

DOCUMENTACIÓN DEL PROYECTO

k0lmenIA

Agentes de QA para Claude Code: diseño de pruebas, automatización con k0lmena e integración con Xray, QMetry, AIO Tests y Azure DevOps — todo en español.

35 agentes

Web · API · Mobile · Performance

Playwright · Cucumber · Appium · k6 · Artillery · JMeter

Xray · QMetry · AIO Tests · Azure DevOps

Jira · Confluence · Figma

PostgreSQL · MySQL · SQL Server · MongoDB

Suites sin tokens

QARMY · Underc0de

github.com/underc0delabs/k0lmenIA

Licencia MIT

Contenido

- 1 Qué es k0lmenIA
- 2 Arquitectura
- 3 Cómo se usa
- 4 Agentes
- 5 Requisitos
- 6 Instalación
- 7 Configuración: el archivo .env
- 8 Investigación de contexto y falta información
- 9 Guía por tarea: diseño y ejecución
- 10 Automatización con k0lmena
- 11 Pruebas de performance
- 12 Gestión de pruebas: Xray, QMetry, AIO Tests y Azure DevOps
- 13 Verificación en base de datos
- 14 Pruebas exploratorias
- 15 Accesibilidad
- 16 Rendimiento web
- 17 Análisis de fallos
- 18 Logs del servidor
- 19 Cobertura y trazabilidad
- 20 Regresión visual (pixel perfect)
- 21 Cross-browser
- 22 Reparación de la automatización
- 23 Selección de casos para un cambio
- 24 Revisión de casos y tiempo estimado
- 25 Charters y sesiones exploratorias
- 26 Correos

27 Traducciones e idiomas

28 Usabilidad

29 Formularios

30 Historial de corridas

31 Seguridad web

32 Conectores MCP

33 Convenciones

34 Estructura del repositorio

35 Problemas frecuentes

36 Extender el proyecto

Qué es k0lmenIA

Un equipo de **agentes especializados en QA** para Claude Code, pensado para profesionales de testing manual: no hace falta programar, se trabaja conversando dentro de VS Code.

Ponés un insumo (una historia, una observación de bug, un contrato de API, o directamente una key de Jira o un link de Figma), pedís lo que necesitás en lenguaje natural y el agente que corresponde genera el resultado: análisis, planes, casos en Excel o Gherkin, datos, bugs, reportes HTML, automatización y carga en la herramienta de gestión.

1 • Diseño

Analizar historias, planificar, escribir y revisar casos manuales, BDD y de API (con su tiempo estimado), generar datos y charters, y redactar bugs.

Queda en `output/`

2 • Automatización

Los agentes *mapper* escriben la automatización una sola vez; después corre con `npm`, sin tokens.

Queda en `herramientas/k0lmena/`

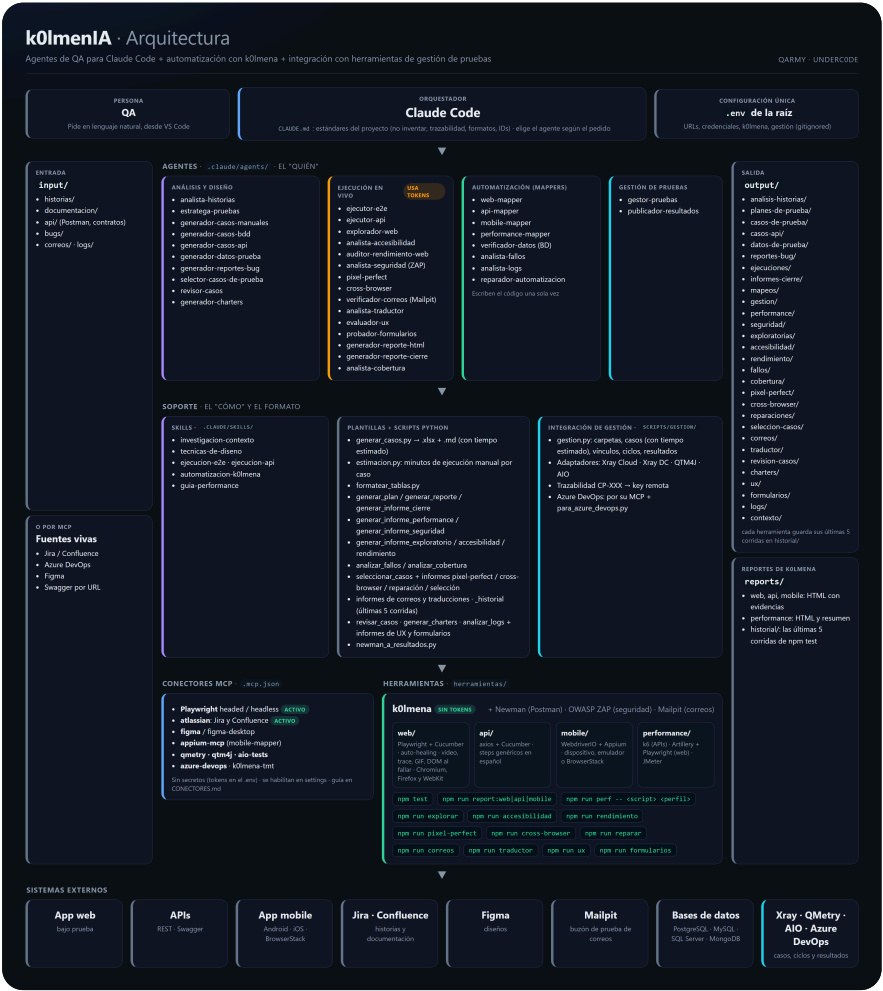
3 • Gestión

Casos, ciclos y resultados con evidencias en Xray, QMetry o AIO Tests (casos también en Azure DevOps), y el informe de cierre.

