WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3

UNIVERSIDAD NACIONAL DE COLOMBIA

Taller 3 - Testing
Ingeniería de Software I - 27/02/2025

Introducción

ChazaUNApp es una plataforma pensada y diseñada para la gestión y promoción de pequeños negocios y ventas informales dentro de la Universidad Nacional de Colombia. Inicialmente, permite a los dueños de las chazas o chazeros registrar sus negocios y poder hacer gestión de los estudiantes que serán vendedores o trabajadores de las chazas, así como también le permite a los estudiantes poder vincularse para trabajar en alguna de las chazas.

Resumen de los tests realizados

Test #1:

- **Nombre del integrante:** William Darío Vanegas Marín
- **Tipo de prueba realizada:** Unitaria
- **Descripción breve del componente probado:** El componente probado de la aplicación es el servicio de login, el cuál es indispensable para poder verificar el acceso a la aplicación, tanto por parte del dueño de la chaza o chazero como por parte del estudiante trabajador. De forma particular, esta prueba se va a llevar a cabo sobre la función getEmail() del servicio que en el código de este proyecto se denomina services_login.dart, la cuál busca recuperar una lista de correos electrónicos desde la colección Chazero en la base de datos Firebase Firestore.
- **Herramienta o framework usado:** Una herramienta utilizada es flutter_test, el cuál es un paquete de Flutter que contribuye bastante a desarrollar la prueba unitaria al establecer un marco adecuado para ello; y otra herramienta es mockito que es un paquete de Dart que permite poder realizar la prueba unitaria aunque a diferencia de la anterior, se hace de forma aislada y sin depender de otros módulos de la aplicación.
- **Screenshot del código del test:**

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3

UNIVERSIDAD NACIONAL DE COLOMBIA

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:mockito/mockito.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:chazaunapp/Services/services_login.dart';

class MockFirebaseFirestore extends Mock implements FirebaseFirestore {}
class MockCollectionReference extends Mock implements CollectionReference {}
class MockQuerySnapshot extends Mock implements QuerySnapshot {}
void main() {
  group('Pruebas de servicio de login', () {
    test('getEmail - devuelve lista de correos electrónicos', () async {
      final mockFirestore = MockFirebaseFirestore();
      final mockCollection = MockCollectionReference();
      final mockSnapshot = MockQuerySnapshot();

      when(mockFirestore.collection('Chazero')).thenReturn(mockCollection);
      when(mockCollection.get()).thenAnswer((_) async => mockSnapshot);
      when(mockSnapshot.docs).thenReturn([
        {'email': 'test1@example.com'},
        {'email': 'test2@example.com'}
      ]);

      final emails = await getEmail();

      expect(emails, isA<List>());
      expect(emails.length, 2);
      expect(emails, contains('test1@example.com'));
      expect(emails, contains('test2@example.com'));
    });
  });
}
```

- **Resultado de la ejecución:**

