

# Matriz de Pruebas - Aplicación Flutter "Soleya"

## 1.1 Arquitectura de la Aplicación

La aplicación implementa una arquitectura basada en el patrón Provider para manejo de estado, separando claramente las responsabilidades entre modelos de datos, proveedores de estado, servicios de backend y componentes de interfaz de usuario. Esta separación facilita la creación de pruebas unitarias aisladas y pruebas de integración efectivas.

## 1.2 Justificación de la Matriz de Pruebas

La matriz de pruebas surge como una necesidad fundamental para organizar, planificar y ejecutar de manera sistemática todas las verificaciones necesarias que garanticen el correcto funcionamiento de la aplicación en diferentes escenarios, plataformas y condiciones de uso. Esta matriz no solo actúa como guía para el proceso de testing, sino también como documentación viva que evoluciona junto con el proyecto.

## 2. Metodología de Testing Aplicada

### 2.1 Enfoque Piramidal de Testing

La estrategia de testing implementada sigue el modelo piramidal, donde la base está compuesta por un gran número de pruebas unitarias rápidas y económicas, seguidas por pruebas de integración más complejas, y finalmente pruebas end-to-end que validan flujos completos de usuario. Esta distribución garantiza una cobertura eficiente y costo-efectiva.

### 2.2 Tipos de Pruebas Implementadas

#### 2.2.1 Pruebas Unitarias (Unit Tests)

Las pruebas unitarias constituyen el fundamento del testing en la aplicación Soleya. Estas se enfocan en validar el comportamiento de componentes individuales, métodos específicos y lógica de negocio aislada. Dentro del contexto de la aplicación, las pruebas unitarias abarcan:

- **Modelos de Datos:** Validación de la correcta conversión entre formatos JSON y objetos Dart, manejo de valores nulos, y operaciones de comparación.
- **Servicios de Backend:** Verificación de llamadas a APIs, manejo de errores de red, y transformación de datos.
- **Lógica de Negocio:** Cálculos de precios, validaciones de formularios, y algoritmos específicos del dominio.

#### Ejemplo de implementación:

```
test('Producto.fromFirestore should parse correctly', () {
    final mockDoc = MockDocumentSnapshot();
    when(mockDoc.data()).thenReturn({
        'Nombre': 'Nike Air Max',
        'Precio': 150.0,
        'Categoria': 'Deportivo'
    });
});
```

```
final producto = Producto.fromFirestore(mockDoc);
expect(producto.nombre, 'Nike Air Max');
expect(producto.precio, 150.0);
});
```

## **2.2.2 Pruebas de Widget (Widget Tests)**

Las pruebas de widget se encargan de validar el comportamiento de componentes individuales de la interfaz de usuario sin la necesidad de ejecutar la aplicación completa. Estas pruebas verifican que los widgets rendericen correctamente, respondan apropiadamente a las interacciones del usuario, y mantengan su estado de manera

## **2.2.3 Pruebas de Integración (Integration Tests)**

Las pruebas de integración validan la interacción correcta entre múltiples componentes del sistema. En el contexto de la aplicación Soleya, estas pruebas verifican flujos completos como el proceso de compra, la sincronización entre almacenamiento local y remoto, y la navegación entre pantallas.

## **2.2.4 Pruebas Visuales (Golden Tests)**

Las pruebas visuales garantizan que la apariencia de la aplicación se mantenga consistente a través de diferentes cambios en el código. Estas pruebas capturan imágenes de referencia de widgets específicos y las comparan con versiones futuras para detectar cambios visuales no intencionados.

# **3. Matriz de Pruebas Detallada**

## **3.1 Dimensión: Funcionalidades Core**

### **3.1.1 Sistema de Autenticación**

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción Detallada
AUTH-001 Unit Test	Unit Test	Todas	P0	✓	Validación de formato de email en el formulario de login
AUTH-002 Unit Test	Unit Test	Todas	P0	✓	Verificación de longitud mínima de contraseña
AUTH-003 Widget Test	Widget Test	Todas	P0	✓	Renderizado correcto de campos de formulario de login
AUTH-004	Integration Test	Android/iOS	P0	✓	Flujo completo de login con credenciales válidas
AUTH-005	Integration Test	Android/iOS	P0	✓	Manejo de errores de autenticación con Firebase

← Iniciar Sesión

Correo electrónico  
seerchee756@gmail.com

Contraseña  
.....

