
31             decoration: InputDecoration(
32                 border: OutlineInputBorder(),
33                 labelText: 'Correo',
34                 hintText: 'Introduce␣un␣correo␣valido',
35                 icon: Icon(Icons.mail, color: Colors.grey))));
36   }
37
38   Widget _mostrarEntradaClave() {
39     return Padding(
40         padding: EdgeInsets.only(top: 20.0),
41         child: TextFormField(
42             obscureText: true,
43             decoration: InputDecoration(
44                 border: OutlineInputBorder(),
45                 labelText: 'Clave',
46                 hintText: 'Introduce␣clave,␣tamano␣min␣6',
47                 icon: Icon(Icons.lock, color: Colors.grey))));
```

```
48   }
49
50 Widget _mostrarAccionesForma() {
51     final ButtonStyle raisedButtonStyle = ElevatedButton.styleFrom(
52       onPrimary: Colors.black87,
53       primary: Theme.of(context).primaryColor,
54       minimumSize: Size(88, 36),
55       padding: EdgeInsets.symmetric(horizontal: 16),
56       shape: const RoundedRectangleBorder(
57         borderRadius: BorderRadius.all(Radius.circular(10)),
58       ),
59     );
60     return Padding(
61       padding: EdgeInsets.only(top: 20.0),
62       child: Column(
63         children: [
64           ElevatedButton(
65             style: raisedButtonStyle,
66             child: Text('Enviar',
67                 style: Theme.of(context)
68                     .textTheme
69                     .bodyText1!
70                     .copyWith(color: Colors.black)),
71             onPressed: () => print('Enviado'),
72           ),
73           TextButton(
74             child: Text('Ya␣te␣registraste?␣Entrar'),
75             onPressed: () => print('Entrar'),
76           )
77         ],
78       ),
79     );
80   }
81
82
83   @override
84   Widget build(BuildContext context) {
85     return Scaffold(
86         appBar: AppBar(title: Text('Registro')),
87         body: Container(
88           padding: EdgeInsets.symmetric(horizontal: 20.0),
89           child: Center(
90             child: SingleChildScrollView(
91               child: Form(
92                 child: Column(
93                   children: [
```

```
94                      _mostrarTitulo(),
95                      _mostrarEntradaUsuario(),
96                      _mostrarEntradaCorreo(),
97                      _mostrarEntradaClave(),
98                      _mostrarAccionesForma()
99                    ],
100                 ),
101               ),
102             ),
103           ),
104         ));
105   }
106 }
```

En programa anterior de **lib /paginas /pagina_registro.dart** se crean 5 funciones: _mostrarTitulo(), _mostrarEntradaUsuario(),_mostrarEntradaCorreo(), _mostrarEntradaClave(), _mostrarAccionesForma(). Todas las funciones regresan un Widget, que crea una columna de Widgets. Cada uno formando el titulo, la entrada de usuario, la entrada de correo, la clave y las formas del boton de enviar y de entrar.

Tambien cuando se pulsa sobre el botón **enviar** y sobre el botón **¿Ya te registraste? Entrar** se imprime en la terminal los letreros enviar y entrar

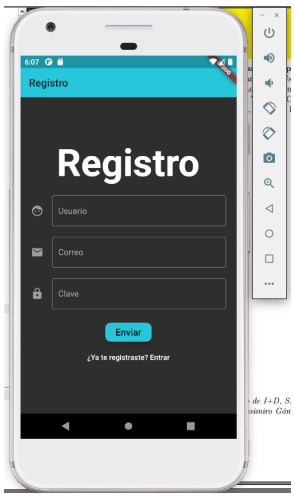
1.3.2. Agregando validación de forma, creando estado de forma

Ahora agregaremos validación a la forma. Primero agregamos validación a la forma de usuario.

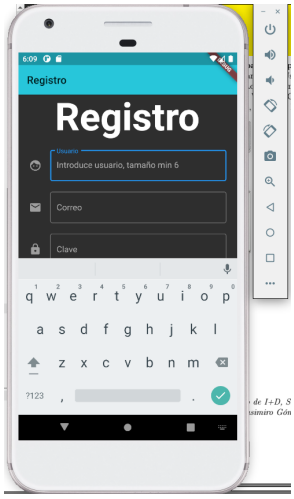
```
1 ...
2   Widget _mostrarEntradaUsuario() {
3     return Padding(
4         padding: EdgeInsets.only(top: 20.0),
5         child: TextFormField(
6             validator: (val) => val!.length < 6 ? 'Nombre␣usuario␣
                  muy␣corto': null,
7             decoration: InputDecoration(
8 ...
```

Posteriormente agregamos validación a la forma de correo

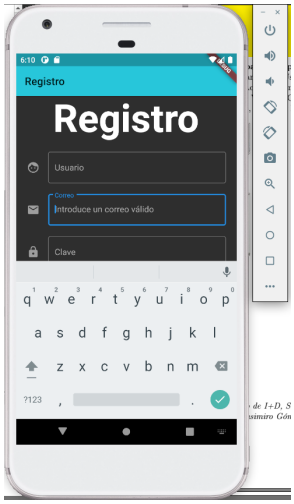
```
1 ...
2   Widget _mostrarEntradaCorreo() {
3     return Padding(
4         padding: EdgeInsets.only(top: 20.0),
```



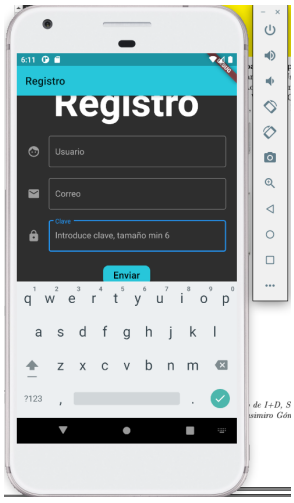
(a) Pantalla de registro



(b) click en usuario



(c) click en correo



(d) click en clave

Figura 1.6: Imagenes del Funcionamiento de la página de registro

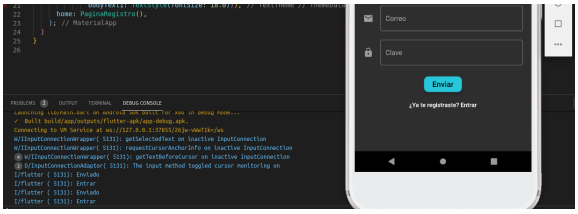


Figura 1.7: Salida al pulsar el botón **enviar** o **¿ya te registraste? enviar**

