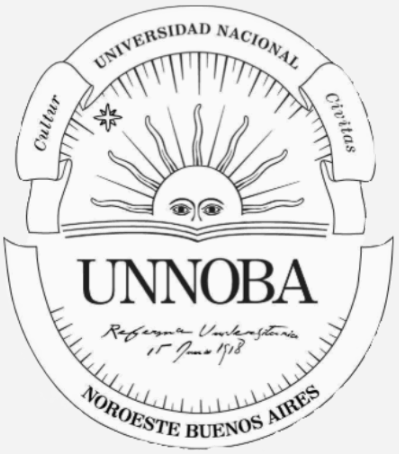
Universidad Nacional del Noroeste de la Provincia de Buenos Aires



INFORME FINAL DE PRÁCTICA PROFESIONAL SUPERVISADA

Estudiante: Luis Francisco Martínez

Carrera: Ingeniería en Informática

Organización: Edit Software

Tutor Docente: Paula Lucrecia Lencina

Tutor de Empresa: Diego de la Riva

Fecha de presentación: Marzo 2026

Índice

1. Introducción

La presente Práctica Profesional Supervisada (PPS) se desarrolló en el seno de la organización Edit Software, una empresa dedicada al desarrollo de soluciones tecnológicas integrales para el sector salud. La actividad principal de la organización reside en el mantenimiento y evolución de un sistema ERP (Enterprise Resource Planning) Médico, cuya arquitectura monolítica basada en Java EE gestiona procesos críticos como la administración de pacientes, gestión de turnos, historias clínicas y facturación a obras sociales. [1]

Dada la naturaleza sensible de los datos médicos y necesaria disponibilidad del servicio, la calidad del software es un requisito no funcional imperativo. La finalidad del trabajo consistió en la ingeniería e implementación de una estrategia de Aseguramiento de Calidad (QA) Automatizado. Previo a esta intervención, el ciclo de desarrollo de software enfrentaba cuellos de botella significativos debido a la dependencia exclusiva de pruebas manuales, lo que elevaba el riesgo de regresión en despliegues productivos y limitaba la frecuencia de las entregas.

El proyecto abordó el diseño de un framework de automatización a medida, capaz de interactuar con desafíos técnicos complejos inherentes a la tecnología PrimeFaces [2] (renderizado dinámico y asincronía). El resultado buscado fue transicionar de un modelo reactivo a uno preventivo, integrando validaciones automáticas que aseguren la estabilidad operativa del sistema ante cambios continuos en el código fuente, garantizando así la integridad de la información sensible de los pacientes.

2. Objetivos

Los objetivos se fundamentan en la necesidad de mitigar la deuda técnica asociada al proceso de validación manual y reducir el Time-to-Market de las nuevas funcionalidades sin sacrificar la robustez del sistema.

Objetivo General:

Desarrollar e implementar un ecosistema integral de Quality Assurance (QA) que automatice las pruebas de regresión, funcionales y de integración del ERP Médico, utilizando un stack tecnológico moderno para garantizar la integridad de los datos, la disponibilidad del servicio y la optimización de los recursos de desarrollo.

Objetivos Específicos:

- ➢ **Analizar** la arquitectura del sistema legado (JSF/PrimeFaces) para identificar los desafíos de automatización relacionados con el ciclo de vida de las peticiones AJAX, la gestión de estados de vista (ViewState) y la seguridad basada en sesiones HTTP.
- ➢ **Diseñar** una arquitectura de pruebas híbrida que combine la interacción de interfaz de usuario (UI) para flujos de usuario final con la manipulación directa de API para la gestión eficiente y atómica de datos de prueba masivos.
- ➢ **Implementar** un framework de automatización en Java (JDK 17) [3] utilizando patrones de diseño avanzados (Page Object Model, Singleton, Factory) y algoritmos de sincronización inteligente para eliminar la fragilidad (flakiness) de los tests en entornos asíncronos.
- ➢ **Validar** cuantitativamente el impacto de la solución mediante la medición de indicadores de rendimiento, contrastando tiempos de ejecución manual versus automatizada para demostrar el Retorno de Inversión (ROI) y la aceleración del ciclo de feedback.

---

3. Plan de trabajo y Carga horaria

Las actividades se estructuraron bajo una metodología incremental dividida en cinco fases, abarcando un total de 10 semanas de trabajo intensivo. Este cronograma incluyó instancias periódicas de reuniones de avance y seguimiento con los tutores para validar el progreso y ajustar el rumbo técnico del proyecto. La planificación detallada se adjunta en el **Anexo I: Diagrama de Gantt.**

Resumen de Fases Ejecutadas:

- ➢ **Fase 0 (Inducción e Investigación):** Selección del stack tecnológico mediante matriz de decisión y configuración del entorno de desarrollo (IDE IntelliJ, Repositorios Git,

Maven).

- **Fase 1 (Arquitectura):** Análisis de ingeniería inversa sobre el comportamiento del protocolo HTTP en aplicaciones JSF y diseño de la solución "Híbrida" (Bridge UI-API).

- **Fase 2 (Diseño de Pruebas):** Definición de la matriz de cobertura, escritura de casos de prueba (Smoke y Regresión) y definición de la estrategia de "Limpieza de Datos".

- **Fase 3 (Implementación):** Codificación del núcleo del framework, desarrollo de utilidades de sincronización (JsfUtil) y scripting de las suites de prueba.

- **Fase 4 (Transferencia y Métricas):** Ejecución final de validación, recolección de métricas de desempeño, redacción de manuales técnicos y transferencia de conocimiento al equipo de desarrollo.

Dinámica de Seguimiento y Reuniones: El desarrollo de la práctica estuvo acompañado por un esquema de seguimiento activo para validar el progreso donde se establecieron encuentros fijos dos veces por semana para reportar estado de avance y gestionar impedimentos técnicos. Además, como extra se realizaron reuniones espontáneas al finalizar cada etapa crítica del cronograma para la demostración práctica de los entregables y la validación de los tutores antes de avanzar a la siguiente fase.

