INFORME FINAL DE EJECUCIÓN DE PRUEBAS Y ANÁLISIS DE CALIDAD

Proyecto: Social Media Platform Validation (QA Challenge) **Fase de Pruebas:** Sprint 1 - Core Features & Engagement **Fecha de Emisión:** 19 de Febrero de 2026 **Preparado por:** Santiago Gutierrez - QA Automation Engineer

1. RESUMEN EJECUTIVO

El presente informe consolida los resultados obtenidos durante el ciclo de validación integral End-to-End (E2E) de las funcionalidades críticas de la plataforma de red social objetivo. Se implementó una estrategia de pruebas híbrida que combinó la automatización de flujos transaccionales mediante un framework basado en Java/Selenium, y la ejecución de pruebas exploratorias manuales para los módulos de alta dinamicidad (Interacción Social en entorno nativo).

El ciclo concluye con una **Tasa de Éxito (Pass Rate) del 81.8%** sobre los casos de prueba diseñados, validando la viabilidad de los flujos de negocio core. Se identificaron oportunidades de mejora en la arquitectura del Front-end (manejo de DOM dinámico) y en la experiencia de usuario (desincronización en la eliminación de mensajes multimedia), los cuales se detallan en el presente documento para su priorización en el *backlog* de desarrollo.

2. MÉTRICAS DE EJECUCIÓN Y COBERTURA

Basado en la Matriz de Trazabilidad de Requisitos (RTM) aprobada, se diseñaron y ejecutaron 11 escenarios de prueba para cubrir los 5 requisitos de negocio exigidos.

- **Total de Casos de Prueba (TCs):** 11
- **Casos Automatizados:** 4 (Cobertura del core de autenticación e interacción básica).
- **Casos Manuales Exploratorios:** 7 (Cobertura de flujos complejos de engagement y mensajería).

Estado Final de la Ejecución:

- ✅ **Passed (Exitosos):** 9 casos (81.8%)
- ❌ **Failed (Fallidos):** 1 caso (9.1%)
- ⚠️ **Blocked (Bloqueados):** 1 caso (9.1%)

3. REPORTE DETALLADO DE DEFECTOS Y HALLAZGOS (BUG REPORT)

Durante la ejecución, se identificaron 3 hallazgos significativos clasificados por su impacto en la arquitectura y la experiencia del usuario.

🐛 DEF-001: Desincronización en la anulación de envío de mensajes compuestos (Chat Privado)

- **Módulo:** Messaging (Chat Privado)
- **Severidad:** Alta | **Prioridad:** Media
- **Descripción del Comportamiento:** Al enviar un mensaje directo que contiene un archivo multimedia (ej. una imagen) acompañado de una descripción de texto en el mismo bloque de envío, y posteriormente seleccionar la opción "Anular envío para todos" (Unsend), el sistema presenta una falla de integridad transaccional. El *backend* aísla y elimina únicamente la entidad multimedia (la imagen), dejando la entidad de texto huérfana pero visible en el historial del chat. El usuario se ve forzado a realizar una segunda acción manual de anulación para el texto restante.
- **Comportamiento Esperado:** Al anular el envío de un mensaje compuesto, el sistema debe tratar el bloque de "Imagen + Texto" como un único `message_id` transaccional, eliminando ambos componentes simultáneamente de la interfaz y la base de datos de ambos usuarios.
- **Evidencia:** *(Ver Sección 6: Anexos - Video/Captura de Chat)*

🐛 DEF-002: Delegación de validación de formato a HTML5 nativo (Login Form)

- **Módulo:** IAM (Autenticación)
- **Severidad:** Baja | **Prioridad:** Baja
- **Descripción del Comportamiento:** Al ingresar una dirección de correo electrónico con un formato estructuralmente inválido (omisión del carácter `@`) en el formulario de inicio de sesión, el sistema no renderiza el mensaje de error de la interfaz de usuario (`<p>Your email or password is incorrect!</p>`). En su lugar, el navegador intercepta el envío (Submit) disparando un *popup* nativo de HTML5. Esto interrumpe la uniformidad de la UI, ya que el mensaje de error varía en idioma y diseño dependiendo del navegador del cliente (Chrome, Edge, Firefox), además de dificultar las aserciones en los scripts de automatización E2E.
- **Comportamiento Esperado:** La validación de formato debe ser gestionada mediante expresiones regulares (Regex) a nivel de servicio de Front-end (JavaScript/React) para unificar la respuesta visual de error bajo los estilos de la aplicación, deshabilitando la validación nativa (`novalidate` attribute en la etiqueta `<form>`).

