Automatización de pruebas de procesamiento de lenguaje natural

Autor:
Juan Martín Paz Ussa
Cód. 100613020435

Asesores:
Oscar Mauricio Caicedo Rendón, Ph.D.
Hebert Jair Gómez Fajardo

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Programa de Ingeniería Electrónica y Telecomunicaciones
Octubre 2025

Contenido

1 Introducción

El presente informe documenta el trabajo realizado durante la pasantía desarrollada en la empresa **Grandtek**, enfocada en el *desarrollo de soluciones especializadas de software* basadas en tecnologías de la información y las comunicaciones (TIC). Grandtek ofrece servicios de mantenimiento y soporte informático, así como desarrollo de software especializado para diversos sectores, respaldados por un equipo de ingenieros expertos en diseñar soluciones confiables basadas en TIC.

La pasantía se orientó a la **automatización de pruebas de conversaciones de WhatsApp** para un bot del ámbito agrícola, con el propósito de validar historias de usuario a lo largo de *tres fases del proyecto* de la aplicación de Agricultura de Precisión. Este trabajo se articuló en torno al ámbito de la automatización web, pruebas end-to-end con la librería Playwright y gestión reproducible de datos de prueba.

El desarrollo de este proyecto se sustenta en principios establecidos de ingeniería de software, automatización de pruebas y sistemas conversacionales. Las pruebas automatizadas son una práctica fundamental en el desarrollo de software moderno. Según Garousi y Mäntylä [1], las pruebas automatizadas reducen significativamente el tiempo de ejecución de pruebas repetitivas y aumentan la cobertura de código, permitiendo detectar regresiones tempranamente. En el contexto de aplicaciones web, las pruebas end-to-end (E2E) validan el sistema completo desde la perspectiva del usuario final, asegurando que todos los componentes integrados funcionen correctamente en conjunto [2]. El patrón *Page Object Model* (POM), utilizado en este proyecto, es reconocido como una práctica efectiva para estructurar pruebas automatizadas de interfaces de usuario. Leotta et al. [3] demuestran que POM mejora la mantenibilidad del código de pruebas al encapsular la lógica de interacción con elementos de la interfaz, reduciendo la duplicación y facilitando la adaptación a cambios en la UI. Por su parte, los agentes conversacionales o *chatbots* han experimentado un crecimiento exponencial en diversos dominios. Adamopoulou y Moussiades [4] clasifican los chatbots según su arquitectura en: basados en reglas, basados en recuperación de información, y basados en modelos generativos. El bot bajo prueba en este proyecto se enmarca en sistemas híbridos que combinan procesamiento de lenguaje natural con lógica de negocio específica del dominio agrícola.

WhatsApp Web presenta desafíos específicos para la automatización debido a su naturaleza asíncrona, actualizaciones en tiempo real y dependencia de servicios externos. La literatura sobre testing de aplicaciones de mensajería instantánea [8] destaca la importancia de: (i) esperas inteligentes con estrategias de *polling* para sincronización con el servidor; (ii) manejo robusto de variabilidad en tiempos de respuesta; y (iii) aislamiento de datos entre pruebas para garantizar reproducibilidad. El enfoque implementado en este proyecto, basado en detección de patrones mediante expresiones regulares y sistema de reglas con prioridades, se alinea con las mejores prácticas identificadas en la literatura para testing de interfaces conversacionales no determinísticas [9].

Los antecedentes que motivaron esta pasantía incluyen la experiencia previa del pasante en **desarrollo web** y **automatización de pruebas**, lo cual facilitó la adopción de tecnologías como TypeScript y Playwright, así como el diseño de *flujos conversacionales automatizados*

que interactúan con WhatsApp Web. Este punto de partida permitió alinear el trabajo con las necesidades del proyecto.

La pasantía se articula con la formación del programa específicamente en el énfasis en **Telemática**, al integrar patrones de diseño de software, plataformas de comunicación en red y pruebas de software para construir soluciones de validación de extremo a extremo. Esta experiencia fortaleció competencias técnicas y blandas alineadas con el perfil del estudiante.

El alcance de la pasantía abarcó: (i) la validación de historias de usuario priorizadas en las *tres fases* del proyecto; (ii) la definición y materialización de **datos de prueba versionados** para dominios agrícolas (cultivos, fertilizantes, fitosanitarios); (iii) la **automatización de flujos conversacionales** en WhatsApp Web con Playwright; y (iv) la **estandarización de reportes** de resultados de ejecución.

Entre las principales limitaciones se encuentran: la dependencia de la disponibilidad de *WhatsApp Web* y del bot bajo prueba; restricciones de tiempo propias del periodo de práctica que acotan el trabajo a las historias priorizadas de Fases 1–3.

Objetivos

Objetivo general. Desarrollar un conjunto de pruebas automatizadas de WhatsApp que valide las funcionalidades de las historias de usuario de las tres fases del proyecto de la app de Agricultura de precisión, generando una base para futuras extensiones del proyecto.

Objetivos específicos.

1) **Modelo de datos de prueba reproducible y seguro.** Diseñar un modelo de datos de prueba para conversaciones, con entregables que incluyan: conjunto de datos versionados (cultivos, fertilizantes, fitosanitarios, etc.), plantillas y selección de dataset por etiqueta/ambiente.

2) **Automatización de intercambio de mensajes (Fases 1–3).** Automatizar end-to-end el intercambio de mensajes usuario–bot para las historias de usuario priorizadas de Fase 1, 2 y 3, mediante *steps* y *Page Objects* reutilizables. Entregables: repositorio de steps para todas las fases (enviar mensaje, seleccionar opción, validar respuesta), orquestación por etiquetas de Fase/HU y ejecución paralela local.

3) **Generación de reportes estandarizada.** Implementar y estandarizar la generación de reportes post-ejecución con Playwright, consolidando HTML, JSON y evidencia de capturas automáticas. Entregable: script de generación de reportes que produce HTML y reporte de casos exitosos y pasos completados.

2 Motivación: El problema de la validación en sistemas conversacionales

La adopción acelerada de interfaces conversacionales en aplicaciones empresariales ha expuesto una paradoja fundamental en el aseguramiento de calidad del software: mientras que los sistemas tradicionales con interfaces gráficas permiten validación determinística mediante secuencias de acciones predefinidas, los agentes conversacionales introducen un espacio de estados exponencialmente mayor debido a la naturaleza no estructurada del lenguaje natural. Esta complejidad se amplifica cuando el canal de comunicación es una plataforma de mensajería de terceros como WhatsApp, donde el tester no controla ni el protocolo de comunicación ni los tiempos de respuesta del servidor.

El desafío central radica en que cada intención del usuario puede expresarse mediante decenas o cientos de variaciones lingüísticas semánticamente equivalentes. Un usuario puede solicitar información sobre cultivos diciendo "dame la lista de cultivos", "muéstrame qué cultivos tengo", "cuáles son mis cultivos registrados" o cualquiera de otras muchas formulaciones. Si consideramos que un sistema conversacional empresarial típico maneja entre veinte y cincuenta intenciones distintas, y cada una admite entre diez y cien variaciones naturales, el espacio de prueba crece hasta miles de combinaciones posibles. La estrategia tradicional de escribir casos de prueba manualmente para cada variación resulta inviable tanto por el costo temporal como por el riesgo de obsolescencia ante cambios en el modelo de procesamiento de lenguaje natural del bot.

Esta situación plantea un problema de ingeniería de particular relevancia: cómo diseñar una arquitectura de pruebas automatizadas que sea suficientemente flexible para adaptarse a la variabilidad lingüística, pero lo suficientemente robusta para detectar regresiones funcionales de manera confiable. La complejidad se multiplica cuando se considera que el bot bajo prueba no es una entidad aislada, sino que depende de múltiples servicios backend (bases de datos, APIs de terceros, motores de reglas de negocio) cuyo estado puede influir en las respuestas del agente conversacional. Una prueba exitosa no solo debe validar que el bot comprende correctamente la intención expresada en lenguaje natural, sino también que ejecuta correctamente la lógica de negocio asociada y devuelve información precisa desde las capas de persistencia.