Desvíos y Ajustes al Plan Original:

Durante la Fase 3, se detectó un desvío técnico no previsto relacionado con la latencia en la renderización del DOM por parte de la librería PrimeFaces. Las estrategias de espera estándar (Implicit Waits) resultaron insuficientes, provocando fallos aleatorios en los scripts ("Falsos Positivos").

Para mitigar este riesgo sin impactar la fecha de entrega final, se reasignaron horas destinadas originalmente a "Pruebas de Reportes" hacia la investigación y desarrollo de una solución de Esperas Explícitas basadas en JavaScript (JsfUtil). Esta reingeniería permitió estabilizar el framework, absorbiendo el desvío dentro del margen de contingencia planificado.

Adicionalmente, en la etapa final de integración (Fase 4), surgió un desafío crítico relacionado con la inmutabilidad de los identificadores en JSF. Se detectó que la estrategia inicial de inyección de datos vía API fallaba silenciosamente (efecto conocido como 'Click Fantasma') debido a que el servidor regeneraba los IDs de los componentes (javax.faces.source) en cada renderizado. Esto obligó a realizar una refactorización arquitectónica no planificada de la capa de Servicios (PacienteService), evolucionando de un modelo de inyección estática a uno dinámico que requiere un 'Scraping en Tiempo Real' previo a cada petición API. Este ajuste consumió horas de trabajo no previstas en su momento, las cuales fueron recuperadas optimizando la fase de documentación.

---

4. Marco Teórico y Fundamentos de la Calidad

La ingeniería de software contemporánea ha evolucionado para establecer que la calidad no debe ser una fase aislada de verificación al final del desarrollo, sino un proceso continuo e integrado. El presente trabajo se sustenta en un enfoque preventivo alineado con el paradigma de Shift-Left Testing, el cual propone desplazar las actividades de validación hacia las etapas iniciales del ciclo de vida.

Esta metodología busca mitigar los riesgos inherentes al modelo tradicional en cascada, donde la detección tardía de defectos eleva exponencialmente los costos de corrección [4]. Aunque la arquitectura heredada del sistema ERP limitó la aplicación purista de metodologías como TDD (Test Driven Development), la estrategia implementada simula sus beneficios al reducir drásticamente el tiempo entre la introducción de un defecto y su detección.

El diseño de la solución técnica se fundamentó en la teoría de la "Pirámide de Automatización" propuesta por Mike Cohn, que sugiere priorizar una base sólida de pruebas unitarias y de integración sobre las pruebas de interfaz de usuario (UI), debido a la fragilidad y costo de mantenimiento de estas últimas [5].

El diagnóstico inicial de la organización reveló la presencia del anti-patrón opuesto, conocido como el "Cono de Helado", caracterizado por una dependencia excesiva de pruebas manuales y escasa cobertura automatizada. Para revertir esta tendencia sin detener el flujo productivo, se desarrolló una arquitectura híbrida que combina la robustez y velocidad de las peticiones a

nivel de servicios (API) con la validación visual estrictamente necesaria para las interfaces JSF.

En el contexto específico de sistemas legados (legacy) y arquitecturas monolíticas, la literatura especializada identifica a las Pruebas de Regresión como el mecanismo de defensa crítico. Dado el alto acoplamiento entre componentes en aplicaciones Java EE antiguas, el riesgo de efectos colaterales indeseados ante modificaciones es elevado.

Por tanto, la automatización desarrollada no tiene como único fin probar nuevas funcionalidades, sino actuar como una red de seguridad (Safety Net) que garantiza la estabilidad operativa de los procesos de negocio críticos ante la evolución constante del código fuente [6].

Finalmente, la estrategia cobra valor operativo mediante la adopción de prácticas de Integración Continua (CI). La implementación del pipeline automatizado materializa el principio de Fail Fast (Fallar Rápido), asegurando que el feedback sobre la calidad del software sea inmediato [7].
Para garantizar la sostenibilidad de este ecosistema a largo plazo, el desarrollo del framework se rigió por patrones de diseño estructurales como el Page Object Model (POM). Este enfoque teórico permite desacoplar la lógica de prueba de la estructura del DOM, tratando al código de prueba (testware) como un activo de software de primera clase que requiere los mismos estándares de mantenibilidad y limpieza que el código de producción [8].

---

5. Descripción de la Práctica Profesional Efectuada

El núcleo de la Práctica Profesional Supervisada consistió en la construcción de un entorno de software complejo: un Framework de Automatización de Pruebas. A continuación, se detalla el desarrollo técnico del proyecto, desde la selección tecnológica hasta la implementación de algoritmos específicos.

5.1. Análisis del Contexto Tecnológico y Selección del Stack

El sistema bajo prueba presenta una arquitectura monolítica basada en Java Server Faces (JSF). Esta tecnología impone desafíos particulares para la automatización:

IDs Dinámicos: Los identificadores de los elementos HTML son autogenerados y mutables.

Ciclo de Vida AJAX: Las actualizaciones parciales de la página no disparan eventos de carga estándar del navegador (**window.onload**), lo que confunde a los drivers de automatización tradicionales.

Para abordar esto, se ha realizado una investigación técnica, seleccionando el siguiente stack:

Lenguaje: Java (JDK 17).
Se ha priorizado la homogeneidad con el backend del ERP para facilitar la adopción por parte del equipo de desarrollo (White-box testing potential) y reutilizar modelos de datos.

Motor de UI: Selenide. [9]
Seleccionado por sobre Selenium Vanilla debido a su capacidad de manejo automático de StaleElementExceptions, capturas de pantalla automáticas y su sintaxis fluida.

Motor de API: RestAssured. [10]
Utilizado para la inyección de datos y validaciones de backend, permitiendo manipular cookies y headers complejos.