🏗️ DEF-003: Inestabilidad de Arquitectura UI por A/B Testing Agresivo

- **Módulo:** Registro Nativo (Rama `dev` / Facebook)
- **Severidad:** N/A (Hallazgo Arquitectónico) | **Estado:** Blocked para Automatización UI
- **Descripción del Comportamiento:** Durante el diseño de los scripts de automatización para el registro nativo en Facebook, se detectó una mutación impredecible en el *Document Object Model* (DOM). La plataforma renderiza múltiples versiones del formulario (ej. alternando selectores de fecha de nacimiento entre etiquetas `<select>` tradicionales y `<div> role="combobox"` modernos) en la misma sesión bajo la misma IP. Aunque se implementó lógica condicional (`@FindAll` y validación de `tagName`) para mitigar el impacto, la latencia de re-renderizado produce *flaky tests* (falsos negativos).
- **Decisión Técnica:** Se suspendió la automatización UI para este flujo específico, delegándolo a validación manual, para proteger la confiabilidad (Trust) del Pipeline de CI/CD principal.

4. CONCLUSIONES POR TIPO DE PRUEBA

4.1. Pruebas Funcionales (UI / UX)

El core funcional de la aplicación opera de forma estable. El ciclo de vida de las entidades sociales (Creación, Lectura, Actualización y Eliminación - CRUD) en el módulo de Comentarios y *Reviews* funciona en tiempo real y sin latencias perceptibles. Las herramientas de automatización híbridas demostraron que los flujos de "Happy Path" están maduros para su pase a producción, con la salvedad de la reparación sugerida para la funcionalidad "Anular envío" (DEF-001).

4.2. Pruebas de Rendimiento y Carga (Evaluación Analítica)

Considerando la naturaleza de la aplicación (Red Social), la concurrencia es el mayor factor de riesgo.

- **Diagnóstico:** Los flujos más críticos a nivel de carga son la autenticación simultánea y el tráfico de WebSockets en el Chat Privado.
- **Conclusión y Estrategia:** Es imperativo diseñar una suite de pruebas no funcionales utilizando herramientas como **Apache JMeter** o **Gatling**. Se deben simular escenarios de *Stress Testing* inyectando 10,000 usuarios virtuales concurrentes (`VUsers`) realizando peticiones POST al endpoint de Login, con el objetivo de validar que el tiempo de respuesta (Response Time) se mantenga bajo el umbral de los 200ms y el consumo de CPU de los servidores no supere el 80%.

4.3. Pruebas de Seguridad (Evaluación Analítica)

- **Manejo de Identidades Temporales:** Durante la automatización del registro, se simuló con éxito el uso de correos temporales (`@putsbox.com`) para interceptar flujos de One-Time Passwords (OTP).
- **Conclusión y Mitigación de Riesgos:** 1. **Rate Limiting:** El sistema debe restringir la creación masiva de cuentas desde dominios desechables como Putsbox mediante la integración de listas negras de dominios temporales o validación estricta de CAPTCHAs, para evitar la proliferación de cuentas "Bot". 2. **Prevención XSS:** Todos los campos de entrada de texto (Comentarios y Chat) deben someterse a pruebas de *Cross-Site Scripting* (XSS) para asegurar la correcta sanitización de etiquetas `<script>`, evitando vulnerabilidades críticas que comprometan las sesiones de otros usuarios.