La literatura académica sobre testing de sistemas conversacionales ha identificado tres problemas fundamentales que permanecen sin solución definitiva. El primero es el problema de la completitud: dado un conjunto finito de casos de prueba automatizados, ¿cómo garantizar que cubren adecuadamente el comportamiento esperado del sistema frente a entradas del mundo real? La naturaleza abierta del lenguaje natural hace imposible enumerar exhaustivamente todas las entradas válidas. El segundo problema es el de la detección de falsos positivos: cuando un bot devuelve una respuesta que difiere textualmente de la esperada pero es semánticamente correcta, el sistema de pruebas debe ser capaz de reconocer esta equivalencia sin requerir intervención manual. El tercer problema, especialmente crítico en aplicaciones empresariales, es la reproducibilidad: las pruebas deben ejecutarse de manera efectiva independientemente del estado previo del sistema, lo cual requiere estrategias de gestión de datos de prueba y aislamiento de contexto conversacional.

La motivación de este proyecto surge precisamente de la necesidad de abordar estos problemas mediante una arquitectura de pruebas que combine tres pilares conceptuales. El primero es la

parametrización exhaustiva de datos de prueba: en lugar de codificar casos de prueba individuales, se define un modelo de datos que describe las variaciones lingüísticas permitidas para cada intención, permitiendo generar dinámicamente cientos de casos de prueba a partir de plantillas. El segundo pilar es el sistema de detección de respuestas basado en patrones con prioridades: mediante expresiones regulares ordenadas por especificidad, el framework puede reconocer múltiples formas de respuesta válida del bot sin requerir coincidencia textual exacta. El tercer pilar es el manejo de estado conversacional mediante operaciones idempotentes: cada prueba comienza limpiando el historial de conversación y termina en un estado conocido, garantizando que las ejecuciones subsecuentes no sufran interferencia por efectos residuales.

3 Plan de trabajo avalado

3.1 Actividades

Preparación y configuración inicial

- Recopilar criterios de aceptación por HU.
- Reescribir criterios de aceptación iniciales en Gherkin
- Configurar repo TS + Cucumber + Playwright.
- Login persistente a WhatsApp Web (state.json) y chat Twilio o Whatsapp Business.
- Definir convenciones: tags por HU/Fase, nombres de steps, estructura de carpetas, formato de reporte.

Base de automatización

- POM WhatsApp: abrir chat, enviar mensaje, esperar respuesta, confirmar textos, manejar multi-turno.
- Wait helpers: esperas activas por “pregunta del bot”, “confirma”, “listado”.
- Message parsers: utilidades para detectar prompts del bot y listas/tablas en texto.
- Modelo de datos mínimo: crear/limpiar cultivos y productos de prueba.

Automatización de Fase 1

- HU-002: Escenarios de usuario válido/ inválido y sesión expirada.
- HU-006: Listado y filtrado de cultivos.
- HU-007 a HU-019: Listado de variedades, fertilizantes, fitosanitarios y otros recursos.

- Refactorización de pasos comunes (inicio/cierre de sesión, navegación).
- Registro de incidentes detectados durante la ejecución.

Automatización de Fase 2

- HU-020/021/022: Creación de productos (cultivo, fertilizante, químico) con validación de campos.
- HU-003: Listar productos por fabricante
- HU-004: Asignación de precio de producto y verificación de persistencia.
- HU-005: Búsqueda por materia activa (coincidencias exactas y parciales).
- Reutilización de page objects y steps; generación de reportes básicos.

Automatización de Fase 3

- HU-023: Creación de campañas (manejo de datos y fechas).
- HU-024: Consulta de historial de campañas (filtrado, orden, detalle).
- HU-025: Planificación de trabajos a campos, confirmando estados.
- Integración parcial de flujos end-to-end combinando fases anteriores.

Revisión, documentación y entrega

- Consolidar scripts y pasos en un README final con instrucciones detalladas.
- Especificar dependencias y cómo añadir nuevas pruebas.
- Redactar informe de cobertura, hallazgos y recomendaciones futuras para abordar CI/CD.
- Elaborar un informe de retrospectiva con aprendizajes y sugerencias.

3.2 Cronograma de la pasantía:

Semana 1 — Page objects reutilizables — 40 h

- Reescritura Gherkin; definición de tags y convenciones — 10 h
- Setup repo; POM WhatsApp (buscar chat, enviar/leer) + wait helpers base — 10 h
- DataFactory v1 (cultivos/variedades mínimas) + datasets versionados — 6 h

- Login persistente y validación de sesión/chat objetivo — 2 h
- Implementación HU-002 + evidencia — 4 h
- Implementación HU-006 + evidencia — 4 h
- Revisión y ajustes — 4 h

Entregables: repositorio funcional, features/steps ejecutables, reporte HTML con screenshots.

Semana 2 — Fase 1 completa + "Crear cultivo" — 40 h

- HU-016 listar por nombre: exacto/parcial/inexistente — 4 h
- HU-007 listar variedades — 3 h
- HU-008 listar fertilizantes — 2 h
- HU-009 listar fitosanitarios — 2 h
- HU-010 fertilizantes por nombre — 2 h
- HU-011 último precio producto — 2 h
- HU-012 variación precio producto — 2 h
- HU-013 precio mínimo producto — 2 h
- HU-014 consultar campos sin planificar — 2 h
- HU-017 consultar distribución cultivos — 3 h
- HU-018 consultar planificación campaña — 3 h
- HU-019 consultar trabajos — 2 h
- Flujo "Crear cultivo" — 6 h
- Refactor de pasos comunes + consolidación de message parsers + registro de bugs — 5 h

Entregables: HUs de Fase 1 + Crear cultivo automatizados; reporte y matriz de cobertura v1.

Semana 3 — Fase 2 — 40 h

- HU-021 crear fertilizante: campos obligatorios/valores inválidos — 8 h
- HU-004 asignar precio: persistencia/validaciones de rango — 8 h
- HU-005 buscar por materia activa: exacta/parcial/sin resultados — 7 h
- HU-003 listar productos por fabricante (fertilizantes y fitosanitarios) — 5 h
- Automatización HU-020 y HU-022 — 6 h
- Validación manual + ajustes de fixtures + reportes + refactor — 6 h

Entregables: Automatización HUs de Fase 2, reportes con evidencias.

Semana 4 — Fase 3 — 40 h

- U-023 crear campaña fechas validaciones estado — 10 h
- HU-024 consultar historial de campañas filtros orden detalle — 5 h
- HU-025 planificar trabajo a un campo estados confirmaciones — 5 h
- HU-026 reportar trabajo en un campo — 4 h
- HU-027 consultar último trabajo realizado — 4 h
- HU-028 implementación del RAG — 6 h
- Estabilización medir flakiness, afinar waits reintentos — 3 h
- Documentación final README guía para nuevas HUs matriz de cobertura informe de hallazgos proyección CD CI — 3 h

Entregables: HUs fase 3 completa; matriz final HU–Escenarios–Evidencia; reportes con screenshots; informe final.

Duración: 4 semanas

Intensidad horaria: 160 horas (40 h / semana)

Número de créditos: 3

4 Actividades de la pasantía

Durante la pasantía se ejecutó un conjunto de actividades orientadas a diseñar, implementar y operar un sistema de validación end-to-end de conversaciones de WhatsApp para un bot del

dominio agrícola. Este trabajo implicó la construcción de un framework de automatización que incluye orquestación de pruebas, gestión de datos, ejecución automatizada de conversaciones y generación de reportes con evidencias detalladas.

4.1 Descripción detallada de las actividades

Fase de preparación y configuración inicial

La primera semana se dedicó a establecer los cimientos del proyecto de automatización. Se configuró un repositorio TypeScript con Playwright como motor de automatización de navegador.