```
00:01 +2: All tests passed!
```

Test #2:

- **Nombre del integrante:** Brahian Camilo Gómez Carvajal
- **Tipo de prueba realizada:** Unitaria
- **Descripción breve del componente probado:** El componente probado de la aplicación es el servicio de registro de chazero , este servicio es esencial para el manejo de autenticación y almacenamiento de datos de usuarios en la aplicación. Se usa principalmente cuando un

usuario intenta registrar o verificar la existencia de su correo antes de crear una cuenta. De forma particular, esta prueba se va a llevar a cabo sobre la función Future<bool> emailExists(String email) async del servicio que en el código de este proyecto se denomina services_registrochazero.dart,la cual verifica si un correo ya está registrado en la colección "Chazero" de Firestore y se utiliza para prevenir registros duplicados y validar cuentas existentes. Y también sobre la función crearChazero(muchos parametros) la cual agrega un nuevo usuario "Chazero" a la base de datos con sus datos personales y de contacto y se usa durante el proceso de registro de nuevos usuarios en la aplicación.

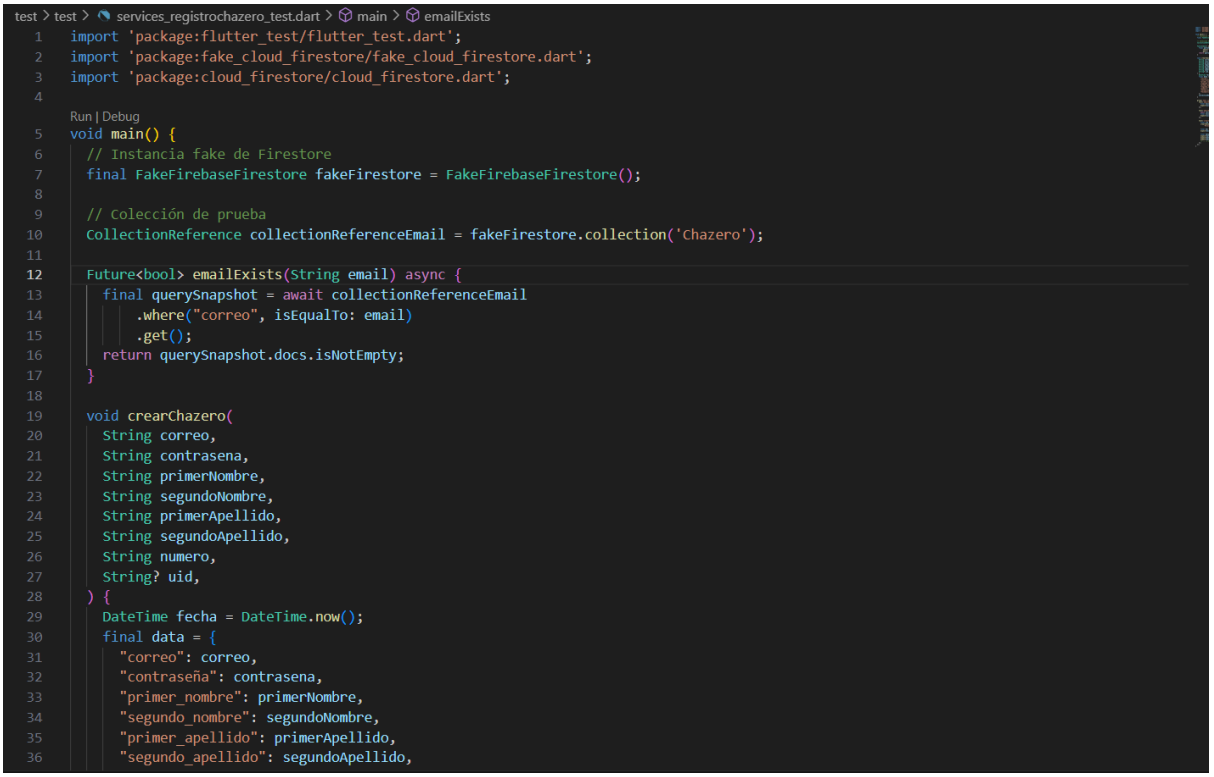
- **Herramientao framework usado:** Básicamente lo mismo que mi compañero anterior, flutter_test para ejecutar los tests unitarios y mockito para mockear las interacciones con Firebase Firestore y evitar llamadas reales a la base de datos.

- **Screenshot del código del test:**

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co

UNIVERSIDAD NACIONAL DE COLOMBIA

Taller 3

- 

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3

UNIVERSIDAD NACIONAL DE COLOMBIA

```
      "numero": numero,
      "FechaCreacion": fecha,
      "FechaUltimaActualizacion": fecha
    };
    collectionReferenceEmail.doc(uid).set(data);
  }

  Run | Debug
  group('Test con FakeFirestore', () {
    Run | Debug
    test('emailExists devuelve false si el email no existe', () async {
      final exists = await emailExists('no_existe@example.com');
      expect(exists, false);
    });

