Práctica 3

Pruebas de desarrolladores

3.1. Programación y objetivos

Esta práctica constará de una sola sesión. Tendrá una puntuación en la nota final de prácticas de 1 punto sobre 10.

3.1.1. Objetivos generales de la práctica 3

1. Aprender a diseñar las pruebas para maximizar el número de errores detectados.
2. Aprender a hacer pruebas básicas de unidad en Dart/Flutter y de componentes en Flutter.
3. Entender el funcionamiento de las colecciones o grupos de pruebas (suites, groupes).
4. Investigar cómo hacer pruebas de integración en Flutter.

3.1.2. Planificación y tareas específicas

Se espera de cada estudiante que en esta práctica:

1. Diseñe las pruebas de unidad y de componentes a realizar.
2. Aprenda cómo hacer pruebas de unidad en Dart/Flutter y de componentes en Flutter.
3. Codifique las pruebas de unidad y componentes y las pase.

Sesión	Semana	Tareas
S1	23-28 abril	Diseño, implementación y ejecución de pruebas de unidad, de widgets y (opcionalmente) de integración

3.2. Criterios de evaluación

Se tendrán en cuenta los siguientes criterios:

- Diseño de las pruebas de unidad y de componentes (condiciones y casos de prueba) y agrupación de las mismas (groups): 20 %
- Pruebas de unidad: Codificar y probar un grupo con al menos 6 pruebas (test), correspondientes a 2 métodos de 3 clases distintas: 20 %
- Éxito en las pruebas de unidad [1]: 20 %
- Pruebas de componentes (widgets): Codificar y probar un grupo con 3 pruebas (testWidgets), correspondientes a widgets de al menos dos páginas distintas: 20 %
- Éxito en las pruebas de componentes (widgets) [2]: 20 %
- Pruebas de integración (opcional): 20 %

3.3. Material a entregar

Se subirá un pdf con el diseño de las pruebas (descripción de las condiciones de prueba y los casos de prueba) y un informe del resultado de la ejecución de las mismas (incluir capturas de pantalla). Para las pruebas de unidad y de widgets, se recomienda que se capture la pantalla del navegador con los resultados de las pruebas exportados a html (opción .ExPort Test Results-y seleccionar 'abrirlo en el browser'– a la izquierda del total de test superados y fallidos). Para las pruebas de integración (opcionales), puede hacerse una captura de la ventana *Run*, *terminal* o de cualquier otro lugar donde se hayan probado.

3.4. Plazos de entrega y presentación de la práctica

Esta práctica será subida a PRADO en una tarea que terminará justo antes del inicio de la práctica 4 (a las 15:30 horas del día de la primera sesión de la misma). La práctica completa

[1] Se deben encontrar al menos 4 fallos en el código subido en su día.
[2] Se debe encontrar al menos 1 fallo en el código subido en su día.

será evaluada mediante una entrevista con el profesor de prácticas.

3.5. Descripción general de los requisitos de la práctica

Esta práctica está destinada a probar el código de la práctica 2, tanto mediante pruebas de unidad como mediante pruebas de widgets o componentes. De forma opcional, también pueden realizarse pruebas de integración.

Durante la creación de las pruebas podría ser necesario realizar pequeños cambios en el diseño de las clases del proyecto original, para simplificar el diseño de las pruebas (por ejemplo, para usar herencia en las pruebas) o para corregir fallos detectados. Los cambios deben ser registrados en el informe final.