Configuración del entorno de desarrollo Se estableció un proyecto Node.js con TypeScript, configurando compatibilidad ES2020 y gestión de módulos. El archivo de configuración del framework se parametrizó con timeouts extendidos (15 minutos por prueba) para permitir conversaciones multi-turno prolongadas.

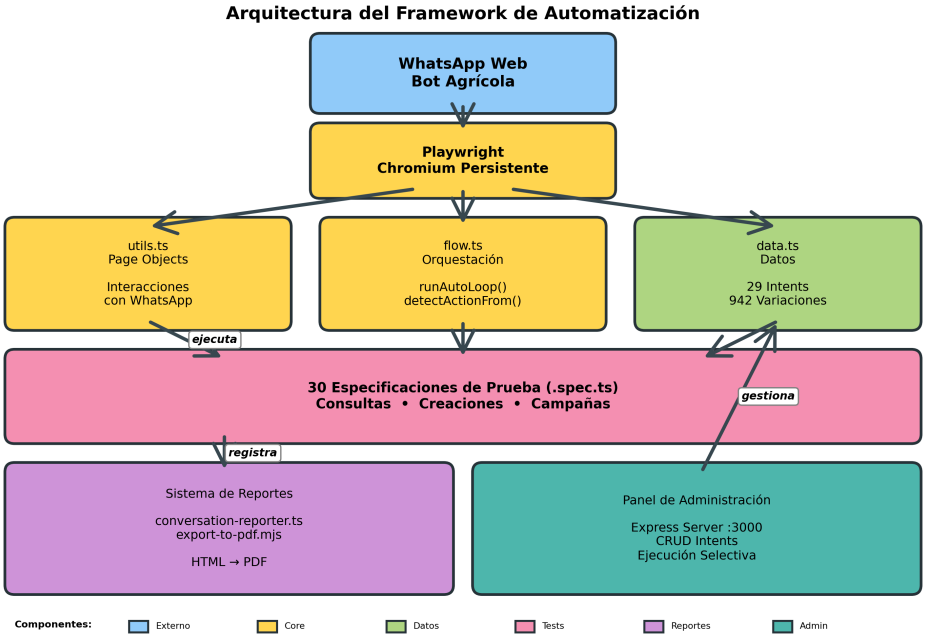

Figure 1. Estructura del proyecto de automatización

Manejo de sesión persistente en WhatsApp Web Un desafío crítico fue mantener la sesión de WhatsApp Web entre ejecuciones de pruebas sin requerir autenticación manual con código QR en cada inicio. Se implementó la estrategia de contexto persistente de Playwright, almacenando el estado del navegador (cookies, almacenamiento local y de sesión) en un directorio dedicado. El módulo de utilidades de autenticación contiene una función especializada que:

- Verifica si existe una sesión activa comprobando elementos característicos de WhatsApp Web cargado
- Si no hay sesión, espera la presentación del código QR y la autenticación manual
- Una vez autenticado, persiste el estado para ejecuciones futuras
- Incluye mecanismos de reintento con esperas exponenciales para manejar latencias de red

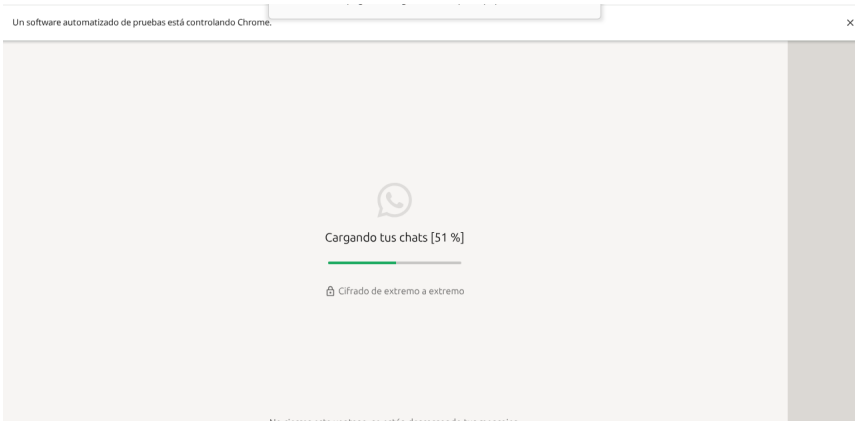

Figure 2. Proceso de autenticación y persistencia de sesión

Definición de convenciones y etiquetado Se estableció un sistema de nomenclatura y organización:

- Archivos de prueba nombrados según la funcionalidad o historia de usuario que validan
- Estructura de carpetas organizada: directorio principal para especificaciones de prueba y subdirectorios para utilidades compartidas
- Etiquetas (tags) por fase y tipo de funcionalidad para permitir ejecución selectiva
- Convenciones de nombrado para variables de entorno en archivo de configuración

Construcción de la base de automatización

La segunda semana se enfocó en construir los componentes reutilizables que formarían el núcleo del framework de pruebas.

Page Objects y utilidades de interacción Siguiendo el patrón Page Object Model, se desarrolló un conjunto de funciones especializadas para interactuar con la interfaz de WhatsApp Web:

- Función para localizar y abrir el chat con el contacto específico (el bot bajo prueba)
- Función para limpiar el historial del chat e iniciar con contexto limpio
- Función para escribir en el campo de texto del compositor de mensajes, manejando el foco correcto
- Función para ejecutar el envío del mensaje (mediante clic en botón o tecla Enter)
- Función para esperar activamente la aparición de una nueva burbuja de mensaje del bot
- Función para obtener solo los mensajes nuevos recibidos desde un punto de referencia
- Función para contar el total de burbujas de mensajes entrantes en el chat

Estas funciones encapsulan los selectores CSS y el manejo de condiciones de carrera propias de la interfaz web de mensajería.