    Run | Debug
    test('emailExists devuelve true si el email existe', () async {
      await collectionReferenceEmail.add({'correo': 'existe@example.com'});
      final exists = await emailExists('existe@example.com');
      expect(exists, true);
    });

    Run | Debug
    test('crearChazero crea un documento con los datos correctos', () async {
      final uid = 'uidTest';
      crearChazero('test@example.com', '123456', 'Juan', 'Carlos',
          'Perez', 'Lopez', '9876543210', uid);

      final doc = await collectionReferenceEmail.doc(uid).get();
      expect(doc.exists, true);

      final data = doc.data() as Map<String, dynamic>?;
      expect(data?['correo'], 'test@example.com');
      expect(data?['contraseña'], '123456');
      expect(data?['primer_nombre'], 'Juan');
```

```
      // ... y así sucesivamente
    });
  });
}
```

- **Resultado de la ejecución:**

```
PS C:\Users\bcami\Downloads\chazaUNApp\chazaUNApp-Final> flutter test test/test/services_registrochazero_test.dart
00:01 +3: All tests passed!
PS C:\Users\bcami\Downloads\chazaUNApp\chazaUNApp-Final>
```

Test #3:

- **Nombre del integrante:** Juan David Rivera Buitrago

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3

- **Tipo de prueba realizada:** Prueba de Integración

- **Descripción breve del componente probado:** Se probó la función ***getChazasporChazero***, ubicada en el archivo ***services_menuchazero.dart***, que tiene la responsabilidad de recuperar todas las chazas registradas que pertenecen a un determinado chazero dentro de la plataforma ChazaUNApp. La función ejecuta una consulta en la colección ***"Chaza"*** de Firebase Firestore, filtrando por el ID del chazero proporcionado. Luego, procesa los documentos obtenidos y devuelve una lista con la información de cada chaza, incluyendo su nombre, ubicación y detalles relevantes. Esta funcionalidad es esencial para garantizar que los dueños de los puestos de venta puedan visualizar correctamente las chazas que administran en la plataforma.

- **Herramienta o framework usado:** Para la implementación de esta prueba se utilizaron herramientas diseñadas específicamente para el entorno de Flutter. Se empleó ***fake_cloud_firestore***, una biblioteca que permite simular Firestore en memoria, lo que facilita la validación de consultas sin depender de una conexión a internet o afectar la base de datos real. Además, se utilizó ***flutter_test***, el framework de pruebas integrado en Flutter, el cual proporciona un entorno adecuado para la ejecución de pruebas unitarias e integradas, asegurando que el servicio funcione correctamente en condiciones controladas y contribuyendo a la estabilidad general del sistema.

- **Screenshot del código del test:**

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3

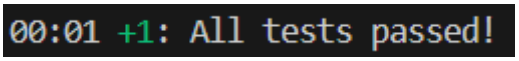
UNIVERSIDAD NACIONAL DE COLOMBIA

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:fake_cloud_firestore/fake_cloud_firestore.dart';
import 'package:chazaunapp/Services/services_menuchazero.dart'; // Ruta real del servicio


class ChazaService {
  final FakeFirebaseFirestore firestore;


  ChazaService({required this.firestore});


  Future<List<Map<String, dynamic>>> getChazasporChazero(String idChazero, {FakeFirebaseFirestore? firestore}) async {
    final db = firestore ?? this.firestore; // Usar Firestore inyectado o el predeterminado
    List<Map<String, dynamic>> chazas = [];
    QuerySnapshot querySnapshot = await db
        .collection('Chaza')
        .where('ID_Chazero', isEqualTo: idChazero)
        .get();
    for (var doc in querySnapshot.docs) {
      chazas.add(doc.data() as Map<String, dynamic>);
    }
    return chazas;
  }
}


void main() {
  group('Prueba de recuperación de chazas por Chazero', () {
    late FakeFirebaseFirestore fakeFirestore;
    late ChazaService chazaService;


    setUp(() {
      fakeFirestore = FakeFirebaseFirestore();
      chazaService = ChazaService(firestore: fakeFirestore);
    });
```

```dart
    test('Recuperación de chazas por ID de Chazero', () async {
      final String idChazero = 'chazero123';
      final List<Map<String, dynamic>> fakeChazas = [
        {'nombre': 'Chaza 1', 'ubicacion': 'Bloque A', 'ID_Chazero': idChazero},
        {'nombre': 'Chaza 2', 'ubicacion': 'Bloque B', 'ID_Chazero': idChazero},
      ];


      // Insertar datos falsos en Firestore simulado
      for (var chaza in fakeChazas) {
        await fakeFirestore.collection('Chaza').add(chaza);
      }


      // Llamar la función de prueba pasando la base de datos simulada
      final resultado = await chazaService.getChazasporChazero(idChazero, firestore: fakeFirestore);


      // Verificar que los datos devueltos sean correctos
      expect(resultado.length, 2);
      expect(resultado[0]['nombre'], 'Chaza 1');
      expect(resultado[1]['nombre'], 'Chaza 2');
    });
  });
}
```

