Documentación Técnica - Framework de Automatización WhatsApp Claro

1. Descripción General del Proyecto

Este proyecto es un framework de automatización de pruebas para las funcionalidades del canal de WhatsApp de Claro Colombia. Utiliza Serenity BDD con el patrón Screenplay para automatizar pruebas tanto en líneas prepago como postpago, incluyendo funcionalidades como compra de paquetes, recargas, consulta de saldos, y servicios adicionales.

Características Principales

- **Framework**: Serenity BDD con Screenplay Pattern
- **Plataforma**: Automatización móvil Android con Appium
- **Gestión de Datos**: Integración con Excel y JSON para datos de prueba
- **Reportería**: Generación automática de reportes Word con capturas de pantalla
- **Arquitectura**: Modular y escalable con separación clara de responsabilidades

---

2. Arquitectura del Sistema

2.1 Estructura de Directorios

```
src/
├── main/java/
│   ├── hooks/                # Hooks para reportes y manejo de escenarios
│   ├── interactions/         # Interacciones reutilizables
│   │   ├── Click/            # Interacciones de click
│   │   ├── Validaciones/     # Validaciones de elementos
│   │   ├── comunes/          # Acciones comunes
│   │   ├── scroll/           # Interacciones de scroll
│   │   └── wait/             # Esperas y tiempos
│   ├── models/               # Modelos de datos
│   ├── questions/            # Preguntas para validaciones
│   ├── tasks/                # Tareas de negocio
│   │   ├── CompraTusPaquetes/  # Lógica de compra de paquetes
│   │   ├── CompraWhatsApp/     # Compra específica de WhatsApp
│   │   ├── HazTusRecargas/     # Funcionalidad de recargas
│   │   ├── OtrasOpciones/      # Opciones adicionales
│   │   ├── Postpago/           # Funcionalidades específicas postpago
```

```
│   │   └── TodoSobreTuLinea/   # Consultas sobre la línea
│   ├── userinterfaces/         # Page Objects con localizadores
│   └── utils/                  # Utilidades y helpers
└── test/
    ├── java/
    │   ├── runners/            # Configuración de ejecución
    │   └── stepDefinitions/    # Definiciones de pasos Cucumber
    └── resources/
        ├── config/             # Archivos de configuración
        ├── features/           # Archivos .feature de Cucumber
        └── backup/             # Respaldos de features
```

2.2 Patrones de Diseño Utilizados

Screenplay Pattern: Implementado a través de Serenity BDD para crear pruebas más legibles y mantenibles.

Page Object Model: Centralización de localizadores en la capa `userinterfaces`.

Data Provider Pattern: Gestión centralizada de datos de prueba mediante `TestDataProvider`.

Builder Pattern: Utilizado en la construcción de reportes y evidencias.

---

3. Tecnologías y Dependencias

3.1 Stack Tecnológico

gradle

```
// Framework principal
Serenity BDD: 2.0.71
Cucumber: 1.9.51
JUnit: 4.12

// Automatización móvil
Appium Java Client: 7.3.0
Selenium: 3.141.59

// Gestión de datos
Apache POI: 5.2.3 (Excel)
Jackson: Para JSON
JSON: 20210307

// Herramientas adicionales
WebDriverManager: 5.0.3
Log4j: 1.2.17
```

3.2 Configuración de Gradle

gradle

```
java {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

tasks.withType(JavaCompile) {
    options.encoding = "UTF-8"
}

test {
    systemProperties = System.properties
    maxParallelForks = Runtime.runtime.availableProcessors()
}
```

---

4. Componentes Principales

4.1 Capa de Interactions

Propósito: Encapsula acciones básicas reutilizables

```java
// Ejemplo: ClickTextoQueContengaX.java
public class ClickTextoQueContengaX implements Interaction {
    private String text;

    @Override
    public <T extends Actor> void performAs(T actor) {
        // Lógica de interacción
    }
}
```

Tipos de Interacciones:

- **Click**: Diferentes tipos de clicks (elemento, texto, coordenadas)
- **Validaciones**: Validaciones de texto, elementos visibles
- **Wait**: Esperas dinámicas y estáticas
- **Scroll**: Navegación vertical y horizontal
- **Comunes**: Acciones básicas como "Atrás"

4.2 Capa de Tasks

Propósito: Implementa lógica de negocio compleja

```java
// Ejemplo: ComprarPaquete.java
public class ComprarPaquete implements Task {
    @Override
    public <T extends Actor> void performAs(T actor) {
        actor.attemptsTo(
            SeleccionarCategoria.paquetes(),
            ElegirPaquete.porTipo(tipoPaquete),
            ConfirmarCompra.conMetodoPago(metodoPago),
            ValidarTransaccion.exitosa()
        );
    }
}
```

Categorías de Tasks:

- **CompraTusPaquetes**: Compra de diferentes tipos de paquetes

- **HazTusRecargas**: Proceso de recarga de saldo
- **TodoSobreTuLinea**: Consultas de información de línea
- **Postpago**: Funcionalidades específicas para usuarios postpago

4.3 Modelos de Datos

java

```java
// User.java
public class User {
    private String saludo;
    private String numeroPre;
    private String numeroPost;
    private String valor;

    // Getters...
}
```

4.4 User Interfaces (Localizadores)

java

```java
// WhatsAppPage.java
public class WhatsAppPage {
    public static final Target BTN_ENVIAR =
        Target.the("Botón enviar").located(By.id("send"));

    public static final Target TXT_MENSAJE =
        Target.the("Campo mensaje").located(By.id("entry"));
}
```

---

5. Gestión de Datos de Prueba

5.1 Configuración JSON

json

```
// real-user.json
{
  "saludo": "Hola",
  "numeroPre": "3558",
  "valor": "$ 50.000",
  "numeroPost": "2840"
}
```

5.2 Integración con Excel

java

```
// DataToFeature.java
public static void overrideFeatureFiles(String featuresDirectoryPath)
    throws IOException, InvalidFormatException {
    // Reemplaza datos en features con información de Excel
}
```

5.3 Provider de Datos

java

```
// TestDataProvider.java
public static User getRealUser() {
    ObjectMapper objectMapper = new ObjectMapper();
    File file = new File("src/test/resources/config/real-user.json");
    return objectMapper.readValue(file, User.class);
}
```

---

6. Sistema de Reportería

6.1 Hooks para Reportes

java

```java
// ReportHooks.java
@Before
public void beforeEachScenario() {
    EstadoPrueba.inicio = System.currentTimeMillis();
    pasosEjecutados.clear();
}

@After
public void generarReporteFinal(Scenario scenario) {
    WordAppium.generarReporte(
        scenario.getName(),
        pasosEjecutados.toArray(new String[0]),
        lineaUsada,
        duracionFormato,
        pasoFallido,
        estadoFinal
    );
}
```

6.2 Captura de Evidencias

```java
// EvidenciaUtils.java
public static void registrarCaptura(String paso) {
    String pasoNumerado = contadorPasos++ + ". " + paso;
    ReportHooks.registrarPaso(pasoNumerado);
    CapturaDePantallaMovil.tomarCapturaPantalla(pasoNumerado);
    Serenity.recordReportData().withTitle(paso).andContents(pasoNumerado);
}
```

6.3 Generación de Reportes Word

- Integración con Apache POI para generar documentos Word
- Inclusión automática de capturas de pantalla
- Registro de duración de pruebas y pasos ejecutados
- Identificación de pasos fallidos

---

7. Configuración de Ejecución

7.1 Runners de Cucumber

java

```java
@CucumberOptions(
    features = "src/test/resources/features",
    glue = {"stepDefinitions", "utils", "hooks"},
    snippets = SnippetType.CAMELCASE,
    tags = ""
)
@RunWith(CustomRunner.class)
public class GeneralRunner {
    @BeforeSuite
    public static void test() throws InvalidFormatException, IOException {
        DataToFeature.overrideFeatureFiles("src/test/resources/features");
    }
}
```