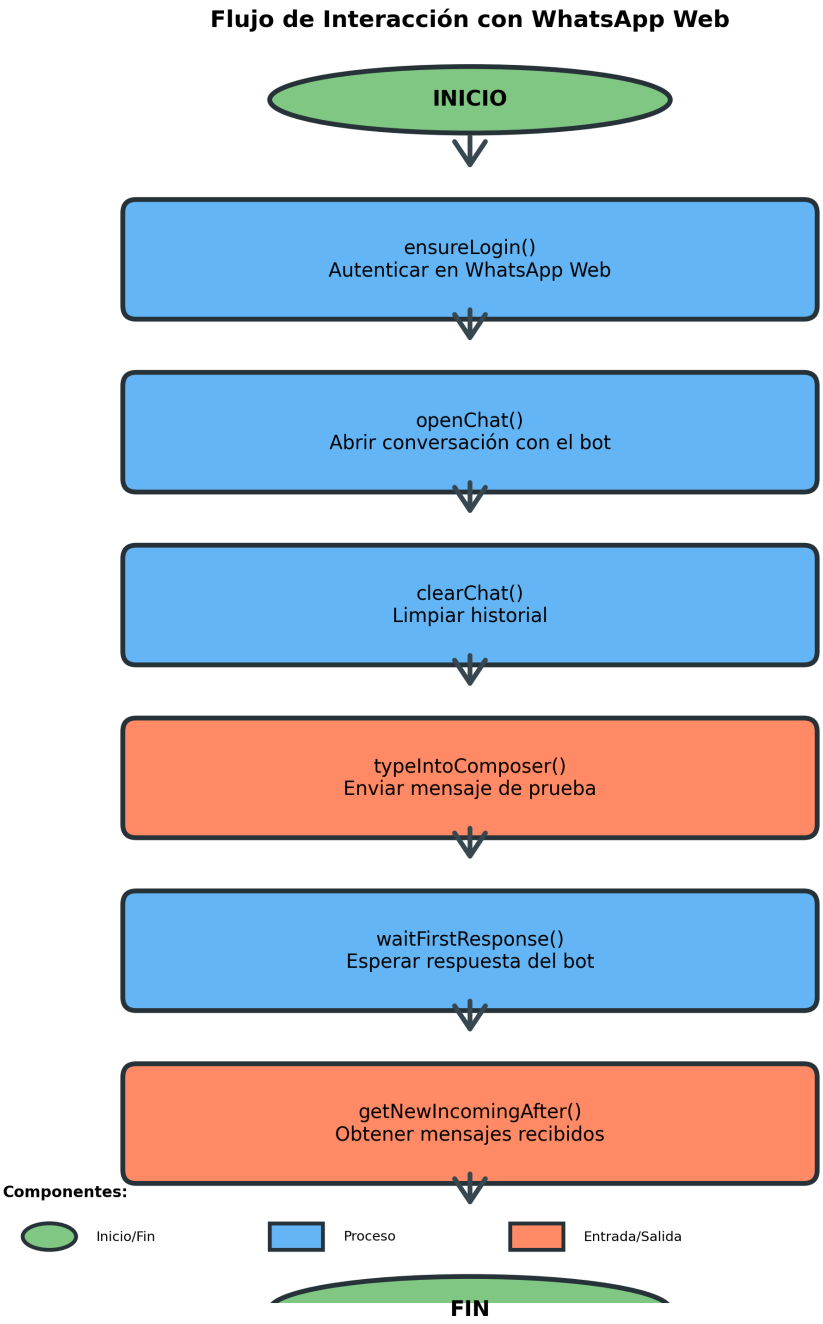

Figure 3. Flujo de interacción con WhatsApp Web mediante Page Objects

Esperas inteligentes y manejo de asincronía Dado que las respuestas del bot dependen de latencias de red, procesamiento del servidor y tiempos de entrega de WhatsApp, se implementaron estrategias de espera robustas:

- **Esperas explícitas con polling:** En lugar de esperas fijas, se implementa un bucle que verifica periódicamente la aparición de nuevos mensajes, con tiempo límite configurable
- **Detección de patrones de texto:** Funciones especializadas que esperan hasta que aparezca un mensaje que coincida con un patrón específico
- **Reintentos acotados:** Se permite un número limitado de reintentos con incrementos exponenciales de tiempo cuando se detectan condiciones transitorias (ej. carga lenta de interfaz)
- **Tiempos límite configurables:** Los timeouts se parametrizan mediante variables de entorno para ajustar según las características del servidor de pruebas

Parsers de mensajes y lógica de decisión El bot responde con textos requeridos (listados, mensajes de existe o no existe, opciones). Se desarrollaron analizadores sintácticos (parsers) especializados:

- Función para extraer el primer elemento de una lista y seleccionarlo automáticamente
- Sistema de reglas basado en expresiones regulares con prioridades, que analiza la respuesta del bot y determina la acción apropiada (responder con un valor específico, terminar exitosamente, terminar con error, etc.)
- Función que aplica las reglas sobre los mensajes recibidos y retorna la acción a ejecutar
- Función para resolver plantillas con variables del contexto actual (ej. reemplazar marcadores de posición con valores generados dinámicamente)

Este sistema permite que las pruebas sean declarativas y adapten su comportamiento dinámicamente según las respuestas del bot.

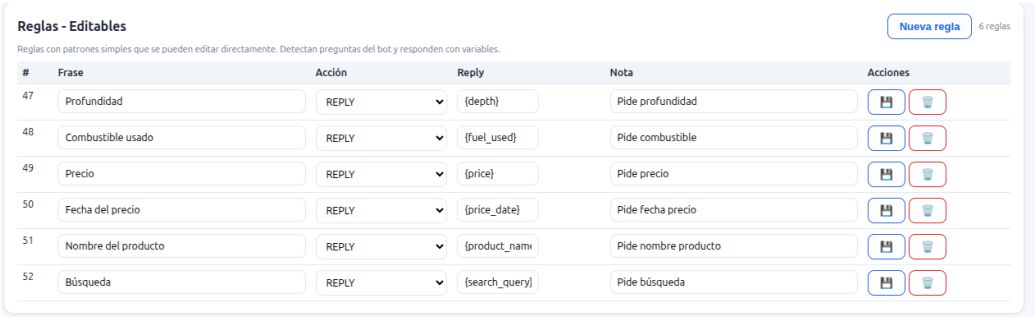

Figure 4. Sistema de reglas por palabras clave para decisión automática

Modelo de datos y fixtures Se centralizó la gestión de datos de prueba en un módulo dedicado:

- **Diccionario de variables:** Variables contextuales (nombres de cultivos, fertilizantes, clientes, fechas) que se materializan en las plantillas de mensajes
- **Mapa de intents:** Estructura que asocia cada intent (ej. "crear cultivo", "listar fertilizantes") con múltiples frases de ejemplo para probar variaciones de entrada
- **Funciones de selección aleatoria:** Funciones para seleccionar datos aleatorios y variar entre ejecuciones, evitando colisiones
- **Reinicio de variables:** Función para asegurar estado limpio al inicio de cada prueba
- **Mutaciones para reintentos:** Función que altera ligeramente los datos en caso de fallos transitorios para intentar nuevamente

Automatización de Fase 1: Consultas básicas

La segunda y tercera semana se automatizaron las historias de usuario de consulta y listado (Fase 1 del plan de trabajo):

HU-002: Autenticación y gestión de sesión Se implementaron escenarios para validar el flujo de solicitud de OTP por correo, ingreso de código válido, manejo de código inválido y detección de sesión expirada.

HU-006: Listado y filtrado de cultivos Las pruebas automatizadas implementan:

- Listado completo de cultivos registrados
- Filtrado por nombre (coincidencia exacta y parcial)
- Manejo de casos sin resultados
- Creación exitosa de cultivos proporcionando todos los campos requeridos (nombre, variedad, marca, destino)

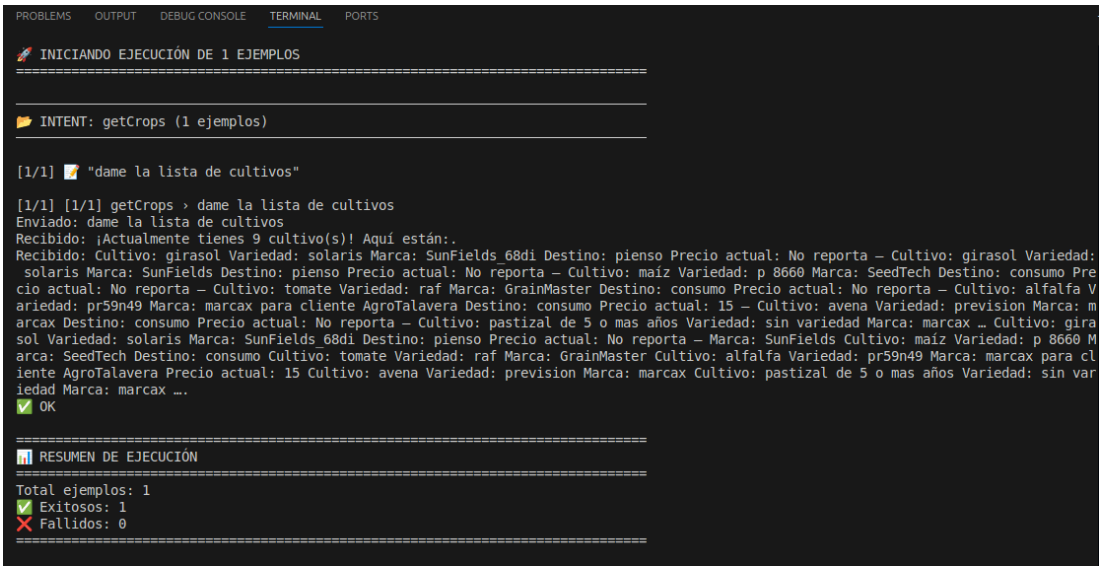

Figure 5. Ejecución de listado y filtrado de cultivos

HU-007 a HU-019: Consultas de recursos agrícolas Se automatizaron especificaciones individuales para:

- **HU-008, HU-009, HU-010:** Listado de fertilizantes y fitosanitarios, con filtrado por nombre
- **HU-011, HU-012, HU-013:** Consultas de precios (último precio, variación histórica, precio mínimo)
- **HU-014:** Consulta de campos sin planificación
- **HU-017, HU-018, HU-019:** Consultas de distribución de cultivos, planificaciones de campaña y trabajos pendientes

Cada especificación de prueba sigue un patrón estructurado donde se define el caso de prueba y se ejecuta un bucle automático que:

1. Limpia el chat para iniciar con contexto limpio
2. Envía la frase de inicio del intent
3. Entra en un bucle automático de lectura de respuestas y envío de acciones determinadas por las reglas definidas
4. Registra cada paso en el sistema de trazabilidad conversacional
5. Retorna resultado de éxito o fallo según los criterios de aceptación

Automatización de Fase 2: Creación y gestión de productos

Durante la tercera semana se automatizaron los flujos de creación con validación exhaustiva de campos:

HU-020, HU-021, HU-022: Creación de productos Se implementaron especificaciones para crear cultivos, fertilizantes y productos químicos mediante flujos conversacionales completos:

- Flujos de creación exitosa proporcionando todos los campos requeridos por el bot
- Interacción multi-turno donde el bot solicita secuencialmente cada campo necesario
- Verificación de que la creación se completa con mensaje de éxito del bot

Las especificaciones de prueba contienen múltiples variaciones de frases de inicio para probar diferentes formas de expresar el mismo intent.

HU-004: Asignación de precios Las pruebas implementan:

- Flujo de asignación de precio a un producto mediante conversación con el bot
- Completar el proceso proporcionando producto, precio y fecha cuando el bot lo solicita
- Verificación de mensaje de confirmación exitosa del bot

HU-005: Búsqueda por materia activa Las pruebas verifican búsquedas exactas, parciales y casos sin coincidencias de productos fitosanitarios por su principio activo.

HU-003: Productos por fabricante Se valida el listado de productos de un fabricante específico, incluyendo casos con múltiples productos y fabricantes sin productos.

Automatización de Fase 3: Campañas y trabajos

En la cuarta semana se completaron los flujos más complejos relacionados con planificación agrícola:

HU-023: Creación de campañas Las pruebas implementan:

- Flujo de creación de campaña proporcionando fecha de inicio
- Asociación de campaña con campos y cultivos mediante interacción conversacional
- Completar el proceso hasta recibir confirmación exitosa del bot

HU-024: Historial de campañas Las pruebas validan consultas de historial con:

- Filtrado por rango de fechas
- Ordenamiento cronológico
- Detalles de cada campaña (cultivos asociados, trabajos planificados)

HU-025, HU-026, HU-027: Gestión de trabajos Se automatizaron:

- **Planificación de trabajos**: asignación de tarea a un campo con fecha y tipo de labor
- **Reporte de trabajo finalizado**: confirmación de ejecución con cantidades y observaciones
- **Consultas de trabajos**: trabajos pendientes y último trabajo realizado

Panel de administración y ejecución selectiva

Paralelamente a la automatización de HUs, se desarrolló una interfaz web de administración para facilitar la ejecución de pruebas:

Servidor de administración Se implementó un servidor web local con Express y TypeScript que:

- Sirve una interfaz web accesible localmente (puerto 3000)
- Lee dinámicamente todos los intents y ejemplos desde el módulo de datos, permitiendo editar variables y reglas de automatización.
- Expone servicios REST para obtener la lista de intents, ejemplos por intent, ejecutar selección de ejemplos y sirve como CRUD de intents y frases ejemplo.

Interfaz de usuario La interfaz web permite:

- **Visualización agrupada:** Todos los intents listados con acordeón expandible, mostrando ejemplos dentro de cada intent
- **Ejecución con un clic:** Botón de ejecución que dispara la suite de pruebas seleccionadas mediante API
- **Edición de datos:** Formularios para agregar, editar o eliminar intents y ejemplos, persistiendo cambios en el repositorio de datos
- **Acceso directo a reportes:** Botón para abrir la carpeta de reportes generados

Este panel permite seleccionar, editar y ejecutar subconjuntos de pruebas mediante una interfaz gráfica sin necesidad de editar código.

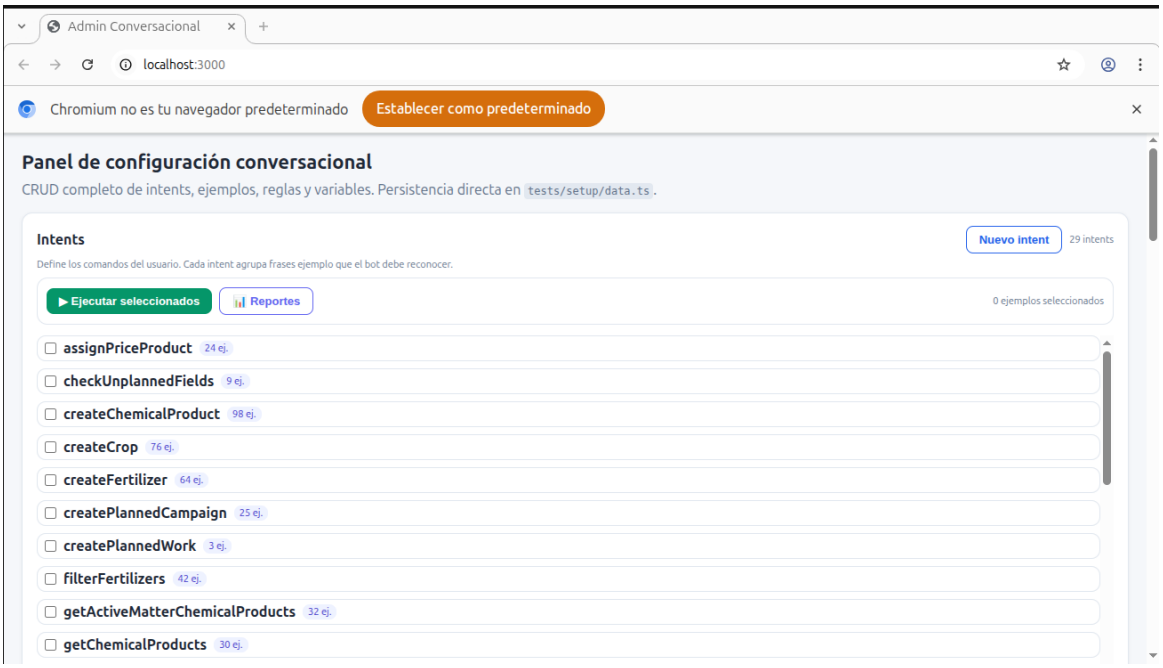

Figure 6. Panel de administración para ejecución selectiva de pruebas

Sistema de reportes con evidencias detalladas

Se desarrolló un generador de reportes personalizado que produce documentos HTML y PDF con:

Características del sistema de reportes

- **Línea temporal conversacional:** Cada mensaje enviado y recibido con marca de tiempo UTC precisa
- **Agrupación por intent:** Los reportes muestran cada intent ejecutado como una sección expandible
- **Indicadores visuales:** Insignias de estado (éxito/fallo) por intent y por paso individual
- **Estadísticas globales:** Resumen con total de eventos, exitosos, fallidos y número de intents ejecutados
- **Exportación automática a PDF:** Script que convierte los HTML a PDF preservando estilos, ejecutado automáticamente al finalizar cada suite
- **Nomenclatura con marca temporal:** Archivos nombrados con fecha y hora para trazabilidad histórica