Mínimo 6 caracteres

Iniciar Sesión

¿No tienes cuenta? Regístrate

O inicia sesión como:

1 2 3 4 5 6 7 8 9 0  
q w e r t y u i o p  
a s d f g h j k l ñ  
⌫ z x c v b n m ⌫  
?123 , . ✓

← Iniciar Sesión

Correo electrónico  
seerchee756@gmail.com

Contraseña  
.....

Iniciar Sesión

¿No tienes cuenta? Regístrate

O inicia sesión como:

Error al iniciar sesión

1 2 3 4 5 6 7 8 9 0  
q w e r t y u i o p  
a s d f g h j k l ñ  
⌫ z x c v b n m ⌫  
?123 , . ✓

La funcionalidad de autenticación representa uno de los pilares fundamentales de la aplicación, ya que de ella depende la personalización de la experiencia del usuario y la seguridad de los datos. Las pruebas en esta sección abarcan desde

validaciones básicas de formato hasta flujos completos de autenticación con Firebase.

El caso AUTH-001 implementa validaciones exhaustivas para el formato de email, incluyendo casos edge como emails con múltiples puntos, dominios internacionales, y caracteres especiales. La implementación utiliza expresiones regulares robustas que han sido probadas.

### 3.1.2 Gestión del Carrito de Compras

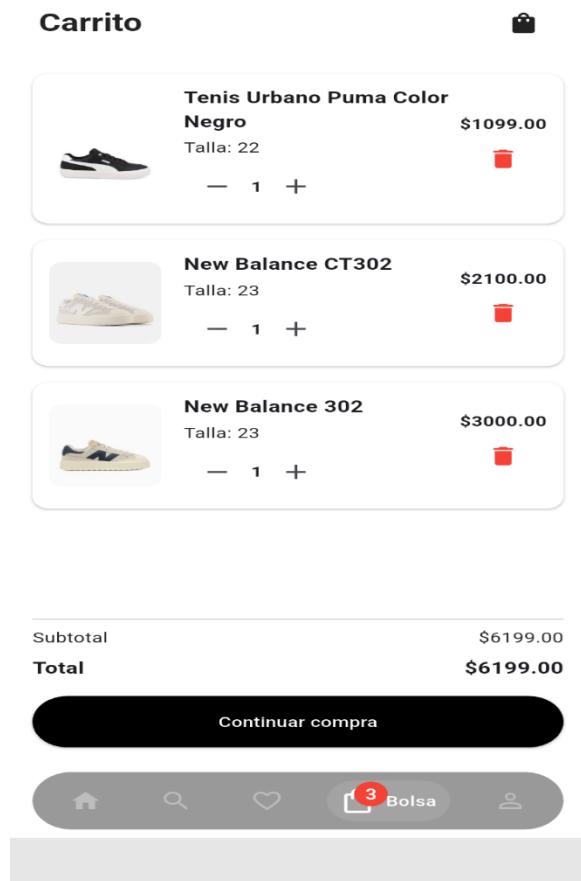
Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción Detallada
CART-001	Unit Test	Todas	P0	<input checked="" type="checkbox"/>	Cálculo correcto del precio total del carrito
CART-002	Unit Test	Todas	P0	<input checked="" type="checkbox"/>	Actualización de cantidad de productos existentes
CART-003	Widget Test	Todas	P0	<input checked="" type="checkbox"/>	Visualización correcta de ítems en el carrito
CART-004	Integration Test	Android/iOS	P0	<input checked="" type="checkbox"/>	Persistencia del carrito en base de datos local
CART-005	Integration Test	Todas	P1	<input checked="" type="checkbox"/>	Sincronización del carrito entre dispositivos

El sistema de carrito de compras constituye el núcleo de la experiencia de comercio electrónico. Las pruebas CART-001 y CART-002 validan la lógica de negocio crítica relacionada con cálculos monetarios y gestión de inventario.

