SCHOOLME

ACADEMICS

Pruebas Unitarias Aplicativo Movil

Centro de la industria, la empresa y los servicios
ADSO 2900177

SCHOOLME ACADEMICS

Grupo de Desarollo SchollMe

Jesus Fernando Carvajal jesusanacona017@gmail.com
Santiago Chaparro Riaño alfasan1481@gmail.com

MANUAL DE PRUEBAS UNITARIAS – SCHOOLMEMOVIL
Versión: 1.0
Autor: Equipo de desarrollo SchoolMeMovil
Fecha: Noviembre 2025
Confidencialidad: Uso interno del equipo de desarrollo

---

1. Introducción

1.1 Propósito

El presente documento describe el proceso, estructura y ejecución de las pruebas unitarias implementadas en el proyecto **SchoolMeMovil**, una aplicación móvil desarrollada en **React Native** con **TypeScript**.
El objetivo de las pruebas unitarias es garantizar la calidad del código mediante la verificación de cada componente, servicio y módulo de forma aislada.

1.2 Alcance

Este manual cubre:

- Configuración del entorno de pruebas.
- Estructura de archivos y convenciones.
- Ejecución de comandos de prueba.
- Descripción detallada de los casos de prueba implementados.
- Estrategias de mantenimiento y mejora continua.

1.3 Beneficios

- Detección temprana de errores.
- Mayor confiabilidad del sistema.
- Reducción de tiempos de depuración.
- Facilita la integración continua (CI/CD).

---

2. Configuración del Entorno de Pruebas

2.1 Dependencias necesarias

En el proyecto, se utiliza **Jest** como framework principal para pruebas unitarias, junto con librerías auxiliares.

```
npm install --save-dev jest @testing-library/react-native @types/jest jest-expo
npm install --save-dev ts-jest
```

2.2 Configuración de Jest

En el archivo package.json se define la configuración principal:

```
"jest": {
  "preset": "jest-expo",
  "transform": {
  "^.+\\.(js|ts|tsx)$": "ts-jest"
  },
  "setupFilesAfterEnv": ["@testing-library/jest-native/extend-expect"],
  "testPathIgnorePatterns": ["/node_modules/", "/android/", "/ios/"]
}
```

2.3 Mocks y entorno simulado

Se implementan **mocks** para simular dependencias como Axios y AsyncStorage.
Ejemplo (__mocks__/axios.js):

```
export default {
  get: jest.fn(() => Promise.resolve({ data: {} })),
  post: jest.fn(() => Promise.resolve({ data: {} }))
};
```

---

3. Estructura de las Pruebas

3.1 Ubicación

Las pruebas se almacenan dentro de la carpeta __tests___en cada módulo o en /src/__tests__/ a nivel global.

```
src/
├── components/
│   ├── CustomButton.tsx
│   ├── CustomInput.tsx
│   └── __tests__/
│       ├── CustomButton.test.tsx
│       └── CustomInput.test.tsx
├── services/
│   └── __tests__/
│       ├── AuthService.test.ts
│       └── TokenManager.test.ts
```

3.2 Convenciones

- Los archivos de prueba terminan con .test.ts o .test.tsx.
- Se sigue el patrón **AAA** (Arrange, Act, Assert).
- Cada prueba debe ser independiente.
- Los mocks deben limpiarse tras cada ejecución (afterEach(jest.clearAllMocks)).

---

4. Ejecución de las Pruebas

4.1 Comando general

npm test

4.2 Ejecución de un archivo específico

npm test CustomButton

4.3 Reporte de resultados

El reporte estándar de Jest incluye:

- Número total de pruebas ejecutadas.
- Casos exitosos y fallidos.
- Tiempo total de ejecución.

---

5. Pruebas por Componente

5.1 TokenManager

Archivo: src/services/__tests__/TokenManager.test.ts

Objetivo:

Verificar el almacenamiento, obtención y eliminación del token JWT en el almacenamiento local.

Casos:

1. **Guardar token:**

- Entrada: "abc123".
- Resultado esperado: token almacenado en AsyncStorage.

2. **Obtener token:**
   - Debe retornar el token almacenado.
3. **Eliminar token:**
   - El token debe ser eliminado y el resultado debe ser null.

5.2 AuthService
Archivo: src/services/__tests__/AuthService.test.ts
Objetivo:
Validar el correcto funcionamiento del inicio y cierre de sesión.
Casos:

1. **Login exitoso:**
   - Simula respuesta del servidor 200 OK.
   - Espera guardar token y retornar true.
2. **Login fallido:**
   - Simula error 401.
   - Espera lanzar excepción o retornar false.
3. **Logout:**
   - Debe eliminar token y limpiar contexto de usuario.

5.3 CustomButton
Archivo: src/components/__tests__/CustomButton.test.tsx
Objetivo:
Verificar el comportamiento del botón personalizado.
Casos:

1. **Renderizado:**
   - El texto del botón se muestra correctamente.
2. **Evento onPress:**
   - Ejecuta la función asignada al presionar.
3. **Estado disabled:**
   - No debe ejecutar la función cuando está deshabilitado.

5.4 CustomInput
Archivo: src/components/__tests__/CustomInput.test.tsx
Objetivo:
Validar el campo de entrada personalizado.
Casos:

1. **Renderizado:**
   - Muestra el placeholder correctamente.
2. **Cambio de texto:**
   - Actualiza el valor del input.
3. **Modo seguro (password):**
   - Debe ocultar el texto cuando secureTextEntry={true}.

---

6. Mantenimiento y Mejora Continua

- Revisar periódicamente las pruebas al modificar componentes.
- Usar **coverage reports** (npm test -----coverage) para identificar código sin probar.
- Implementar pruebas de regresión automatizadas en el flujo CI/CD.
- Documentar nuevos módulos y componentes probados.
- Mantener los mocks actualizados con los cambios de API.

---

7. Glosario

Término	Descripción
Unit Test	Prueba que verifica una función o módulo individual.
Mock	Objeto o función simulada que imita dependencias externas.
Jest	Framework de pruebas de JavaScript y TypeScript.
React Testing Library	Herramienta para pruebas de componentes de React Native.
Coverage	Porcentaje del código cubierto por las pruebas.
CI/CD	Integración y entrega continua, proceso de automatización en desarrollo.

---

8. Conclusión

El presente manual documenta el proceso de implementación, mantenimiento y ejecución de las **pruebas unitarias** en el proyecto **SchoolMeMovil**, estableciendo un estándar de calidad y asegurando el correcto funcionamiento del sistema antes de cada despliegue.

Grupo de Desarollo SchollMe
Jesus Fernando Carvajal jesusanacona017@gmail.com
Santiago Chaparro Riaño alfasan1481@gmail.com