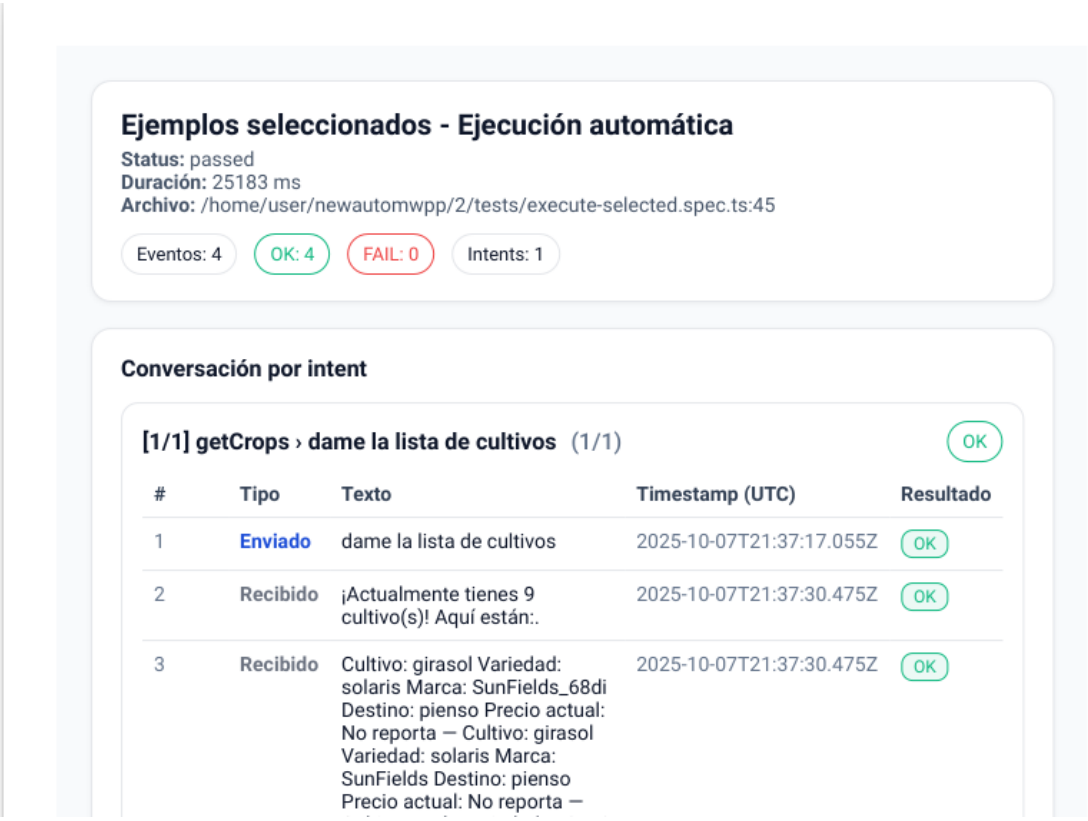

Figure 7. Reporte de conversación con timeline y estados

Estructura del reporte Cada reporte incluye:

```
[Título del test]
Status: passed / failed
Duración: X ms

Resumen:
- Eventos: N
- OK: X
- FAIL: Y
- Intents: Z

Conversación:
[1/Z] intent_name - OK
  >> Enviado: [mensaje]
  << Recibido: [respuesta bot]
  ...
[2/Z] otro_intent - FAIL
  >> Enviado: [mensaje]
  XX Error: [razón del fallo]
```

Los reportes se almacenan en directorios organizados: formato HTML en un directorio de resultados de pruebas y formato PDF en un directorio de exportaciones.

4.2 Relación con conocimientos académicos previos

La ejecución del proyecto requirió la integración de conocimientos del programa de Ingeniería en Electrónica y Telecomunicaciones, énfasis Telemática:

Programación y desarrollo de software

- Patrones de diseño: Page Object Model, Factory para generación de datos, Strategy para reglas de decisión
- Diseño de casos de prueba basados en criterios de aceptación
- Node.js: gestión de módulos, sistema de archivos, variables de entorno.

4.3 Casos presentados, problemas y soluciones

Durante la ejecución surgieron desafíos técnicos que requirieron análisis y soluciones creativas:

Inestabilidad por latencia y sincronización

Problema: Las respuestas del bot tienen tiempos variables dependiendo de carga del servidor, complejidad de consulta y latencia de WhatsApp. Esperas fijas causaban fallos por timeout antes de recibir respuesta o tiempos de ejecución innecesariamente largos.

Solución implementada:

- Esperas explícitas con polling cada 200ms verificando aparición de nuevos mensajes mediante conteo de burbujas entrantes
- Timeout fijo de 45 segundos para todas las respuestas del bot, con esperas adicionales de 5 segundos para asegurar captura completa de mensajes
- Detección de patrones mediante sistema de reglas con expresiones regulares que analizan el contenido de los mensajes recibidos y determinan la acción a ejecutar
- Registro de timestamps UTC en reportes para análisis posterior de tiempos de respuesta

Expiración de sesión de WhatsApp Web

Problema: WhatsApp Web expira sesiones después de períodos de inactividad o por reconexión desde otro dispositivo. Las pruebas fallaban al iniciar sin sesión válida.

Solución implementada:

- Uso de contexto persistente de Playwright (`launchPersistentContext`): almacena automáticamente las cookies, tokens de autenticación y estado de la sesión de WhatsApp Web en un directorio del sistema de archivos (`~/.wapp-autoloop-session`), evitando tener que escanear el código QR en cada ejecución
- Verificación al inicio: comprueba si el usuario ya está autenticado detectando elementos de la interfaz cargada; si no hay sesión activa, muestra el código QR y espera autenticación manual
- El directorio de sesión se reutiliza entre ejecuciones, manteniendo la sesión activa mientras WhatsApp no la expire por inactividad prolongada

Confusión por acumulación de mensajes históricos

Problema: Después de múltiples ejecuciones de pruebas, el chat acumulaba cientos de mensajes. El sistema de conteo y extracción de mensajes se confundía al intentar identificar qué mensajes eran nuevos, causando que las pruebas procesaran respuestas antiguas como si fueran actuales o fallaran al no encontrar el contexto esperado.

Solución implementada:

- Implementación de función `clearChat()` que automatiza la limpieza del historial: localiza el botón de menú del chat, selecciona la opción "Vaciar chat" y confirma la acción
- Limpieza automática al inicio de cada prueba: cada conversación comienza con contexto limpio, eliminando mensajes anteriores

Variaciones de formato en respuestas del bot

Problema: El bot alterna entre formatos de listados (numerados, con bullets, tabulados) y pequeñas variaciones de redacción. Selectores y aserciones rígidos fallaban con cambios menores.

Solución implementada:

- Analizadores basados en expresiones y palabras clave flexibles que detectan intención (ej. “cualquier línea que contenga la palabra clave fertilizante”)
- Sistema de reglas con prioridades: reglas más específicas primero, genéricas como respaldo
- Extracción de opciones dinámica: funciones que identifican el formato y extraen el primer elemento. Entendiendo que los elementos están separados por comas.
- Tolerancia a espacios, mayúsculas/minúsculas y signos de puntuación

Detección de bucles infinitos en conversaciones

Problema: Configuraciones incorrectas de reglas podían causar que el bot y la automatización entraran en bucle infinito (ej. bot pregunta → automación responde “cancelar” → bot pregunta de nuevo → loop).