7.2 CustomRunner

java

```java
public class CustomRunner extends Runner {
    @Override
    public void run(RunNotifier notifier) {
        try {
            DataToFeature.backUpFeaturesFile();
            runAnnotatedMethods(BeforeSuite.class);
            // Ejecución de pruebas
        } finally {
            DataToFeature.restoreBackUpFeatures();
        }
    }
}
```

---

8. Funcionalidades Implementadas

8.1 Gestión de Líneas

- Selección automática de líneas prepago/postpago

- Validación de políticas de tratamiento de datos
- Consulta de información de línea

8.2 Compra de Paquetes

- **Paquetes Todo Incluido**: Datos + Voz + SMS
- **Paquetes de Voz**: Solo minutos
- **Paquetes de Datos**: Solo navegación
- **Paquetes de Apps**: WhatsApp, Instagram, Waze, YouTube

8.3 Sistema de Recargas

- Selección de valores de recarga
- Integración con diferentes medios de pago
- Validación de transacciones exitosas

8.4 Servicios Postpago

- Consulta de facturas
- Información de plan contratado
- Consumos del período actual
- Gestión de pagos

8.5 Servicios Adicionales

- Claro Video
- Claro Drive
- Claro Música
- Registro de equipos

---

9. Constantes y Configuración

9.1 Constantes de Paquetes

java

```java
// Constants_Paquetes.java
public static final String PAQ_TI_1D_2500_PRECIO = "$2.500 x 1 Día";
public static final String PAQ_TI_1D_2500_DESC = "50MB + WhatsApp + 50 minutos";
```

9.2 Constantes Generales

java

```
// Constantes.java - Mensajes del sistema
// ConstantesPost.java - Específicos para postpago
```

---

10. Mejores Prácticas Implementadas

10.1 Manejo de Errores

- Reintentos automáticos en operaciones críticas
- Manejo de mensajes de actualización del sistema
- Validaciones robustas con esperas dinámicas

10.2 Mantenibilidad

- Separación clara de responsabilidades
- Reutilización de componentes
- Documentación integrada en código

10.3 Escalabilidad

- Estructura modular para agregar nuevas funcionalidades
- Configuración centralizada
- Soporte para ejecución paralela

---

11. Configuración y Instalación

11.1 Prerrequisitos

- Java 11+
- Android SDK
- Appium Server
- Gradle 7.4+

11.2 Configuración del Proyecto

bash

```
# Clonar el repositorio
git clone [repository-url]

# Instalar dependencias
./gradlew build

# Ejecutar pruebas
./gradlew test
```

11.3 Configuración de Appium

properties

```
# test.properties
Environment=QA
appium.hub=http://localhost:4723/wd/hub
app.package=com.whatsapp
app.activity=.HomeActivity
```

---

12. Comandos de Ejecución

bash

```
# Ejecutar todas las pruebas
./gradlew test

# Ejecutar con tags específicos
./gradlew test -Dcucumber.options="--tags @smoke"

# Generar reportes Serenity
./gradlew test aggregate

# Ejecutar pruebas en paralelo
./gradlew test -Dmax.parallel.forks=4
```

---

13. Mantenimiento y Soporte

13.1 Estructura de Logs

- Logs de ejecución en `target/site/serenity/`

- Capturas de pantalla en directorio `Capturas/`
- Reportes Word generados automáticamente

13.2 Backup y Restauración

- Backup automático de archivos `.feature` antes de modificaciones
- Restauración automática post-ejecución
- Versionado de configuraciones

13.3 Integración Continua

- Compatible con Jenkins/GitHub Actions
- Reportes automáticos por email
- Integración con SonarQube para calidad de código

Autor: Equipo de QA Automatización
Fecha: Septiembre 2025
Versión: 1.0
Framework: Serenity BDD + Screenplay Pattern