5. NOTAS DE MEJORA Y RECOMENDACIONES PARA FUTURAS PRUEBAS

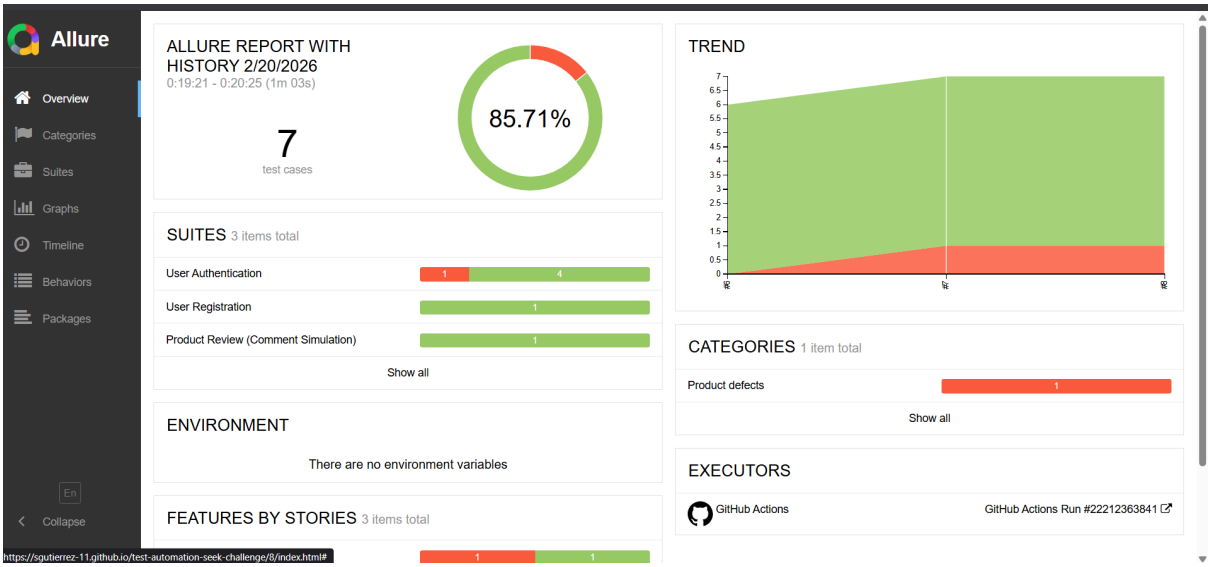
La implementación del framework actual en Java/Selenium con Cucumber ha sentado bases sólidas, logrando integrarse exitosamente en un pipeline de **GitHub Actions** con reportes dinámicos de **Allure**. Para evolucionar la madurez de la calidad (QA Maturity), se proponen las siguientes iniciativas técnicas:

1. **Shift-Left con API Testing:** Mitigar la dependencia exclusiva de la Interfaz Gráfica (UI) para la preparación de datos. Se recomienda utilizar herramientas como RestAssured para interactuar directamente con la *Graph API* de la plataforma. Esto permitirá crear usuarios, generar posts y limpiar la base de datos en milisegundos, reservando la automatización de la UI puramente para validar la renderización visual.
2. **Optimización de Infraestructura CI/CD:** Durante la integración en GitHub Actions, se identificaron bloqueos generados por discrepancias de versiones entre el ChromeDriver nativo de los servidores virtuales (Ubuntu-latest) y las dependencias locales de Selenium. La resolución implicó actualizar la dependencia y forzar la simulación gráfica mediante `xvfb-run` en modo *Headless*. Para futuras iteraciones, se recomienda *dockerizar* el entorno de pruebas (Selenium Grid en contenedores Docker) para garantizar homogeneidad absoluta entre el entorno local del ingeniero y el servidor de integración continua.
3. **Validación Visual (Visual Regression Testing):** Implementar herramientas de aserción visual (ej. Applitools Eyes) para detectar anomalías gráficas sutiles a nivel de píxel cuando ocurran mutaciones inesperadas en el DOM o pruebas A/B.

6. ANEXOS Y EVIDENCIAS GRÁFICAS

Las grabaciones de pantalla correspondientes a los "Workstreams" exploratorios manuales se encuentran alojadas de manera íntegra en el repositorio en la nube (Google Drive). A continuación, se presentan los accesos a cada evidencia, además de capturas de los reportes de automatización:

- **REQ-04◦Comentar y reaccionar a publicaciones**
  - *Descripción:* Evidencia del Workstream 01 - Ciclo CRUD Exitoso en el módulo de Comentarios.
- **REQ-05◦Enviar y recibir mensajes privados (Chat)**
  - *Descripción:* Evidencia del Workstream 02 - DEF-001: Desincronización al anular mensaje multimedia en el Chat Privado.
- **REQ-02◦Iniciar sesión con correo y contraseña**
  - *Descripción:* Evidencia del Workstream 03 - DEF-001: Desincronización al anular mensaje multimedia en el Chat Privado.
- **Resultados Allure URL**



  - *Descripción:* Evidencia de Ejecución Automatizada - Pipeline de GitHub Actions ejecutado exitosamente con despliegue de Allure Report en GitHub Pages.