- **Resultado de la ejecución:**

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3



Test #4:

- **Nombre del integrante:** Wullfredo Javier Barco Godoy

- **Tipo de prueba realizada:** Unitaria

- **Descripción breve del componente probado:** El archivo services_menu_inicial.dart define una función llamada getChazas que se encarga de interactuar con Firestore para obtener todos los documentos de la colección "Chaza". En su implementación, primero se accede a la instancia global de Firestore mediante FirebaseFirestore.instance, y a partir de allí se crea una referencia a la colección "Chaza". La función realiza una consulta para obtener todos los documentos presentes en esa colección y, para cada documento obtenido, convierte sus datos a un mapa de clave-valor. Además, añade el identificador único del documento al mapa bajo la clave "id", lo que permite tener acceso tanto a los datos como al identificador en la lista resultante. Finalmente, la función retorna la lista completa de mapas que representan cada documento de la colección.

- **Herramienta o framework usado:** Para probar esta función se utiliza el framework flutter_test, que es el entorno estándar de pruebas unitarias en Flutter, junto con fake_cloud_firestore, una herramienta que simula el comportamiento de Firestore de forma local. Esto permite ejecutar las pruebas sin necesidad de conectarse a una base de datos real, inyectando una instancia simulada en la variable global db y comprobando que la función getChazas obtiene y procesa correctamente los datos de la colección "Chaza".

- **Screenshot del código del test:**

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co

UNIVERSIDAD NACIONAL DE COLOMBIA

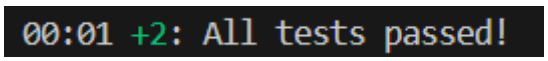
Taller 3

test > test > services_menu_inicial_test.dart > ...

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:fake_cloud_firestore/fake_cloud_firestore.dart';
import 'package:chazaunapp/Services/services_menu_inicial.dart' as menu;


Run | Debug
void main() {
  Run | Debug
  group('Pruebas de getChazas en services_menu_inicial.dart', () {
    late FakeFirebaseFirestore fakeFirestore;

    setUp(() {
      // Crea una instancia falsa de Firestore.
      fakeFirestore = FakeFirebaseFirestore();

      // Sobrescribe la variable global "db" del archivo de servicios para que use la instancia fake.
      menu.db = fakeFirestore;
    });

    Run | Debug
    test('getChazas retorna una lista vacía si no hay documentos', () async {
      // Aseguramos que la colección "Chaza" esté vacía.
      final chazas = await menu.getChazas();
      expect(chazas, isEmpty);
    });

    Run | Debug
    test('getChazas retorna la lista de documentos con sus IDs', () async {
      // Inserta algunos documentos falsos en la colección "Chaza".
      await fakeFirestore.collection('Chaza').doc('doc1').set({
        'nombre': 'Chaza 1',
        'direccion': 'Dirección 1'
      });
      await fakeFirestore.collection('Chaza').doc('doc2').set({
        'nombre': 'Chaza 2',
        'direccion': 'Dirección 2'
      });

