Propuesta de Arquitectura - Framework de Automatización Mobile

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 y facilitar el mantenimiento:

```
mobile-automation-framework/
├── pom.xml (o build.gradle)         # Archivo principal para gestionar
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
```

- `main` contiene el core del framework (configuración, drivers, utils, reporting).
- `test` contiene los casos y suites, separados del core.
- `resources` centraliza configuración por plataforma/entorno.
- `reports` almacena los resultados de ejecución.

---

3. Patrones aplicados

- **Page Object Model (POM):** Cada pantalla se representa como una clase con sus elementos y acciones, promoviendo reutilización y orden.
- **Factory:** Se encarga de crear el driver adecuado según la plataforma (Android/iOS).
- **Strategy/Builder:** Permite armar configuraciones dinámicas según entorno y plataforma, evitando código rígido.
- **Singleton/ThreadLocal:** Garantiza una instancia de driver independiente por test, evitando conflictos en ejecuciones paralelas.

---

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**, soportando 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, integración, etc.
- **Documentación y versionamiento:** Uso de versionado semántico y convenciones claras para mantener el orden a largo plazo.

---

7. Organización de dependencias y ejecución

- **Gestión con Maven o Gradle:** Dependencias centralizadas y actualizadas.
- **Suites de pruebas:** Agrupación en smoke, regression e integración para distintos niveles de validación.
- **Ejecución en paralelo:** Soporte tanto en entornos locales como en la nube (ejemplo: BrowserStack, Sauce Labs).