Orquestador: JUnit 5.[11]
Para la gestión del ciclo de vida de los tests, inyección de dependencias y generación de reportes vía Allure.[12]

5.2. Arquitectura de la Solución: El Patrón "Hybrid Session Hijacking"

Uno de los aportes más significativos de la práctica fue el diseño de una arquitectura híbrida. Las pruebas de UI son tradicionalmente lentas (simular un alta de paciente toma ~45 segundos). Para optimizar esto, se ha diseñado un mecanismo de **Session Hijacking** (Secuestro de Sesión Controlado).

Funcionamiento del Mecanismo:

1. El test inicia sesión en el navegador (UI) simulando un usuario real.

2. El framework captura la cookie **JSESSIONID** del navegador y extrae el token de seguridad **javax.faces.ViewState** del DOM.

3. Estas credenciales se inyectan en el cliente HTTP (**RestAssured**).

4. El sistema permite entonces realizar operaciones pesadas (como crear 50 pacientes o limpiar la base de datos) vía API en segundos, compartiendo la sesión del usuario logueado en el navegador.

Este enfoque redujo el tiempo de pre-condición de los tests significativamente, permitiendo que la automatización se centre en validar la lógica de negocio y no pierda tiempo en la carga de datos.

5.3. Implementación Técnica y Manejo de Asincronía

La mayor barrera técnica encontrada fue la sincronización con *PrimeFaces*. Para resolver la inestabilidad de los tests (flakiness), se ha desarrollado una librería de utilidades personalizada (**JsfUtil**).

En lugar de utilizar **Thread.sleep()** (una mala práctica que ralentiza la ejecución), se ha implementado un algoritmo de **Esperas Inteligentes**. Este algoritmo inyecta código JavaScript en el navegador en tiempo de ejecución para consultar la cola de eventos de la librería gráfica.

Lógica del algoritmo implementado:

> "Mientras **PrimeFaces.ajax.Queue.isEmpty()** sea falso, el test debe esperar."

Esta implementación garantiza que el test solo interactúe con la interfaz cuando la aplicación ha terminado completamente de procesar los datos, eliminando virtualmente los errores por "Elemento no interactuable".

5.4. Diseño de Casos de Prueba y Cobertura

Se ha aplicado el patrón de diseño **Page Object Model (POM)** [8] para desacoplar la lógica de prueba de la estructura HTML. Esto significa que si la interfaz gráfica cambia (ej: se

renombra un botón), solo es necesario actualizar la clase de la "Página", sin tocar los cientos de tests que la utilizan.

Se han definido y automatizado dos suites principales cuyo detalle de cobertura se presenta en el **Anexo II: Diseño de Casos de Prueba**

1. **Smoke Suite:** Ejecución crítica post-despliegue. Verifica Login y las operaciones de creación de Pacientes y Turnos.
2. **Regression Suite:** Cobertura exhaustiva de los módulos de Pacientes y Turnos. Incluye:
    - Ciclo de vida de Pacientes (CRUD).
    - Asignación y cancelación de Turnos.
    - Validaciones de seguridad y borrado lógico/físico.

5.5. Gestión de Datos (Test Data Management)

Dada la naturaleza sensible de los datos médicos, se ha implementado una estrategia de Datos Sintéticos Efímeros. Se ha utilizado el patrón Factory para generar pacientes con DNIs y Emails únicos basados en marcas de tiempo (System.currentTimeMillis()), asegurando la atomicidad de los tests (ningún test falla porque el anterior dejó datos sucios o duplicados). Además, se han implementado rutinas de limpieza (Teardown) estrictas mediante bloques finally que eliminan los datos creados al finalizar cada ejecución, manteniendo la base de datos limpia ("Leave no trace").

5.6. Estrategia de CI/CD y Arquitectura de Repositorios

Para garantizar la calidad del código de prueba, se ha adoptado un flujo de trabajo basado en Gitflow. Las nuevas funcionalidades del framework se desarrollaban en ramas específicas (feature branches) y se integraban a la rama principal (main) únicamente a través de Merge Requests (MR) en GitLab.
Este proceso incluía revisiones de código (Code Reviews) obligatorias por parte del Tutor de Empresa, asegurando que el código cumpliera con los estándares de nomenclatura y patrones de diseño definidos antes de ser fusionado. Esto evitó la introducción de "código sucio" o redundante en el repositorio de pruebas.

Como parte de la modernización del ciclo de desarrollo, se ha diseñado una estrategia de integración continua para ejecutar las pruebas automatizadas de manera segura y eficiente, considerando las restricciones de infraestructura (Firewall/WAF) de la organización.

Arquitectura de Repositorios Desacoplados: Se ha optado por mantener el código de automatización (empresa-group/test) en un repositorio separado del código fuente de la aplicación. Esta decisión, alineada con el modelo "Pipeline Multi-Proyecto", evita bloquear el flujo de trabajo de los desarrolladores en caso de fallos en la infraestructura de pruebas.

Infraestructura de Ejecución (GitLab Runner Self-Hosted): Debido al bloqueo de las IPs de GitLab Cloud por parte del Firewall corporativo (Error 403 Forbidden), se ha implementado un Runner Local. Esta solución permite ejecutar las pruebas contra los entornos protegidos (Stage/Test) sin comprometer las políticas de seguridad de la red.

Estrategia de Orquestación (Ejecución Programada): Dado el uso de un Runner Local, se ha configurado una estrategia de ejecución basada en Schedules (Cron Jobs) dentro de GitLab CI. [13]

- ➢ Nightly Builds: Se ha programado la ejecución automática de la suite de Regresión y Smoke completa durante horarios nocturnos.
- ➢ On-Demand: Se ha habilitado la posibilidad de ejecución manual mediante la interfaz de GitLab para validaciones puntuales.

Esta implementación garantiza que cada nueva versión del software sea validada automáticamente, eliminando la dependencia de la ejecución manual en la máquina del analista.