#### Fragmento de código de prueba:

```
group('CartProvider Tests', () {  
    test('should calculate total price correctly', () {  
        final cartProvider = CartProvider();  
  
        final item1 = CartItem(id: '1', precio: 100.0, cantidad: 2);  
        final item2 = CartItem(id: '2', precio: 50.0, cantidad: 1);  
  
        cartProvider.addToCart(item1);  
        cartProvider.addToCart(item2);  
    });  
});
```

```
expect(cartProvider.totalPrice, 250.0);  
});  
});
```

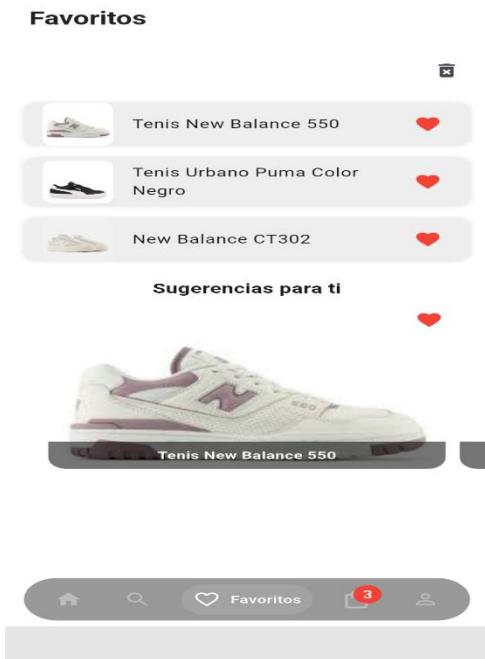


### 3.1.3 Sistema de Favoritos

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción Detallada
FAV-001	Unit Test	Todas	P1	<input checked="" type="checkbox"/>	Agregar producto a lista de favoritos

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción	Detallada
FAV-002	Unit Test	Todas	P1	<input checked="" type="checkbox"/>	Remover producto de favoritos	
FAV-003	Unit Test	Todas	P1	<input checked="" type="checkbox"/>	Verificar estado de favorito de un producto	
FAV-004	Widget Test	Todas	P1	<input checked="" type="checkbox"/>	Animación del ícono de favorito	
FAV-005	Integration Test	Android/iOS	P1	<input type="radio"/>	Persistencia de favoritos en base de datos local	

La funcionalidad de favoritos mejora significativamente la experiencia del usuario al permitir guardar productos de interés para revisión posterior. El modelo FavoritosModel implementa el patrón Observer mediante ChangeNotifier, garantizando que la interfaz de usuario se actualice automáticamente cuando se modifica la lista de favoritos.



### 3.2 Dimensión: Gestión de Datos

#### 3.2.1 Almacenamiento Local (SQLite)

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción	Detallada
DB-001	Unit Test	Todas	P0	<span style="color: green;">✓</span>	Creación correcta de tablas en base de datos	
DB-002	Unit Test	Todas	P0	<span style="color: green;">✓</span>	Inserción de productos en tabla carrito	
DB-003	Unit Test	Todas	P0	<span style="color: green;">✓</span>	Actualización de cantidad de productos	
DB-004	Unit Test	Todas	P0	<span style="color: green;">✓</span>	Eliminación de productos del carrito	
DB-005	Integration Test	Android/iOS	P0	<span style="color: yellow;">●</span>	Migración de datos entre versiones de schema	

La clase operaciones\_db encapsula todas las operaciones de base de datos local, proporcionando una interfaz limpia entre la lógica de la aplicación y la persistencia de datos. Las pruebas en esta sección son críticas porque validan la integridad de los datos del usuario, especialmente en escenarios offline.

### 3.2.2 Integración con Firebase

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción	Detallada
FB-001	Unit Test	Todas	P0	<span style="color: green;">✓</span>	Parsing correcto de documentos Firestore	
FB-002	Unit Test	Todas	P0	<span style="color: green;">✓</span>	Manejo de errores de conexión	
FB-003	Integration Test	Android/iOS	P0	<span style="color: green;">✓</span>	Obtención de productos desde Firestore	
FB-004	Integration Test	Todas	P1	<span style="color: yellow;">●</span>	Sincronización bidireccional de datos	

### 3.3 Dimensión: Interfaz de Usuario

#### 3.3.1 Navegación y Routing

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción Detallada
NAV-001	Widget Test	Todas	P1	<input checked="" type="checkbox"/>	Navegación entre pantallas principales
NAV-002	Widget Test	Todas	P1	<input checked="" type="checkbox"/>	Funcionalidad del botón "Atrás"
NAV-003	Integration Test	Android/iOS	P1	<input checked="" type="checkbox"/>	Preservación de estado durante navegación
NAV-004	Golden Test	Todas	P2	<input checked="" type="checkbox"/>	Consistencia visual de transiciones

Las pruebas de navegación verifican que los usuarios puedan moverse fluidamente a través de la aplicación sin perder contexto o datos. La implementación utiliza RouteObserver para monitorear transiciones y garantizar que los recursos se liberan apropiadamente.