Solución implementada: Detección de patrones repetitivos: si se ve la misma pregunta del bot 3 veces consecutivas, abortar con error

2	Recibido	Nombre del fabricante del producto que deseas registrar.	2025-10-03T20:21:19.004Z	OK
3	Enviado	AgroChemicals SA	2025-10-03T20:21:19.010Z	OK
4	Recibido	Nombre del fertilizante que deseas registrar.	2025-10-03T20:21:28.344Z	OK
5	Enviado	NPK Completo	2025-10-03T20:21:28.347Z	OK
6	Recibido	Nombre del fertilizante que deseas registrar.	2025-10-03T20:21:37.607Z	OK
7	Enviado	NPK Completo	2025-10-03T20:21:37.612Z	OK
8	Recibido	Nombre del fertilizante que deseas registrar.	2025-10-03T20:21:47.084Z	OK
9	Enviado	NPK Completo	2025-10-03T20:21:47.088Z	OK
10	Recibido	Nombre del fertilizante que deseas registrar.	2025-10-03T20:21:56.565Z	OK
11	Enviado	NPK Completo	2025-10-03T20:21:56.571Z	OK
12	Recibido	Nombre del fertilizante que deseas registrar.	2025-10-03T20:22:06.019Z	OK
13	Recibido	⚠️ Bucle infinito detectado: enviando "NPK Completo" 5 veces consecutivas	2025-10-03T20:22:06.020Z	FAIL

Figure 8. Detección de bucle infinito en reporte

4.4 Resultados y métricas

Al finalizar la pasantía se alcanzaron las siguientes métricas:

- **Cobertura:** 30 especificaciones de prueba automatizadas cubriendo las historias de usuario principales del sistema
- **Intenciones (intents):** 29 intents con múltiples frases de ejemplo cada una (total 942 variaciones de entrada)
- **Reportes generados:** Cada ejecución produce reporte HTML detallado con exportación automática a PDF mediante script dedicado
- **Panel de administración:** Interfaz web para gestión de intents, ejecución selectiva de pruebas y acceso a reportes

4.5 Aprendizajes y recomendaciones

Aprendizajes clave

- La automatización de interfaces conversacionales requiere flexibilidad: parsers rígidos fallan ante cambios menores
- La trazabilidad completa (logs, timestamps, capturas) es esencial para depuración en entornos no determinísticos
- Un panel de administración visual facilita la ejecución de pruebas sin necesidad de conocimientos técnicos profundos
- Los datos de prueba deben ser versionados y aislados para reproducibilidad

Recomendaciones futuras

- **Integración continua:** Integrar la suite en pipelines de integración continua para ejecución automática en cada cambio de código
- **Ejecución paralela:** Explorar uso de múltiples instancias con cuentas distintas para ejecutar pruebas en paralelo y reducir tiempo total
- **Integración con sistema de seguimiento:** Crear tickets automáticamente cuando las pruebas fallan de manera consistente

5 Recursos empleados

A continuación se presenta una descripción detallada de los recursos empleados, organizados por categorías.

5.1 Recursos de hardware

El desarrollo del proyecto se realizó en un equipo personal con las siguientes características:

Table 1. Especificaciones del equipo de desarrollo

Componente	**Especificación**
Equipo	Lenovo IdeaPad 3 14ITL05
Procesador	Intel Core i5-1135G7 (11th Gen) 4 núcleos, 8 hilos Frecuencia base: 2.40 GHz Frecuencia máxima: 4.20 GHz
Memoria RAM	4 GB DDR4
Almacenamiento	SSD NVMe 256 GB Espacio utilizado: 55 GB Espacio disponible: 166 GB
Sistema operativo	Linux (distribución basada en Ubuntu)

5.2 Recursos de software

Entorno de desarrollo y herramientas

Servicios y plataformas

Stack tecnológico

La selección de tecnologías fue **apropiada y bien fundamentada**:

Node.js + TypeScript:

- **Ventajas:** Tipado estático reduce errores, excelente ecosistema de paquetes, compatibilidad nativa con JavaScript del navegador
- **Resultado:** Desarrollo ágil con detección de errores en tiempo de compilación

Playwright:

- **Ventajas:** Contexto persistente integrado (crucial para mantener sesión de WhatsApp), esperas automáticas inteligentes, API moderna y bien documentada, soporte nativo para TypeScript

Table 2. Software y herramientas de desarrollo

Herramienta	**Versión**	**Propósito**
Node.js	v22.18.0	Runtime JavaScript/TypeScript
npm	v10.9.3	Gestor de paquetes
TypeScript	v5.9.2	Lenguaje de programación tipado
Playwright	v1.55.0	Framework de automatización de navegador
Express.js	v4.19.2	Framework web para servidor de administración
ts-node	v10.9.2	Ejecución de TypeScript sin compilación previa
dotenv	v16.4.5	Gestión de variables de entorno
Git	–	Control de versiones
VS Code	–	Editor de código (IDE)
LaTeX	pdflatex	Generación de documentación técnica

Table 3. Servicios y plataformas utilizadas

Servicio	**Uso**
WhatsApp Web	Interfaz de usuario del bot bajo prueba
GitHub	Repositorio de código y control de versiones
Chromium	Motor de navegador para Playwright (incluido en Playwright, modo persistente)

- **Alternativas consideradas:** Selenium (rechazado por complejidad de configuración), Puppeteer (rechazado por falta de contexto persistente robusto)
- **Resultado:** Ideal para el caso de uso; redujo significativamente el tiempo de desarrollo

Express.js:

- **Ventajas:** Framework minimalista, rápido de implementar, ideal para APIs REST simples
- **Resultado:** Panel de administración funcional implementado en menos de una semana

Metodología de trabajo

El enfoque incremental por fases resultó efectivo:

- **Fase 1 (consultas):** Estableció los fundamentos y patrones reutilizables
- **Fase 2 (creaciones):** Reutilizó componentes de Fase 1, acelerando el desarrollo
- **Fase 3 (campañas):** Integró aprendizajes de fases anteriores
- **Panel de administración:** Desarrollado en paralelo, no bloqueó automatización de HUs

Este enfoque permitió entregas parciales funcionales cada semana, facilitando revisión y ajustes tempranos.

5.3 Recursos no utilizados pero considerados

Durante la planificación se consideraron alternativas que finalmente no se implementaron:

- **Docker:** Considerado para estandarizar el entorno de ejecución, pero descartado por la simplicidad del setup en Linux nativo y el overhead de recursos en un equipo con RAM limitada
- **CI/CD (GitHub Actions):** Planificado para integración continua, pero pospuesto por limitaciones de tiempo. Queda como recomendación futura
- **Base de datos:** Inicialmente considerada para almacenar datos de prueba, pero descartada en favor de estructuras de datos en memoria por simplicidad y velocidad
- **Framework de BDD (Cucumber):** Evaluado para especificaciones en Gherkin, pero descartado porque las especificaciones en TypeScript resultaron más mantenibles y no requirieron parser adicional

6 Conclusiones

Se desarrolló un sistema completo de gestión de datos de prueba implementado en el módulo `data.ts`, que incluye:

- 29 intents con 942 variaciones de frases de entrada, proporcionando una cobertura exhaustiva de formulaciones de usuario
- Diccionario centralizado de variables contextuales (cultivos, fertilizantes, clientes, fechas) con funciones de materialización de plantillas
- Sistema de selección aleatoria de datos para evitar colisiones entre ejecuciones
- Funciones de mutación para reintentos automáticos ante conflictos de datos existentes

El sistema implementado elimina la dependencia de archivos externos, favoreciendo estructuras de datos en TypeScript que resultan más mantenibles y menos propensas a errores de sintaxis.

Se automatizaron las tres fases del proyecto con 30 especificaciones de prueba:

- **Fase 1 (Consultas):** 15 HUs automatizadas incluyendo autenticación, listados y filtrados de recursos agrícolas
- **Fase 2 (Creación de productos):** 8 HUs automatizadas con flujos multi-turno de creación y asignación de precios
- **Fase 3 (Campañas y trabajos):** 7 HUs automatizadas con gestión de planificación agrícola completa

La arquitectura implementada basada en Page Object Model y sistema de reglas con expresiones regulares demostró ser altamente reutilizable. El desarrollo de componentes base en Fase 1 permitió acelerar significativamente la automatización de Fases 2 y 3.

Se implementó un sistema de reportes:

- Generador de reportes personalizado (`conversation-reporter.ts`) con línea temporal detallada de cada mensaje enviado y recibido
- Exportación automática a PDF mediante script dedicado que utiliza Playwright para renderizado
- Nomenclatura con marca temporal para trazabilidad histórica
- Estadísticas globales con conteo de eventos exitosos, fallidos e intents ejecutados

Adicionalmente, se desarrolló un panel de administración web, que permite ejecución selectiva de pruebas y gestión de intents mediante interfaz gráfica.