5.7. Análisis de Resultados y Retorno de Inversión (ROI)

Para evaluar el éxito del proyecto, se han medido los tiempos de ejecución comparativos entre las pruebas manuales realizadas por un analista QA y la ejecución desatendida del nuevo framework. Los resultados obtenidos en la fase final fueron contundentes:

Tipo de Suite	Cantidad de Tests	Ejecución Manual (Promedio)	Ejecución Automatizada	Ahorro de Tiempo	Factor de Aceleración
Smoke Test	3	5 min	2 min 25 seg	2 min 35 seg	~ 2x más rápido
Regresión	10	15 min	8 min 30 seg	6 min 30 seg	~ 2x más rápido
Ciclo Completo	13	22 min	12 min	10 min	~ 2x más rápido

Tabla 1. Análisis comparativo de tiempos de ejecución: Pruebas Manuales y Automatizadas.

El análisis de los indicadores presentados en la **Tabla 1** revela que la implementación del framework no solo representa una mejora en los tiempos de respuesta, sino un cambio paradigmático en la eficiencia operativa de Edit Software. Al alcanzar un factor de aceleración de aproximadamente 2x en todas las suites, se evidencia que la automatización ha logrado reducir el costo de ejecución a niveles que permiten una frecuencia de validación antes inviable bajo métodos manuales. Esta optimización se traduce en un ahorro de 10 minutos por cada ciclo completo, lo que libera recursos para tareas de análisis de mayor valor y minimiza los costos de corrección al detectar defectos de forma temprana. La validez técnica de estos resultados y la estabilidad del sistema bajo estas nuevas condiciones de carga se encuentran respaldadas por la evidencia visual y las métricas de salud del build detalladas en el **Anexo III: Reportes de Ejecución**, donde se certifica una tasa de éxito del 100% en los escenarios automatizados. En última instancia, esta transición hacia una ejecución desatendida garantiza una red de seguridad constante que preserva la integridad de los datos críticos del ERP frente a cada cambio en el código fuente.

---

6. Conclusiones

La finalización de la Práctica Profesional Supervisada marcó un hito en mi madurez profesional como QA y en el proceso de ingeniería de software de Edit Software. El trabajo

realizado trascendió la mera escritura de scripts de prueba, constituyéndose en una reingeniería del proceso de calidad.

Logros Alcanzados:

Estabilidad: Se logró automatizar el 100% de los flujos críticos definidos en el alcance. La implementación de JsfUtil eliminó la fragilidad típica de las pruebas de UI, entregando resultados confiables sin falsos positivos.

Integración Continua: Se desplegó exitosamente un pipeline de ejecución automatizada utilizando Runners locales, superando las barreras de seguridad de la infraestructura y estableciendo un ciclo de validación diaria.

Eficiencia (ROI): Las métricas finales demostraron una reducción superior al 50% en los tiempos de ejecución de regresión. Esto libera recursos humanos valiosos de tareas repetitivas, permitiéndoles enfocarse en pruebas de mayor valor.

Transferencia Tecnológica: Se entregó un framework modular, documentado y basado en estándares de industria (Maven, JUnit), lo que garantiza su mantenibilidad futura por parte del equipo de desarrollo. Para facilitar esta adopción, se adjunta la guía completa en el **Anexo IV: Manual de Instalación y Transferencia**.

Lecciones Aprendidas:

Durante el desarrollo se comprendió la importancia de la Ingeniería Inversa del Protocolo HTTP para automatizar sistemas legados. Entender cómo viajan los datos "bajo el capó" (cookies, headers, viewstates) fue crucial para superar las limitaciones de la interfaz gráfica. Asimismo, se validó que en aplicaciones modernas asíncronas, el manejo explícito del tiempo y las esperas dinámicas son el factor determinante para el éxito de la automatización.
Más allá del desafío de ingeniería, la práctica también requirió un fuerte componente de adaptación y comunicación. Se aprendió que el éxito de una herramienta de QA no radica solo en su robustez de código, sino en la facilidad con la que el equipo puede adoptarla e integrarla en su flujo de trabajo diario sin percibirla como una fricción.

Trabajos Futuros (Roadmap):

Para continuar la evolución del área de QA, se recomienda:

Orquestación Avanzada: Implementar Triggers (Disparadores) desde el repositorio de desarrollo para ejecutar las pruebas automáticamente tras cada despliegue exitoso (no solo por horario). [14]

Containerización: Empaquetar el entorno de pruebas en una imagen Docker para garantizar la paridad entre los entornos de desarrollo, testing y producción.

Ejecución Paralela: Configurar JUnit 5 para correr tests en paralelo, lo que podría reducir el tiempo de regresión de 8 a 3 minutos.

---

7. Bibliografía

[1] Edit Software, "Especificación Funcional del ERP Médico y Documentación de Arquitectura JSF," Documentación Interna, 2026.

[2] PrimeTek, "PrimeFaces Documentation & User Guide," [En línea]. Disponible: https://avalon.primefaces.org/documentation.xhtml;jsessionid=node0g6q8558wmjjvsedtjx5vrvx16328994.node0 . [Accedido: Enero. 2026].

[3] Oracle Corporation, "Java SE 17 Developer Documentation," [En línea]. Disponible: https://docs.oracle.com/en/java/javase/17/ . [Accedido: Enero. 2026].

[4] L. Crispin y J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley Professional, 2009.

[5] M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley Professional, 2009.

[6] I. Sommerville, Software Engineering, 10th ed., Pearson, 2015.

[7] P. M. Duvall, S. Matyas, y A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, 2007.

[8] M. Fowler, "PageObject Pattern," MartinFowler.com, 2013. [En línea]. Disponible: https://martinfowler.com/bliki/PageObject.html .

[9] Selenium Project, "Selenide: Concise UI Tests in Java," [En línea]. Disponible: https://selenide.org/ . [Accedido: Enero. 2026].

[10] REST-Assured, "Testing and validating REST services in Java," [En línea]. Disponible: https://rest-assured.io/ . [Accedido: Enero. 2026].

[11] JUnit Team, "JUnit 5 User Guide," [En línea]. Disponible: https://junit.org/junit5/docs/current/user-guide/ . [Accedido: Enero. 2026].