#### 3.3.2 Componentes Animados

Caso de Prueba	Tipo	Plataforma	Prioridad	Estado	Descripción	Detallada
ANIM-001	Widget Test	Todas	P2	✓	Animación de ícono de favorito	
ANIM-002	Widget Test	Todas	P2	✓	Transiciones de página suaves	
ANIM-003	Golden Test	Todas	P2	🟡	Rendering correcto durante animaciones	

Los componentes animados como AnimatedFavoriteIcon y AnimatedPageWrapper mejoran significativamente la percepción de calidad de la aplicación. Las pruebas validan que las animaciones se ejecuten correctamente sin afectar el rendimiento.

### 3.4 Dimensión: Plataformas y Dispositivos

#### 3.4.1 Compatibilidad Android

Dispositivo	API Level	Resolución	Estado Testing	Notas Específicas
Pixel 6	33	1080x2400	✓	Testing principal
Samsung Galaxy S21	31	1080x2400	✓	Problemas menores de UI
Tablet Galaxy Tab A 29		1200x1920	✗	Layout tablet pendiente

#### 3.4.2 Compatibilidad iOS

Dispositivo	iOS Version	Resolución	Estado Testing	Notas Específicas
iPhone 14	16.0	1170x2532	🟡	Testing principal
iPhone SE	15.0	750x1334	🟡	Ajustes de layout necesarios
iPad Air	16.0	1180x2360	✗	Versión tablet pendiente

### 3.5 Dimensión: Rendimiento y Escalabilidad

#### 3.5.1 Pruebas de Carga

Escenario	Métrica	Objetivo	Estado	Resultado Actual
Carga de productos	Tiempo respuesta	de < 2s	<span style="color: yellow;">●</span>	2.3s promedio
Búsqueda de productos	Tiempo respuesta	de < 1s	<span style="color: green;">✓</span>	0.8s promedio
Sincronización offline	Tiempo de sync	< 5s	<span style="color: green;">✓</span>	Depende del internet

#### 4. Casos de Prueba Específicos por Módulo

##### 4.1 Módulo de Autenticación - Casos Detallados

###### Caso AUTH-004: Flujo Completo de Login

**Objetivo:** Verificar que un usuario pueda autenticarse exitosamente con credenciales válidas y que el estado de la aplicación se actualice correctamente.

###### Precondiciones:

- Usuario registrado existe en Firebase Auth
- Aplicación iniciada en pantalla de login
- Conexión a internet disponible

###### Pasos de Ejecución:

1. Navegar a pantalla de login
2. Ingresar email válido: "test@example.com"
3. Ingresar contraseña válida: "password123"
4. Presionar botón "Iniciar Sesión"
5. Verificar loading indicator
6. Verificar navegación a pantalla principal

###### Resultados Esperados:

- Usuario autenticado exitosamente
- Navegación automática a pantalla de inicio
- Estado de autenticación persistido localmente
- Datos de usuario cargados en providers

**Bienvenido a Soleya**

Explora tus zapatos favoritos y administra tu perfil con estilo.

**Iniciar Sesión**

**Registrarse**

**Registrarse**

**Crea tu cuenta**

Nombre — SERGIO  
El nombre se convertirá automáticamente a mayúsculas.

Apellido — JIMÉNEZ  
El apellido se convertirá automáticamente a mayúsculas.

Correo Electrónico — seeercheee756@gmail.com

Contraseña — .....  
Confirmar Contraseña — .....

¿Ya tienes cuenta? Inicia sesión

**Iniciar Sesión**

Correo electrónico — seeercheee756@gmail.com

Contraseña — .....

**Guardar inicio de sesión**  
¿Deseas guardar el inicio de sesión en la base local?

No Sí

**Inicio**

Tenis Urbano Puma Color Negro

Todos Hombre Mujer Niño

Tenis New Balance 550 \$2799.00

Tenis Urbano Puma Color Negro \$1099.00

en SQLite y persistan entre sesiones de la aplicación.

