Práctica 3 - Pruebas de desarrolladores

García Verjaga, Ramón
rgarver@correo.ugr.es

Haro Contreras, Daniel
danielharo@correo.ugr.es

4 de mayo de 2021

1. Introducción

En la práctica se ha aprendido a diseñar las pruebas para maximizar el número de errores detectados, se ha aprendido a hacer pruebas de unidad tanto en Dart como en Flutter y a hacer pruebas de componentes en Flutter, se ha comprendido el funcionamiento de las colecciones o grupos de pruebas y se ha investigado cómo hacer pruebas de integración en Flutter.

2. Diseño de las pruebas

Se han diseñado tanto pruebas unitarias, que aseguran que cada funcionalidad del código desarrollado brinda los resultados adecuados, como pruebas de componentes, que aseguran que los widgets tienen un comportamiento correcto.

Además se han desarrollado pruebas de integración, que aseguran que los diferentes módulos del sistema se comunican entre sí de forma correcta, haciendo que todos ellos actúen como parte de un solo sistema.

3. Pruebas de unidad

3.1. Descripción de las condiciones de prueba

Se han desarrollado diferentes pruebas unitarias para probar las funcionalidades de las clases `Student` y `Exam`.

Las pruebas unitarias se han dividido en 4 grupos, dos por cada clase. En cada una de ellas se ha comprobado que es correcto el estado inicial que se le da a las instancias de cada clase. Además, se ha comprobado que: en la clase `Student` las funcionalidades relacionadas con las calificaciones y el cálculo de estadísticas son correctas; y en la clase `Exam` las funcionalidades relacionadas con las calificaciones son correctas.

3.2. Casos de prueba

Pruebas en la clase `Student`:

- Grupo «Estado inicial»: Se ha creado una instancia de `Student` con `id = 1` y `name = 'Juan'`.

- Prueba: El id debe ser 1
- Prueba: El nombre debe ser Juan
- Prueba: La lista de notas debe ser vacía
- Prueba: La media de las calificaciones debe ser un valor nulo
- Prueba: El máximo de las calificaciones debe ser un valor nulo
- Prueba: El mínimo de las calificaciones debe ser un valor nulo

```
void main() {
  group('Estado inicial:', () {
    final student = new Student(1, 'Juan');

    test('El id debe ser 1', () {
      expect(student.id, 1);
    });
    test('El nombre debe ser Juan', () {
      expect(student.name, 'Juan');
    });
    test('La lista de notas debe ser vacía', () {
      expect(student.marks.isEmpty, true);
    });
    test('La media de las calificaciones debe ser un valor nulo', () {
      expect(student.mean, null);
    });
    test('El máximo de las calificaciones debe ser un valor nulo', () {
      expect(student.maximum, null);
    });
    test('El mínimo de las calificaciones debe ser un valor nulo', () {
      expect(student.minimum, null);
    });
  });
```

- Grupo «Calificaciones y estadística»: Se ha creado una instancia de `Student` con `id = 1` y `name = 'Juan'`. Se han creado dos instancias de `Exam`: `exam1` con `name = 'Desarrollo de Software - Tema 1'` y `date = 23/05/2021`; y `exam2` con `name = 'Desarrollo de Software - Tema 2'` y `date = 30/05/2021`. Se ha creado una instancia de `Teaching` y se ha añadido al estudiante la calificación de 5.0 para el primer examen y 10.0 para el segundo examen.

  - Prueba: La lista de notas debe contener dos calificaciones
  - Prueba: La media de las calificaciones debe ser un 7.5
  - Prueba: El máximo de las calificaciones debe ser un 10.0
  - Prueba: El mínimo de las calificaciones debe ser un 5.0

```
  group('Calificaciones y estadística:', () {
    final student = new Student(1, 'Juan');
    final exam1 =
```

```
4         new Exam('Desarrollo de Software - Tema 1', new DateTime(2021,
      05, 23));
5     final exam2 =
6         new Exam('Desarrollo de Software - Tema 2', new DateTime(2021,
      05, 30));
7     final teaching = new Teaching();
8     teaching.addMark(exam1, student, 5.0);
9     teaching.addMark(exam2, student, 10.0);
10
11    test('La lista de notas debe contener dos calificaciones', () {
12      expect(student.marks.length, 2);
13    });
14    test('La media de las calificaciones debe ser un 7.5', () {
15      expect(student.mean, 7.5);
16    });
17    test('El máximo de las calificaciones debe ser un 10.0', () {
18      expect(student.maximum, 10.0);
19    });
20    test('El mínimo de las calificaciones debe ser un 5.0', () {
21      expect(student.minimum, 5.0);
22    });
23  });
24 }
```

Figura 1: Resultados de los test unitarios de la clase `Student`

Pruebas en la clase `Exam`:

- Grupo «Estado inicial»: Se ha creado una instancia de `Exam` con `date = 2021/05/23` y `name = 'Desarrollo de Software - Tema 1'`.
  - Prueba: La fecha del examen debe ser 2021/05/23
  - Prueba: El nombre debe ser Desarrollo de Software - Tema 1
  - Prueba: La lista de notas debe ser vacía
  - Prueba: Intentar acceder a una calificación que no posee un estudiante debe lanzar una excepción. *Gracias a esta prueba logramos realizar una mejora gestionando una excepción cuando se intenta obtener una nota de un estudiante que no existe*

```
void main() {
  group('Estado inicial:', () {
    final exam =
        new Exam('Desarrollo de Software - Tema 1', new DateTime(2021,
    05, 23));

    test('La fecha del examen debe ser 2021/05/23', () {
      expect(exam.date, new DateTime(2021, 05, 23));
    });
    test('El nombre debe ser Desarrollo de Software - Tema 1', () {
      expect(exam.name, 'Desarrollo de Software - Tema 1');
    });
    test('La lista de notas debe ser vacía', () {
      expect(exam.marks.isEmpty, true);
    });
    test(
        'Intentar acceder a una calificación que no posee un estudiante
    debe lanzar una excepción',
        () {
      final student = new Student(1, 'Juan');

      expect(() => exam.getMarkByStudent(student), throwsA(isA<StateError
    >()));
    });
  });
```

- Grupo «Calificaciones»: Se ha creado una instancia de `Student` con `id = 1` y `name = 'Juan'`. Se ha creado una instancia de `Exam`: `exam` con `name = 'Desarrollo de Software - Tema 1'` y `date = 23/05/2021`. Se ha creado una instancia de `Teaching` y se ha añadido al estudiante la calificación de 5.0 para el examen.
  - Prueba: La lista de notas del examen debe contener una calificación
  - Prueba: La única calificación del examen debe ser 5.0

```
  group('Calificaciones:', () {
    final student = new Student(1, 'Juan');
    final exam =
        new Exam('Desarrollo de Software - Tema 1', new DateTime(2021,
    05, 23));
    final teaching = new Teaching();
    teaching.addMark(exam, student, 5.0);

    test('La lista de notas del examen debe contener una calificación',
    () {
      expect(exam.marks.length, 1);
    });
    test('La única calificación del examen debe ser 5.0', () {
      expect(exam.getMarkByStudent(student).grade, 5.0);
    });
```

```
14    });
15  }
```

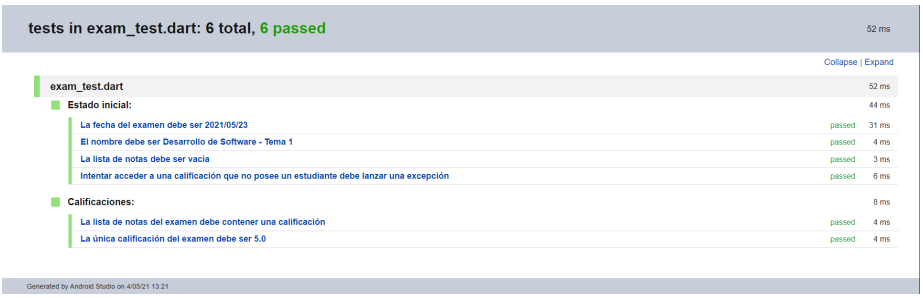

Figura 2: Resultados de los test unitarios de la clase `Exam`

Como se ha podido observar se ha tenido éxito en todas las pruebas de unidad.

4. Pruebas de componentes (widgets)

Se ha creado un archivo de test para cada uno de los widgets correspondientes a `StudentsListView` y `ExamsListView`, que son las dos páginas principales de la app. Como se ha podido observar se ha tenido éxito en todas las pruebas de componentes.

4.1. Casos de prueba

Pruebas en la clase `StudentsListView`:

- Grupo «Página de estudiantes»:
  - Prueba: Al principio no hay ningún estudiante en la lista
  - Prueba: Error al intentar añadir estudiante sin ID
  - Prueba: Error al intentar añadir estudiante sin nombre
  - Prueba: El estudiante se guarda correctamente
  - Prueba: Error al intentar añadir estudiante con ID repetido
  - Prueba: El estudiante se elimina correctamente

```
1  void main() {
2    group("Página de estudiantes", () {
3      testWidgets('Ningún estudiante al principio', (WidgetTester tester)
       async {
4        // Build our app and trigger a frame.
5        await tester.pumpWidget(MyApp());
6
7        // No hay ningún estudiante
8        expect(find.byElementType(ListTile), findsNothing);
```

```
    });

    testWidgets('Intentar añadir estudiante: ID vacío',
        (WidgetTester tester) async {
      await tester.pumpWidget(MyApp());
      // Click en botón +
      await tester.tap(find.byIcon(Icons.add));
      await tester.pumpAndSettle();
      await tester.tap(find.byIcon(Icons.save));
      await tester.pumpAndSettle();
      expect(find.textContaining("ID está vacío"), findsOneWidget);
    });

    testWidgets('Intentar añadir estudiante: nombre vacío',
        (WidgetTester tester) async {
      await tester.pumpWidget(MyApp());
      // Click en botón +
      await tester.tap(find.byIcon(Icons.add));
      await tester.pumpAndSettle();
      var textFields = find.byType(TextFormField);
      await tester.enterText(textFields.first, "1");
      await tester.tap(find.byIcon(Icons.save));
      await tester.pumpAndSettle();
      expect(find.textContaining("nombre está vacío"), findsOneWidget);
    });

    testWidgets('Estudiante queda guardado', (WidgetTester tester) async
    {
      await addStudentNamedJuan(tester);
      expect(find.text("Juan"), findsOneWidget);
    });

    testWidgets('Intentar añadir estudiante: ID repetido',
        (WidgetTester tester) async {
      await addStudentNamedJuan(tester);
      await tester.tap(find.byIcon(Icons.add));
      await tester.pumpAndSettle();
      var textFields = find.byType(TextFormField);
      await tester.enterText(textFields.first, "1");
      await tester.tap(find.byIcon(Icons.save));
      await tester.pumpAndSettle();
      expect(find.textContaining("ID ya existe"), findsOneWidget);
    });

    testWidgets('Estudiante se borra', (WidgetTester tester) async {
      await addStudentNamedJuan(tester);
      await tester.longPress(find.text("Juan"));
      await tester.pumpAndSettle();
      await tester.tap(find.text("Eliminar"));
      await tester.pumpAndSettle();
      expect(find.text("Juan"), findsNothing);
    });
  });
}
```

```
62
63 Future addStudentNamedJuan(WidgetTester tester) async {
64   await tester.pumpWidget(MyApp());
65   // Click en botón +
66   await tester.tap(find.byIcon(Icons.add));
67   await tester.pumpAndSettle();
68   var textFields = find.byType(TextFormField);
69   await tester.enterText(textFields.first, "1");
70   await tester.enterText(textFields.at(1), "Juan");
71   await tester.tap(find.byIcon(Icons.save));
72   await tester.pumpAndSettle();
73 }
```

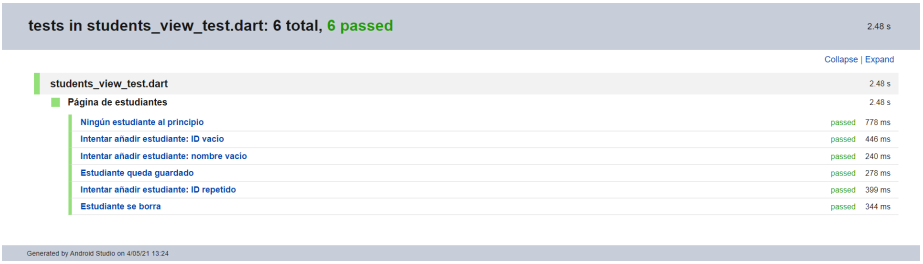

Figura 3: Resultados de los test unitarios de la clase `StudentsListView`

Pruebas en la clase `ExamsListView`:

- Grupo «Página de exámenes»:
  - Prueba: Al principio no hay ningún examen en la lista
  - Prueba: Error al intentar añadir examen sin nombre
  - Prueba: El examen se guarda correctamente
  - Prueba: El examen se elimina correctamente

```
1  void main() {
2    group("Página de exámenes", () {
3      testWidgets('Se abre la pestaña de exámenes', (WidgetTester tester)
       async {
4        await tester.pumpWidget(MyApp());
5        await tester.tap(find.text("Exámenes"));
6        await tester.pump();
7        expect(find.byType(ExamsListView), findsOneWidget);
8      });
9
10     testWidgets('Intentar añadir examen: Nombre vacío',
11         (WidgetTester tester) async {
12       await tester.pumpWidget(MyApp());
13       await tester.tap(find.text("Exámenes"));
14       await tester.pump();
15       await tester.tap(find.byIcon(Icons.add));
16       await tester.pumpAndSettle();
```

```
17        await tester.tap(find.byIcon(Icons.save));
18        await tester.pumpAndSettle();
19        expect(find.textContaining("nombre está vacío"), findsOneWidget);
20      });
21
22      testWidgets('Examen queda guardado', (WidgetTester tester) async {
23        await tester.pumpWidget(MyApp());
24        await tester.tap(find.text("Exámenes"));
25        await tester.pump();
26        await tester.tap(find.byIcon(Icons.add));
27        await tester.pumpAndSettle();
28        var textFields = find.byType(TextFormField);
29        await tester.enterText(textFields, "Examen 1");
30        await tester.tap(find.byIcon(Icons.save));
31        await tester.pumpAndSettle();
32        expect(find.text("Examen 1"), findsOneWidget);
33      });
34
35      testWidgets('Examen se borra', (WidgetTester tester) async {
36        await tester.pumpWidget(MyApp());
37        await tester.tap(find.text("Exámenes"));
38        await tester.pump();
39        await tester.tap(find.byIcon(Icons.add));
40        await tester.pumpAndSettle();
41        var textFields = find.byType(TextFormField);
42        await tester.enterText(textFields, "Examen 1");
43        await tester.tap(find.byIcon(Icons.save));
44        await tester.pumpAndSettle();
45        await tester.longPress(find.text("Examen 1"));
46        await tester.pumpAndSettle();
47        await tester.tap(find.text("Eliminar"));
48        await tester.pumpAndSettle();
49        expect(find.text("Examen 1"), findsNothing);
50      });
51    });
52  }
```