[12] Qameta Software, "Allure Report Documentation," [En línea]. Disponible: https://allurereport.org/docs/ . [Accedido: Enero. 2026].

[13] GitLab, "GitLab CI/CD Documentation," [En línea]. Disponible: https://docs.gitlab.com/ee/ci/ . [Accedido: Enero. 2026].

[14] J. Ferguson Smart, Jenkins: The Definitive Guide, O'Reilly Media, 2011.

---

8. Anexos

En la presente sección se adjunta la documentación técnica, diagramas y evidencias de ejecución que complementan el desarrollo descriptivo del informe. Estos materiales tienen como finalidad demostrar la trazabilidad entre la planificación inicial, el diseño de la solución y los resultados obtenidos.

Anexo I : Diagrama de Gantt y Plan de Trabajo Detallado

Anexo II : Diseño de Casos de Prueba

Anexo III : Reportes de Ejecución

Anexo IV : Manual de Instalación y Transferencia.

Anexo I: Diagrama de Gantt y Plan de Trabajo Detallado

Cronograma de Ejecución (10 Semanas)

Fases del Proyecto	Sem 1	Sem 2	Sem 3	Sem 4	Sem 5	Sem 6	Sem 7	Sem 8	Sem 9	Sem 10
Fase 0: Inducción e Investigación	■									
Config. Entorno y Selección Stack	■									
Fase 1: Análisis de Arquitectura		■								
Ingeniería Inversa JSF y Diseño Híbrido		■								
Fase 2: Diseño del Plan de QA			■	■						
Escritura de Casos (Smoke/Regresión)			■	■						
Fase 3: Implementación					■	■	■	■		

Codificación Framework y Scripts					■	■	■	■		
Resolución Desafío PrimeFaces (JsfUtil)							■	■		
Fase 4: Documentación y Cierre									■	■
Métricas, Manuales y Transferencia									■	■

Tabla 2. Cronograma de ejecución de actividades (Diagrama de Gantt).

Detalle de Actividades por Fase:

Fase 0 (Semana 1): Configuración de IntelliJ, Maven y Git. Investigación comparativa de Selenide vs Selenium.

Fase 1 (Semana 2): Análisis de tráfico HTTP en login y formularios JSF. Definición de arquitectura de pruebas y patrón *Session Hijacking*.

Fase 2 (Semanas 3-4): Redacción de casos de prueba en Gherkin/Excel. Definición de la estrategia de datos sintéticos.

Fase 3 (Semanas 5-8): Desarrollo iterativo del software.

Sem 5-6: Estructura base, configuración de **RestAssured** y pruebas de API.

Sem 7-8: Pruebas de UI, implementación de **JsfUtil** para resolver el asincronismo de PrimeFaces y estabilización de la suite.

Fase 4 (Semanas 9-10): Ejecución final de la regresión, generación de reportes Allure, redacción de documentación técnica y entrega del informe final.

Anexo II: Diseño de Casos de Prueba (Cobertura).

Convenciones:

Prioridad: Alta (Smoke/Core), Media (Regresión), Baja (Validaciones menores).

ID	Módulo	Título / Objetivo	Pre-condiciones	Datos de Entrada	Pasos de Prueba	Resultado Esperado	Funcionó Correctamente	Prioridad
CP_001	Login	**Login Exitoso** Verificar acceso con credenciales válidas.	Base de datos online.	User: admin Pass: secret	1. Ir a /login. 2. Ingresar User/Pass. 3. Click "Ingresar".	Redirección a /dashboard. Menú principal visible.	✅	Alta
CP_002	Login	**Login Fallido** Verificar mensaje de error.	N/A	User: admin Pass: wrong	1. Ir a /login. 2. Ingresar credenciales inválidas. 3. Click "Ingresar".	Mensaje "Credenciales inválidas" visible. URL se mantiene.	✅	Media

ID	Módulo	Título / Objetivo	Pre-condiciones	Datos de Entrada	Pasos de Prueba	Resultado Esperado	✅	Prioridad
CP_003	Pacientes	**Alta Paciente** Crear un paciente con datos mínimos obligatorios.	Logueado como Admin.	DNI: Único Nombre: "Test Auto" Obra Social: "OSDE"	1. Ir a Pacientes > Nuevo. 2. Completar formulario. 3. Click "Guardar". 4. Filtrar grilla por nombre.	Mensaje "Guardado correctamente". El paciente aparece en la grilla.	✅	**Alta**
CP_004	Pacientes	**Edición de Paciente**	Paciente existente.	Nuevo Nombre: "Editado"	1. Buscar Paciente. 2. Click Icono "Lápiz". 3. Cambiar nombre. 4. Click "Actualizar"	Mensaje de éxito. El nombre nuevo se refleja en la grilla.	✅	Media
CP_005	Pacientes	**Restauración de Paciente** Recuperar pacientes de la lista inactivos.	Paciente desactivado (Inactivo).	Switch "Ver Inactivos"	1. Activar switch "Inactivos". 2. Buscar paciente. 3. Click Icono "Activar"	Paciente desaparece de lista inactiva y aparece en activa.	✅	Media

CP_006	Pacientes	**Cancelación de Paciente** Mover al paciente a la lista de inactivos.	Paciente activado.	Switch "Ver activos"	1. Activar switch "activos". 2. Buscar paciente. 3. Click Icono "desactivar"	Paciente desaparece de lista activos y aparece en la lista inactivos .	✅	Media
CP_007	Pacientes	**Borrado Físico (Cleanup)** Eliminar definitivamente.	Pacientes cancelado existente.	Filtro: "Ver Cancelados"	1. Activar switch "Cancelados". 2. Click "Borrar Definitivo". 3. Aceptar Alerta Nativa.	El paciente desaparece de la base de datos.	✅	Alta
CP_013	Pacientes	**Consultar Historia Clínica**	1. Usuario logueado. 2. Existencia de un paciente con datos cargados.	Nombre del Paciente (Ej: "Juan Perez")	1. Navegar al menú "Pacientes" -> "Listado". 2. Filtrar la grilla por nombre del paciente. 3. Ubicar la fila y hacer clic en el botón "Imprimir/Historia" (Ícono Impresora)	Se abre una nueva pestaña en el navegador visualizando el reporte PDF de la Historia Clínica del paciente.	✅	Baja

