

Propuesta de Arquitectura - Framework de Automatización Mobile

Autor: Juan David Coronado

1. Visión general

Definir una arquitectura escalable, mantenible y multiplataforma (Android + iOS) para pruebas automatizadas usando **Java 11+**, **Appium**, patrón **POM** y **JUnit/TestNG**. Dependencias gestionadas con **Maven/Gradle** e integración de logging con **SLF4J + Log4j2**.

2. Estructura del proyecto

La organización del proyecto sigue una estructura clara para separar responsabilidades, facilitar el mantenimiento e integrarse fácilmente con pipelines CI/CD.

```
mobile-automation-framework/  
├─ pom.xml (o build.gradle)           # Gestión de dependencias y  
configuración del build  
├─ src/  
│   └─ main/java/com.company.framework/  
│       └─ config/                     # Clases para leer y manejar  
configuraciones, factories  
│       └─ drivers/                   # DriverFactory y clases  
relacionadas con Android/iOS  
│       └─ pages/                     # Page Objects que representan  
pantallas y componentes de la app  
│       └─ services/                  # Flujos de negocio reutilizables  
(login, compra, etc.)  
│       └─ utils/                     # Funciones utilitarias (esperas,  
helpers, gestos)  
│       └─ reporting/                 # Integración con librerías de  
reportes (Allure, ExtentReports)  
│       └─ test/java/com.company.tests/  
│           └─ suites/                 # Definición de suites de pruebas  
(smoke, regression, integración)  
│           └─ tests/                  # Casos de prueba individuales  
organizados por funcionalidad  
├─ resources/                         # Archivos de configuración  
(properties, JSON, log4j2)  
├─ reports/                           # Carpeta donde se generan los  
reportes tras cada ejecución
```

└─ ci/
GitHub Actions, etc.)

Pipelines CI/CD (Jenkinsfile,

- **main** contiene el core del framework (configuración, drivers, utils, reporting).
- **test** contiene los casos de prueba y suites, separados del core.
- **resources** centraliza configuración por plataforma/entorno.
- **reports** almacena los resultados de ejecución.
- **ci/** contiene las definiciones de pipelines CI/CD para la integración continua. Aquí se configuran los pasos de build, ejecución de pruebas y generación de reportes automáticos en Jenkins, GitHub Actions o similares.

3. Patrones aplicados

- **Page Object Model (POM):** Cada pantalla se representa como una clase con sus elementos y acciones, promoviendo reutilización y orden.
- **Factory:** Crea el driver adecuado según la plataforma (Android o iOS).
- **Singleton:** Asegura una única instancia controlada del driver por ejecución, evitando conflictos en pruebas paralelas.
- **Paralelismo:** El framework permite ejecutar pruebas en paralelo, gestionando instancias independientes de drivers para evitar interferencias entre hilos.

4. Configuración multiplataforma

La configuración se maneja de manera flexible para soportar Android e iOS sin duplicar esfuerzos:

- **Archivos de propiedades y JSON:** Contienen la configuración base por plataforma (ejemplo: `android.conf.json`, `ios.conf.json`) con las *DesiredCapabilities* necesarias.
- **Combinación dinámica:** Un gestor central de configuración lee tanto los archivos base como las variables de entorno y construye la configuración final que se usará para inicializar el driver y ejecutar los tests.

Ejemplo de ejecución: - En local: `mvn clean test -Dplatform=android -Denvironment=local`
- En CI/CD: `PLATFORM=ios ENV=ci mvn clean test`

5. Logs y reportes

- **Logging:** Implementación con **SLF4J** + **Log4j2**, con salida a consola y archivo dedicado por ejecución.
- **Reportes:** Integración con **Allure** o **ExtentReports**, permitiendo adjuntar evidencias (capturas, logs, videos) y compatibilidad con pipelines CI/CD.

6. Escalabilidad y mantenibilidad

- **Arquitectura modular:** Separación clara entre core, pages, services y tests.

- **Reutilización:** Flujos de negocio encapsulados en services, evitando duplicación en tests.
 - **CI/CD:** Integración con pipelines para ejecutar suites de smoke, regression e integración automáticamente.
 - **Ejecución paralela:** Optimiza tiempos de ejecución utilizando múltiples hilos o nodos en la nube (BrowserStack, Sauce Labs).
 - **Documentación y versionamiento:** Uso de versionado semántico y convenciones claras para mantener el orden y la trazabilidad del framework.
-

7. Organización de dependencias y ejecución

- **Gestión con Maven o Gradle:** Dependencias centralizadas y actualizadas.
- **Suites de pruebas:** Agrupación en smoke, regression e integración según el nivel de validación.
- **Ejecución en paralelo:** Soporte tanto en entornos locales como en la nube, garantizando drivers independientes y aislados para cada hilo de ejecución.