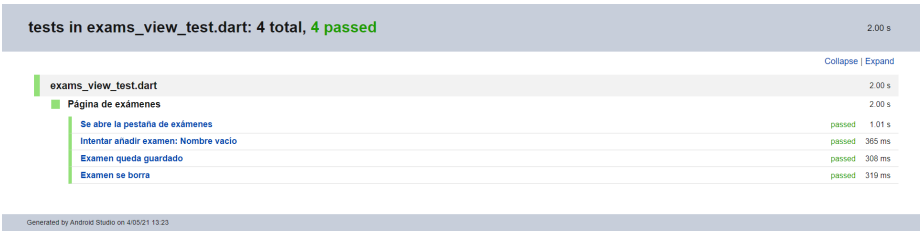

Figura 4: Resultados de los test unitarios de la clase `ExamsListView`

5. Pruebas de integración

Una vez que se han aprobado las pruebas unitarias y las pruebas de componentes, se realizan las pruebas de integración que prueban que todos los elemen-

tos unitarios y componentes que forman el software funcionan juntos correctamente, probándolos en grupo.

Las pruebas de integración implementadas realizan lo siguiente:

- Añadir estudiante Juan
- Añadir examen
- Añadir nota de examen a Juan
- Encontrar nota introducida

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

  group('Test de integración', () {
    testWidgets('Se añade nota al alumno Juan', (WidgetTester tester)
    async {
      app.main();

      await tester.pumpAndSettle();

      await addStudentNamedJuan(tester);

      expect(find.text('Juan'), findsOneWidget);
      expect(find.text("Exámenes"), findsOneWidget);
      await tester.tap(find.text("Exámenes"));
      await tester.pumpAndSettle();
      await addExamNamedExamen1(tester);
      expect(find.text("Examen 1"), findsOneWidget);

      await tester.tap(find.text("Estudiantes"));
      await tester.pumpAndSettle();
      await tester.tap(find.text("Juan"));
      await tester.pumpAndSettle();
      await addMarkToStudent(tester);

      expect(find.text("5.5"), findsWidgets);
    });
  });
}

Future addMarkToStudent(WidgetTester tester) async {
  await tester.tap(find.byIcon(Icons.add));
  await tester.pumpAndSettle();
  await tester.tap(find.text("Nombre"));
  await tester.pumpAndSettle();
  await tester.tap(find.text("Examen 1").first);
  await tester.pumpAndSettle();
  await tester.enterText(find.byType(TextFormField), "5.5");
  await tester.tap(find.byIcon(Icons.save));
  await tester.pumpAndSettle();
}

```

```dart
Future addStudentNamedJuan(WidgetTester tester) async {
  await tester.tap(find.byIcon(Icons.add));
  await tester.pumpAndSettle();
  var textFields = find.byType(TextFormField);
  await tester.enterText(textFields.first, "1");
  await tester.enterText(textFields.at(1), "Juan");
  await tester.tap(find.byIcon(Icons.save));
  await tester.pumpAndSettle();
}

Future addExamNamedExamen1(WidgetTester tester) async {
  await tester.tap(find.byIcon(Icons.add));
  await tester.pumpAndSettle();
  var textFields = find.byType(TextFormField);
  await tester.enterText(textFields, "Examen 1");
  await tester.tap(find.byIcon(Icons.save));
  await tester.pumpAndSettle();
}
```

Figura 5: Resultados de los test de integración de la aplicación

Como se ha podido observar se ha tenido éxito en todas las pruebas de integración.