ID	Módulo	Título / Objetivo	Pre-condiciones	Datos de Entrada	Pasos de Prueba	Resultado Esperado	✅	Prioridad
CP_008	Turnos	**Alta de Turno** Asignar turno a paciente existente.	Paciente creado previamente.	Paciente: "Test Auto" Fecha: Hoy	1. Ir a Turnos > Nuevo. 2. Buscar Paciente. 3. Click en agenda libre. 4. Guardar.	El bloque del turno se renderiza en el calendario.	✅	**Alta**
CP_009	Turnos	**Cancelación Lógica** Dar de baja un turno sin borrarlo.	Turno activo en agenda.	N/A	1. Seleccionar turno. 2. Click "Eliminar". 3. Confirmar en Popup (JSF).	El turno desaparece de la vista activa (pasa a estado Cancelado).	✅	Media
CP_010	Turnos	**Borrado Físico (Cleanup)** Eliminar definitivamente.	Turno cancelado existente.	Filtro: "Ver Cancelados"	1. Activar switch "Cancelados". 2. Click "Borrar Definitivo". 3. Aceptar Alerta Nativa.	El turno desaparece de la base de datos.	✅	Alta
CP_011	Turnos	**Edición de Turno**	Turno creado en agenda.	Nota: "Confirmado	1. Click en Turno	El turno persiste con la nueva	✅	Media

		Agregar notas a un turno existente.		QA"	existente. 2. Modificar campo "Notas". 3. Guardar cambios.	información.		
CP_012	Turnos	**Acceso a Reportes** (Impresión)	Usuario administrador logueado en el sistema.	N/A (Navegación)	1. Desplegar el menú "Turnos". 2. Hacer clic en la opción "Imprimir". 3. Verificar que cargue la pantalla de filtros. 4. Validar existencia del botón "Imprimir"	La pantalla de filtros se visualiza correctamente y el botón "Imprimir" se encuentra habilitado y visible.	✅	Baja

Tabla 3. Matriz de cobertura y diseño de casos de prueba para el Módulo de Autenticación, Pacientes y Turnos.

Anexo III: Reportes de Ejecución.

En la presente sección se adjunta la documentación visual generada automáticamente por la herramienta Allure Framework tras la ejecución final de la suite de regresión. Estos reportes certifican la trazabilidad entre el código de prueba y los resultados obtenidos, proporcionando transparencia sobre la estabilidad del sistema.

A continuación, la **Figura 1** presenta el tablero de control (Dashboard) general de la ejecución. En esta vista de alto nivel se puede corroborar el estado de salud del build actual. El gráfico circular central indica una tasa de éxito del 100% sobre los 13 escenarios críticos definidos en el alcance del proyecto, sin reportes de fallos (failed) ni pruebas rotas (broken), lo que habilita el paso a producción

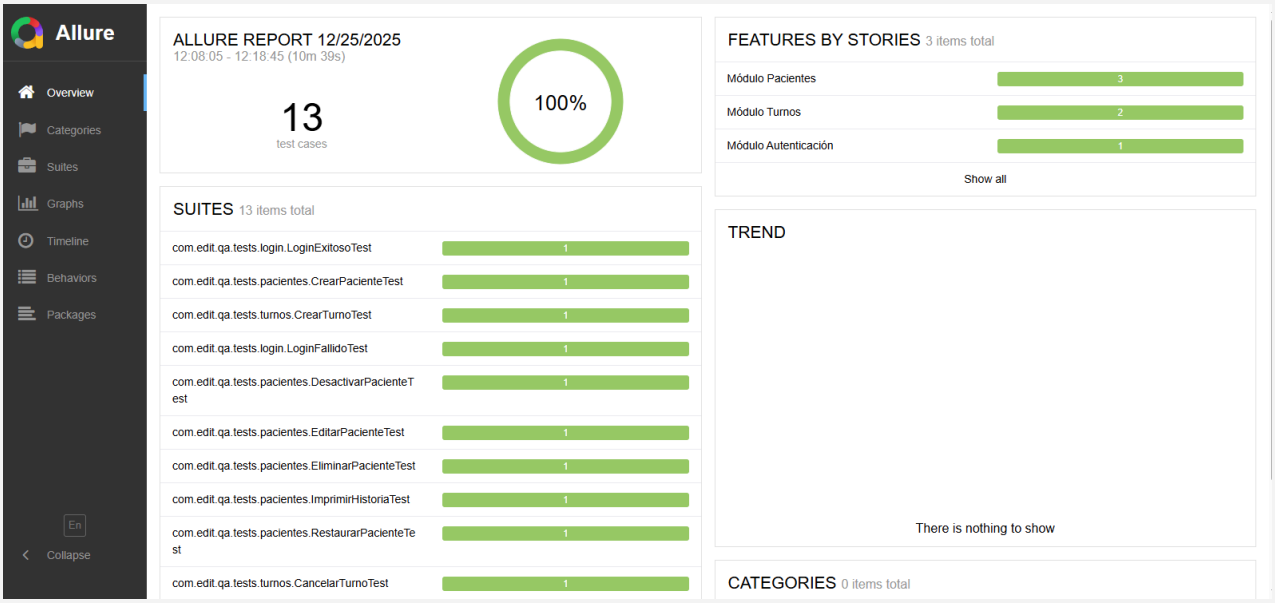


Figura 1. Dashboard principal de Allure Report mostrando el estado global de la ejecución (100% aprobado).

Lista de Behaviors/Módulos

Para facilitar el análisis funcional por parte de los interesados (*stakeholders*), el reporte permite desagregar los resultados según la lógica de negocio. La **Figura 2** detalla la estructura jerárquica de la suite, organizando los casos de prueba en "Comportamientos" (*Behaviors*) que se corresponden con los módulos del ERP (Autenticación, Pacientes y

Turnos). Esta visualización confirma que la cobertura de pruebas ha abarcado todas las áreas funcionales planificadas.