6.1 Conclusiones sobre aspectos técnicos

La decisión de utilizar Playwright en lugar de Selenium o Puppeteer resultó crítica para el éxito del proyecto. Las ventajas observadas incluyen:

- **Contexto persistente nativo:** Se eliminó completamente la necesidad de autenticación manual repetida con código QR, reduciendo el tiempo de setup de varios minutos a segundos
- **Esperas automáticas inteligentes:** Playwright maneja internamente muchas condiciones que requerirían esperas explícitas en Selenium
- **API moderna y tipado robusto:** Integración con TypeScript, reduciendo errores en tiempo de desarrollo

TypeScript sobre JavaScript El uso de TypeScript proporcionó beneficios tangibles:

- Detección de errores en tiempo de compilación antes de ejecutar pruebas
- Autocompletado y documentación inline en el editor, acelerando el desarrollo
- Refactorizaciones seguras con verificación automática de impacto
- Interfaces y tipos personalizados que documentan la estructura de datos

Esta experiencia consolidó varios aprendizajes aplicables a futuros proyectos:

1. **La flexibilidad es crucial en testing de sistemas no determinísticos:** Parsers rígidos y aserciones estrictas fallan ante la variabilidad natural de sistemas conversacionales
2. **La inversión inicial en infraestructura se recupera rápidamente:** El tiempo dedicado a construir utilidades reutilizables en las primeras semanas aceleró significativamente el desarrollo posterior
3. **La trazabilidad completa es esencial para debugging:** Los reportes detallados con timestamps y en cada mensaje intercambiado fueron muy valiosos para diagnosticar fallos
4. **Las herramientas modernas reducen complejidad:** Playwright eliminó gran parte de la complejidad que caracterizaba a Selenium, permitiendo enfoque en lógica de negocio

Agradecimientos y reconocimientos

Se agradece a Grandtek por proporcionar el contexto empresarial real para desarrollar esta pasantía, y a la Universidad del Cauca por la formación académica que hizo posible afrontar los desafíos técnicos con éxito.

7 Recomendaciones

Se recomienda fuertemente la integración de la suite de pruebas automatizadas en un pipeline de integración continua. Actualmente, las pruebas se ejecutan manualmente o mediante el panel de administración, pero vincular la suite a herramientas como GitHub Actions, Jenkins o GitLab CI permitiría la ejecución automática cada vez que se realice un cambio en el código del bot. Esto garantizaría la detección inmediata de regresiones y proporcionaría retroalimentación constante al equipo de desarrollo, reduciendo significativamente el tiempo entre la introducción de un defecto y su identificación.

Relacionado con lo anterior, sería valioso establecer un umbral mínimo de éxito para las pruebas automatizadas como criterio de calidad para despliegues en producción. Por ejemplo, exigir que al menos el noventa por ciento de las pruebas pasen exitosamente antes de autorizar

un despliegue. Esto institucionalizaría la automatización como parte fundamental del proceso de aseguramiento de calidad, evitando que los despliegues se realicen con funcionalidad conocidamente defectuosa.

Se recomienda también ampliar la cobertura de pruebas más allá de los flujos exitosos o "happy paths" actualmente implementados. El sistema de pruebas podría extenderse para validar escenarios de error, como manejo de datos inválidos, campos faltantes, valores fuera de rango y situaciones excepcionales. Por ejemplo, probar qué ocurre cuando el usuario intenta crear un cultivo sin proporcionar todos los campos obligatorios, o cuando ingresa un precio negativo para un producto. Estas pruebas de casos límite y manejo de errores fortalecerían significativamente la robustez del bot y mejorarían la experiencia del usuario final al garantizar respuestas apropiadas incluso ante entradas inesperadas.

8 Anexos

8.1 Anexo A: Repositorio del proyecto

El código completo del proyecto de automatización de pruebas de WhatsApp se encuentra disponible en el repositorio público de GitHub:

Repositorio: `https://github.com/pazussa/twAutomation`

Contenido del repositorio

El repositorio incluye:

- **Código fuente completo:** Todas las especificaciones de prueba (`tests/*.spec.ts`), módulos de utilidades (`tests/setup/`), servidor de administración (`src/admin/`) y scripts auxiliares (`scripts/`)
- **Configuración del proyecto:** Archivos `package.json`, `tsconfig.json`, `playwright.config.ts` con todas las dependencias y configuraciones necesarias
- **Documentación:** Archivo `README.md` con instrucciones detalladas de instalación, configuración y uso del sistema
- **Datos de prueba:** Módulo `tests/setup/data.ts` con los 29 intents y 942 variaciones de frases de entrada
- **Sistema de reportes:** Reportero personalizado (`tests/conversation-reporter.ts`) y script de exportación a PDF (`scripts/export-report-to-pdf.mjs`)
- **Panel de administración:** Código del servidor Express y frontend HTML para gestión visual de intents

Estructura del repositorio

```
twAutomation/
  tests/
    *.spec.ts              (30 especificaciones de prueba)
    setup/
      data.ts              (Intents y datos de prueba)
      flow.ts              (Logica de flujos conversacionales)
      utils.ts             (Utilidades de interaccion WhatsApp)
    conversation-reporter.ts (Generador de reportes)
  src/
    admin/
      server.ts            (Servidor Express)
      public/              (Frontend del panel)
  scripts/
    export-report-to-pdf.mjs
    sync-yml-to-data.mjs
  package.json
  tsconfig.json
  playwright.config.ts
  README.md
```

Instrucciones de uso

Para replicar el entorno de desarrollo y ejecutar las pruebas:

1. Clonar el repositorio: `git clone https://github.com/pazussa/twAutomation`
2. Instalar dependencias: `npm install`
3. Configurar variables de entorno en archivo `.env`
4. Ejecutar pruebas: `npx playwright test`
5. Iniciar panel de administración: `npm run admin`

Consultar el archivo `README.md` del repositorio para instrucciones detalladas de configuración y uso avanzado.

8.2 Anexo B: Información de la empresa

Grandtek - Soluciones Especializadas de Software

La pasantía se desarrolló en Grandtek, empresa especializada en desarrollo de soluciones de software basadas en tecnologías de la información y las comunicaciones (TIC).

Sitio web: https://grandtek.co/

Bibliography

[1] Vahid Garousi and Mika V Mäntylä. Citations, research topics and active countries in software engineering: A bibliometrics study. *Computer Science Review*, 19:56–77, 2016. DOI: 10.1016/j.cosrev.2015.12.002. 1

[2] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Comparing the maintainability of selenium webdriver test suites employing different locators: A case study. In *Proceedings of the 2016 ACM SIGSOFT International Workshop on Software Analytics*, pages 53–59. ACM, 2016. DOI: 10.1145/2989238.2989244. 1

[3] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 272–281. IEEE, 2013. DOI: 10.1109/WCRE.2013.6671303. 1

[4] Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006, 2020. DOI: 10.1016/j.mlwa.2020.100006. 1

[5] Asbjørn Følstad, Theo Araujo, Effie L-C Law, Petter Bae Brandtzaeg, Symeon Papadopoulos, Luis Reis, Marcos Baez, Guy Laban, Patrick McAllister, Carolin Ischen, et al. An experimental study of the effects of an empathetic chatbot on student learning. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–6. ACM, 2018. DOI: 10.1145/3170427.3188553.

[6] Nadia Alshahwan, Xiaocheng Gao, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, Taijin Tei, and Ilya Zorin. Web application testing: A systematic literature review. In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, volume 30, pages 1–63. ACM, 2021. DOI: 10.1145/3450965.

[7] Emil Alégroth, Robert Feldt, and Helena Holmström Olsson. Visual gui testing in practice: challenges, problemsand limitations. In *Empirical Software Engineering*, volume 20, pages 694–744. Springer, 2015. DOI: 10.1007/s10664-013-9293-5.

[8] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M Memon. Using gui ripping for automated testing of android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 258–261. ACM, 2012. DOI: 10.1145/2351676.2351717. 1

[9] Ross Langevin, Ross J Lordon, Talia Avrahami, Benjamin R Cowan, Tad Hirsch, and Gary Hsieh. Testing conversational ai systems: Challenges and opportunities. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–25, 2021. DOI: 10.1145/3449254. 1