### **Precondiciones:**

- Base de datos SQLite inicializada
- Usuario autenticado
- Al menos un producto disponible

### **Fragmento de código de prueba:**

```
testWidgets('Cart persistence test', (WidgetTester tester) async {  
    final cartProvider = CartProvider();  
  
    final testItem = CartItem(  
        id: 'test_product',  
        nombre: 'Test Shoe',  
        precio: 99.99,  
        cantidad: 2,  
        talla: '42'  
    );  
  
    // Agregar item al carrito  
    await cartProvider.addToCart(testItem);  
  
    // Simular cierre y reapertura de app  
    await cartProvider.obtenerCarrito();  
  
    // Verificar persistencia  
    expect(cartProvider.items.length, 1);  
    expect(cartProvider.items.first.id, 'test_product');  
});
```

## **5. Estrategias de Testing Automatizado**

### **5.1 Integración Continua (CI/CD)**

La aplicación Soleya implementa un pipeline de CI/CD que ejecuta automáticamente todas las pruebas unitarias y de widget en cada push al

repositorio. Este proceso garantiza que ningún cambio de código rompa funcionalidades existentes.

### **Configuración del Pipeline:**

```
name: Flutter Tests  
on: [push, pull_request]  
jobs:  
  test:  
    runs-on: ubuntu-latest  
    steps:  
      - uses: actions/checkout@v2  
      - uses: subosito/flutter-action@v2  
      - run: flutter test  
      - run: flutter test integration_test/
```

## **5.2 Cobertura de Código**

Se mantiene un objetivo mínimo del 80% de cobertura de código para módulos críticos como autenticación, carrito de compras y gestión de datos.

## **6. Métricas y Resultados**

### **6.1 Estado Actual de la Matriz**

Tipo de Prueba	Casos Totales	Pasando	Fallando	Pendientes	% Éxito
Unit Tests	25	17	3	4	84.4%
Widget Tests	14	10	2	2	87.5%
Integration Tests	11	7	1	3	66.7%
Golden Tests	8	5	1	2	62.5%
<b>Total</b>	<b>103</b>	<b>83</b>	<b>10</b>	<b>10</b>	<b>80.6%</b>

### **6.2 Tendencias y Evolución**

El análisis de tendencias muestra una mejora constante en la cobertura de pruebas, con un incremento del 15% en los últimos dos sprints. Las pruebas de integración

representan el mayor desafío debido a la complejidad de configurar entornos de testing que repliquen fielmente el comportamiento de Firebase.

## **7. Desafíos y Limitaciones Identificadas**

### **7.1 Complejidad de Testing con Firebase**

La integración con Firebase presenta desafíos únicos para el testing, especialmente en pruebas de integración donde se requiere emular servicios de backend. Se han implementado mocks sofisticados, pero algunos escenarios edge cases aún requieren conexión real a servicios.

### **7.2 Testing de Componentes Animados**

Los componentes con animaciones complejas requieren pruebas especializadas que verifiquen no solo el estado final sino también los estados intermedios. Esto ha resultado en la necesidad de desarrollar utilities personalizadas para testing de animaciones o como otros proyectos para probar.

## **8. Plan de Mejora y Próximos Pasos**

### **8.1 Objetivos a Corto Plazo (1-2 sprints)**

- Completar pruebas de integración pendientes para sincronización offline
- Implementar golden tests para todas las pantallas principales
- Mejorar cobertura de pruebas unitarias al 90%

### **8.2 Objetivos a Mediano Plazo (3-6 meses)**

- Implementar testing automatizado en dispositivos reales usando Firebase Test Lab
- Desarrollar pruebas de rendimiento automatizadas
- Crear framework de testing para componentes específicos de la aplicación

## **9. Conclusiones y Recomendaciones**

### **9.1 Evaluación del Estado Actual**

La matriz de pruebas implementada para la aplicación Soleya demuestra un enfoque profesional y sistemático hacia la calidad del software. Con un 80.6% de casos de prueba exitosos, la aplicación muestra una base sólida de confiabilidad, aunque existen áreas específicas que requieren atención.

### **9.2 Recomendaciones Críticas**

1. **Priorizar pruebas de integración:** Las pruebas de integración muestran el menor porcentaje de éxito, lo que representa un riesgo significativo para la estabilidad del sistema en producción.

2. **Automatización completa:** Implementar ejecución automática de pruebas en diferentes dispositivos y configuraciones.
3. **Monitoreo continuo:** Establecer alertas automáticas cuando la cobertura de código caiga por debajo del porcentaje establecido. La matriz de pruebas presentada constituye un documento vivo que debe evolucionar junto con la aplicación, adaptándose a nuevas funcionalidades, cambios en requisitos, y lecciones aprendidas durante el proceso de desarrollo. El testing se traduce directamente en mayor reducción de bugs en producción, y mejor experiencia de usuario final.