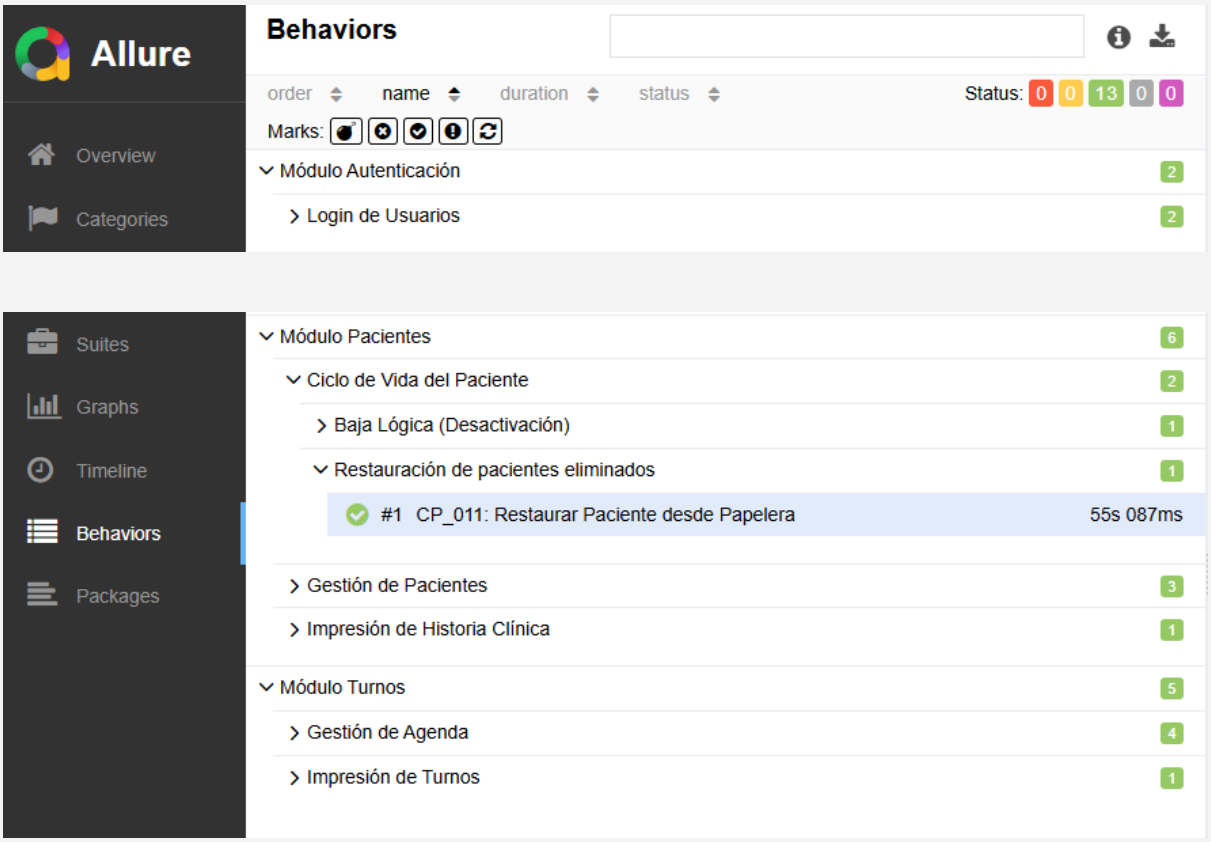


Figura 2. Vista de comportamientos (*Behaviors*) desglosada por módulos funcionales y tiempos de respuesta.

Detalle de Pasos

Finalmente, es posible descender al nivel de granularidad más bajo para auditoría técnica. La **Figura 3** expone la trazabilidad completa del caso de prueba **CP_011 (Restaurar Paciente)**. Se listan cronológicamente las interacciones realizadas por el robot (navegación, clicks, inyección de datos) junto con el tiempo exacto de procesamiento en milisegundos de cada paso. Esta información es crítica para detectar degradaciones de rendimiento en componentes específicos de la interfaz.