      // Llama a la función a testear.
      final chazas = await menu.getChazas();

      // Se espera que se retornen 2 documentos.
      expect(chazas.length, 2);

      // Verifica que cada objeto incluya el ID correcto y los datos correspondientes.
      bool doc1Found = false;
      bool doc2Found = false;

      for (final chaza in chazas) {
        if (chaza['id'] == 'doc1' && chaza['nombre'] == 'Chaza 1') {
          doc1Found = true;
        }
        if (chaza['id'] == 'doc2' && chaza['nombre'] == 'Chaza 2') {
          doc2Found = true;
        }
      }

      expect(doc1Found, isTrue);
      expect(doc2Found, isTrue);
    });
  });
}
```

WULLFREDO JAVIER BARCO GODOY - wbarco@unal.edu.co
BRAHIAN CAMILO GÓMEZ CARVAJAL - bgomezca@unal.edu.co
JUAN DAVID RIVERA BUITRAGO - jriverabu@unal.edu.co
WILLIAM DARÍO VANEGAS MARÍN - wvanegas@unal.edu.co
Taller 3

- **Resultado de la ejecución:**



Lecciones aprendidas y dificultades

Uno de los principales aprendizajes obtenidos durante la realización de este proyecto fue la importancia de implementar pruebas tanto unitarias como de integración para garantizar la estabilidad y funcionalidad del sistema. A lo largo del proceso, se exploraron diferentes herramientas de prueba dentro del ecosistema de Flutter y Dart, lo que permitió profundizar en el uso de paquetes como ***flutter_test, mockito y fake_cloud_firestore***. Sin embargo, el aprendizaje inicial sobre estas herramientas requirió tiempo, especialmente para configurar correctamente los entornos de prueba y lograr que las pruebas reflejaran adecuadamente el comportamiento del sistema en producción.

Entre las principales dificultades encontradas, una de las más relevantes fue la simulación de Firestore sin depender de una base de datos real. Inicialmente, se intentó utilizar ***mockito*** y ***cloud_firestore_mock***, pero estos paquetes presentaban problemas de configuración, requerían una gran cantidad de código adicional y en algunos casos generaban errores al intentar simular la estructura de Firestore. Esto llevó a la necesidad de buscar alternativas más eficientes, encontrando en ***fake_cloud_firestore*** una solución mucho más adecuada. Esta herramienta permitió realizar pruebas en una base de datos simulada en memoria, eliminando la dependencia de una conexión a internet y facilitando la validación de consultas en Firestore de una manera más realista y sencilla.

Otra dificultad fue el diseño de pruebas que realmente validaran los distintos casos de uso de la aplicación. Se aprendió que no solo es necesario verificar el comportamiento esperado cuando las funciones reciben datos correctos, sino también probar escenarios en los que ocurren errores, como la ausencia de datos, entradas inválidas o fallas en la conexión con la base de datos. Implementar estos casos de prueba ayudó a mejorar la calidad del código, haciendo que la aplicación sea más robusta frente a errores inesperados.

Finalmente, la experiencia de testear resaltó la importancia de integrar las pruebas en el proceso de desarrollo desde etapas tempranas. Inicialmente, las pruebas se veían como una tarea secundaria, pero con el avance del proyecto se comprendió que un enfoque basado en pruebas desde el inicio permite detectar errores más rápidamente y evitar retrabajos innecesarios. Además, se observó que una estrategia de testing bien implementada mejora la confianza en el código, asegurando que las funcionalidades críticas del sistema se mantengan estables a medida que el desarrollo avanza.

A pesar de los retos enfrentados, la implementación de pruebas en el proyecto permitió mejorar la fiabilidad y estabilidad del sistema. Se adquirió un conocimiento más profundo

sobre el uso de herramientas de prueba en Flutter, se optimizó el tiempo de desarrollo mediante la selección de las herramientas adecuadas y se fortaleció la capacidad del equipo para diseñar pruebas efectivas que garanticen el correcto funcionamiento de la aplicación.