```
5           child: TextFormField(
6               validator: (val) => !val!.contains('@') ? 'Correo␣
                    invalido':null,
7               decoration: InputDecoration(
8 ...
```

Ahora agregamos validación a la forma de clave

```
1 ...
2  Widget _mostrarEntradaClave() {
3     return Padding(
4           padding: EdgeInsets.only(top: 20.0),
5           child: TextFormField(
6               validator: (val) => val!.length < 6 ? 'Nombre␣usuario␣
                    muy␣corto': null,
7               obscureText: true,
8 ...
```

Capítulo 2

Introducción

Primero creamos el proyecto al cual le agregaremos la funcionalidad de excel.

```
$ flutter create leer_excel
```

Posteriormente debemos agregar el paquete de Excel a su archivo **pubspec.yaml**. Haga clic en el .[ex]cel"de arriba para obtener la última versión del paquete de Excel. Actualmente estoy usando:

```
dependencies:
        excel: ^1.1.5
```

Ahora necesitamos agregar la importación de la librería, para ello creamos el archivo ***leer_ excel.dart*** en donde escribiremos el programa de importación de excel.

```
import  'package:excel/excel.dart';
```

Se modifica el programa principal

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

void main() {
        runApp(const MiApp());
}

class MiApp extends StatelessWidget {
        const MiApp({Key? key}) : super(key: key);

        // This widget is the root of your application.
        @override
        Widget build(BuildContext context) {
                return MaterialApp(
                title: 'Lectura de Excel',
```

```
16                  theme: ThemeData(
17                  // This is the theme of your application.
18                  //
19                  // Try running your application with "flutter run". You'll see the
20                  // application has a blue toolbar. Then, without quitting the app, try
21                  // changing the primarySwatch below to Colors.green and then invoke
22                  // "hot reload" (press "r" in the console where you ran "flutter run",
23                  // or simply save your changes to "hot reload" in a Flutter IDE).
24                  // Notice that the counter didn't reset back to zero; the application
25                  // is not restarted.
26                  primarySwatch: Colors.blue,
27                  ),
28                  home: LeerExcel(),
29                  );
30          }
31  }
```

El archivo de leer_excel.dart queda de la siguiente forma:

```
1  import 'package:excel/excel.dart';
2  import 'package:flutter/material.dart';
3  import 'dart:io';
4  import 'package:path/path.dart';
5  import 'package:excel/excel.dart';
6
7  class LeerExcel extends StatelessWidget {
8          const LeerExcel({ Key? key }) : super(key: key);
9
10         @override
11         Widget build(BuildContext context) {
12                 return Container(color: const Color(0xFF2DBD3A));
13         }
14 }
```

Cuando se intenta compilar el programa marca el siguiente error:

```
1  Error: Cannot run with sound null safety, because the following dependencies
2  don't␣support␣null␣safety
```

2.0.1. Corrección en Android Studio

Para corregir el error anterior agregamos la opción Run → Edit Configurations → Add Additional Run args → –**no-sound-null-safety**. Como se indica en la figura 2.1.

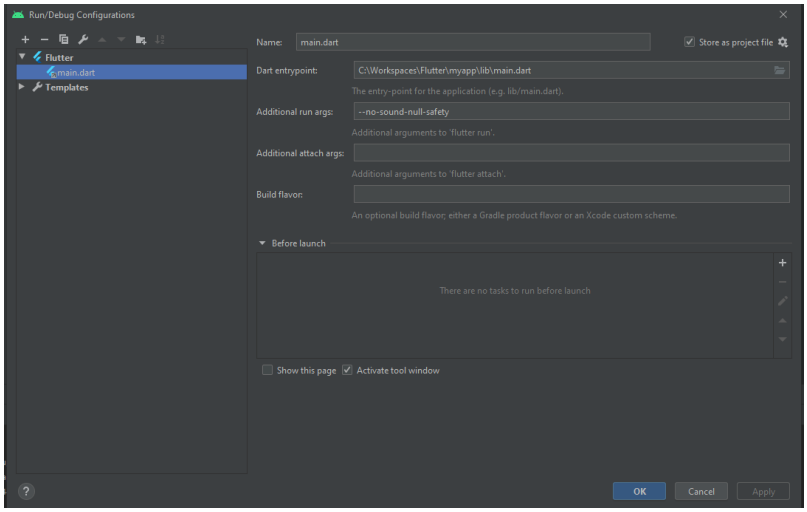


Figura 2.1: Corrección del error de compilación

Después de ejecutar la compilación la salida obtenida se muestra en la figura 2.2.

Figura 2.2: Pantalla de salida de la primera ejecución