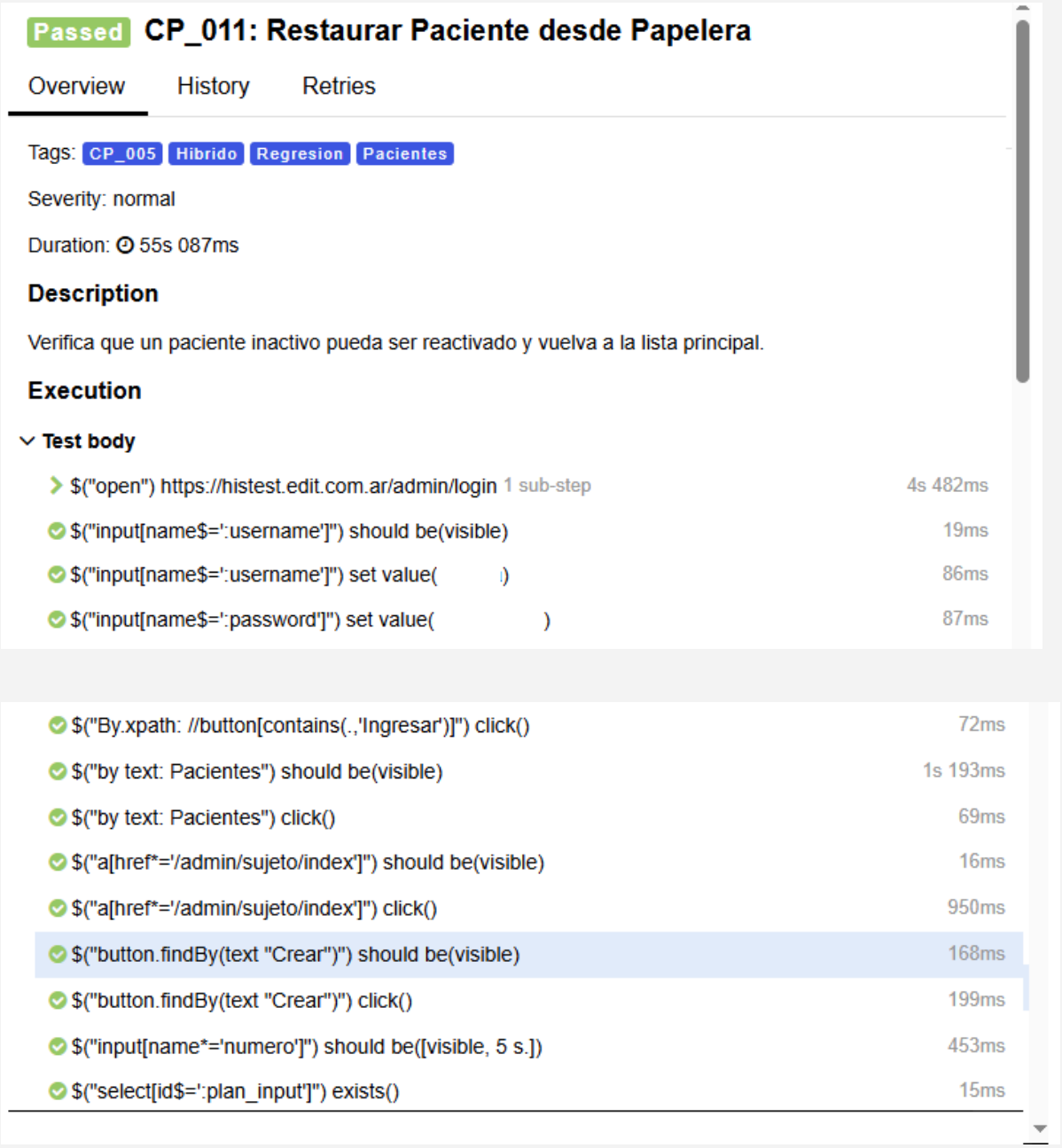


Figura 3. Vista de pasos detallados de la ejecución (*Test Steps*) para el caso de restauración de paciente

Anexo IV: Guía de Instalación y Transferencia técnica.

1. Descripción General del Artefacto

Este apartado es la guía técnica para la instalación, configuración y ejecución del Framework de Automatización de Pruebas de Regresión. Este software ha sido desarrollado bajo una Arquitectura Híbrida para abordar la complejidad de aplicaciones web basadas en JavaServer

Faces (JSF) y PrimeFaces.

2. Requisitos del Entorno (Prerrequisitos)

Para garantizar la correcta ejecución de la suite de pruebas en entornos locales o servidores de CI/CD, se requiere la siguiente configuración base:

Java Development Kit (JDK): Versión 17 LTS o superior.

Gestor de Dependencias: Apache Maven 3.8.1 o superior.

Navegador Web: Google Chrome (versión estable actual). *Nota: El driver es gestionado automáticamente por la dependencia WebDriverManager.*

Conectividad de Red: Acceso habilitado (vía VPN o red local) al entorno de pruebas https://sistema.edit.com.ar

3. Configuración de Propiedades y Credenciales

El proyecto evita el uso de credenciales embebidas (hardcoded) mediante el uso de archivos de propiedades externos.

Procedimiento de Configuración:

1. Localizar el directorio de recursos en la ruta: `src/test/resources`.

2. Crear o editar el archivo `config.properties`.

3. Configurar los parámetros siguiendo la estructura definida a continuación:

```
  # URL Base del Sistema Bajo Prueba (SUT)
   base.url=https://sistema.edit.com.ar
# Credenciales de Administrador
 admin.user=usuario_admin_qa
  admin.pass=contraseña_segura_123
# Configuraciones del Motor de Ejecución
  Timeout explícito global (en milisegundos)
 timeout.standard=10000
```

```
# Ejecución sin interfaz gráfica (true para CI/CD, false para debug)
headless.mode=false
```

4. Manual de Ejecución (Comandos Maven)

La orquestación de las pruebas se realiza a través de la línea de comandos utilizando Maven Wrapper o la instalación local de Maven.

A. Ejecución de la Suite Completa

Ejecuta la totalidad de los casos de prueba automatizados (cobertura total).

```
mvn clean test
```

B. Ejecución de Smoke Tests

Ejecuta un subconjunto crítico de pruebas para validar la disponibilidad del entorno post-despliegue.

```
mvn test -Dgroups=smoke
```

C. Ejecución de la Suite de Regresión

Ejecuta las pruebas funcionales de los módulos de Pacientes y Turnos para asegurar la estabilidad del Core Business.

```
mvn test -Dgroups=regression
```

D. Ejecución de un Script Específico

Comando para aislar y depurar una única clase de prueba (Ejemplo: creación de pacientes).

```
mvn clean test -Dtest="CrearPacienteTest"
```

5. Generación y Visualización de Reportes

El framework utiliza Allure Reports para la generación de evidencia de ejecución. Los artefactos (capturas de pantalla, trazas de error y tiempos) se almacenan en el directorio `/target/allure-results`.

Para visualizar el reporte interactivo, ejecutar el siguiente comando al finalizar las pruebas:

```
mvn allure:serve
```

Este comando levantará un servidor web local y abrirá el reporte en el navegador predeterminado.

9. Agradecimientos

A mi tutor de empresa, Antony Lescano, por su mentoría técnica y por desafiarme a buscar soluciones innovadoras ante las limitaciones de la arquitectura heredada.

A los docentes de la Universidad Nacional del Noroeste de la Provincia de Buenos Aires (UNNOBA), por proporcionarme las bases teóricas de Ingeniería de Software que hicieron posible este análisis.

Al equipo de desarrollo de Edit Software, por su apertura al cambio y su colaboración durante la integración de estas nuevas herramientas en su flujo de trabajo diario.