3.6. Ejemplos de pruebas

Para las pruebas de unidad y de widgets, hay que añadir el paquete test.dart. Para ello (en Android Studio):

1. En el fichero pubspec.yaml, añadimos

   ```
   dev_dependencies:
     test: any
   ```

2. Instalamos el paquete (hacemos `run flutter pub get` o seleccionamos «get packages»)

3. Importamos el paquete en el fichero de pruebas: `import 'package:test/test.dart';`

3.6.1. Ejemplo de prueba de unidad

Consideremos por ejemplo la clase Medallero en Dart:

```
//clase a probar: Medallero
class Medallero
{
  static Medallero _instance=null;

  int _totalPremios=0;

  static Medallero getInstance() {
    if (_instance==null)
```

```
      _instance=new Medallero();
    return _instance;
  }

  void addPremio() {
    if (_totalPremios < 4)
      _totalPremios++;
    else
      resetPremios();
  }

  void resetPremios()=> _totalPremios=0;

  int get totalPremios => _totalPremios;
}
```

Para crear pruebas de unidad sobre esta clase debemos añadir en la carpeta *test* un fichero de nombre *medallero_test.dart* y escribir el siguiente código en él. En ese código hay tres pruebas y un grupo o suite de pruebas (*group*) que ejecuta las tres en secuencia. El método clave para hacer cada prueba es el método *expect*, que hará que el test pase sí y solo si primer argumento es igual al segundo.

```
//fichero de prueba: medallero_test


import 'package:test/test.dart';
import 'package:flutter_taller_vs/medallero.dart';


void main() {
  group('Medallero', () {
    test('value should start at 0', () {

      final medallero = Medallero.getInstance();
      expect(medallero.totalPremios, 0);
    });

    test('value should be incremented', () {
      final medallero = Medallero.getInstance();

      medallero.addPremio();

      expect(medallero.totalPremios, 1);
    });

    test('value should be resent', () {
```

```
      final medallero = Medallero.getInstance();

      medallero.resetPremios();

      expect(medallero.totalPremios, 0);
    });
  });
}
```

3.6.2. Ejemplo de prueba de widgets

Veamos un ejemplo en el que queremos comprobar que al pulsar un botón (elemento de la interfaz gráfica de usuario) se incrementa un contador (elemento del modelo).

En el caso de Flutter, el código a probar puede ser el de la app que se genera por defecto, la cual solo tiene una página un botón que incrementa un contador (fichero *main.dart*):

```
import 'package:flutter/material.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: 'Flutter Demo',
      theme: ThemeData(
          primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
```

```
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
         child: Column(
           mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have clicked the button this many times:',
            ),
            Text(
                '$_counter',
              style: Theme.of(context).textTheme.headline4
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ),
    );
  }
}
```

Igual que con las pruebas de unidad, para probar los widgets, creamos un fichero, por ejemplo *widget_tests.dart* en el directorio test. Implementamos la función, *testWidgets* –que crea implícitamente un testeador (*WidgetTester*) para cada prueba a realizar– y, dentro de ella, hacemos uso de distintos métodos de *WidgetTester* para las pruebas: *pumpWidget* que despliega un widget; *tap*, para hacer click sobre un widget botón o *pump* que vuelve a

recrear el widget desplegado. A continuación se puede ver un ejemplo de implementación de la prueba. La constante $findsOneWidget$ tiene el valor *true* sí y solo si existe un solo widget que cumpla la condición dada al buscador de widgets (clase Find) en el primer argumento de *expect*:

```
import 'package:flutter/material.dart';
import 'package:flutter_taller_vs/main.dart';
import 'package:flutter_test/flutter_test.dart';

void main() {
  // Define a test. The TestWidgets function also provides a WidgetTester
  // to work with. The WidgetTester allows you to build and interact
  // with widgets in the test environment.
  testWidgets('MyHomePage button', (WidgetTester tester) async {
    // Create the widget by telling the tester to build it.
    await tester.pumpWidget(MyApp());

    expect(find.text('0'), findsOneWidget);
    expect(find.text('1'), findsNothing);

    // Tap the '+' icon and trigger a frame.
    await tester.tap(find.byIcon(Icons.add));
    await tester.pump();

    // Verify that our counter has incremented.
    expect(find.text('0'), findsNothing);
    expect(find.text('1'), findsOneWidget);
  });

}
```