Tu herramienta + `output/`

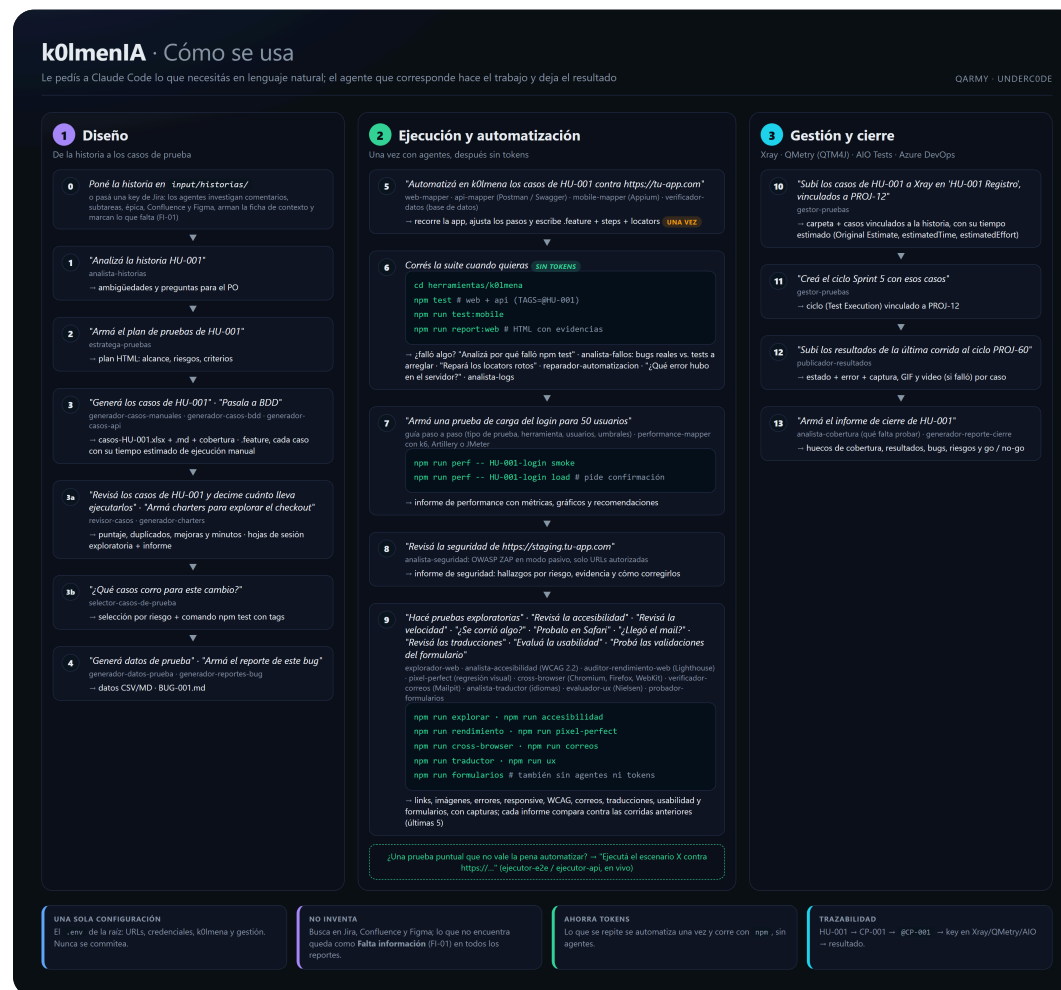
Principios

PRINCIPIO	EN LA PRÁCTICA
No inventar	Si falta un paso, un dato, una regla o un umbral, el agente lo marca y lo pregunta. Mejor un artefacto con huecos señalados que uno completo pero inventado.
Respetar los formatos	Los campos, su orden y sus valores los definen las plantillas y los scripts, no el agente.
Trazabilidad	Todo referencia la historia (<code>HU-XXX</code>); cada escenario automatizado lleva el ID del caso (<code>@CP-001</code>) y la key de la herramienta de gestión.
Cobertura pensada	Positivos, negativos, bordes y validaciones de campos, no solo el camino feliz.
Investigar antes de suponer	Antes de trabajar una historia se revisan sus comentarios, subtareas, épica, issues vinculados, Confluence y Figma. Lo que no aparece queda marcado como Falta información (<code>FI-01</code>).
Ahorro de tokens	Lo que se repite se automatiza una vez y después corre sin agentes.
Seguridad	Los secretos viven en el <code>.env</code> de la raíz, que nunca se versiona; los agentes nunca piden tokens por el chat.



Arquitectura de k0lmenIA · hacé clic para ver en tamaño completo

PIEZA	QUÉ ES	DÓNDE
Claude Code	Orquestador: interpreta el pedido, elige el agente y aplica los estándares del proyecto.	<code>CLAUDE.md</code>
Agentes	El "quién": un especialista por tarea.	<code>.claude/agents/</code>
Skills	El "cómo": conocimiento que se carga cuando hace falta (técnicas de diseño, ejecución E2E y de API, convenciones de k0lmena).	<code>.claude/skills/</code>
Plantillas y scripts	Formato de salida (planilla de casos, cobertura, bug) y reportes HTML determinísticos.	<code>plantillas/</code> · <code>scripts/</code>
Integración de gestión	<code>gestion.py</code> con un adaptador por herramienta: Xray Cloud, Xray Server/DC, QTM4J y AIO Tests. Azure DevOps va por su conector MCP.	<code>scripts/gestion/</code>
Conectores MCP	Playwright y Atlassian (Jira y Confluence) activos; Azure DevOps, Figma, Appium, QMetric, AIO Tests y k0lmenaTMT listos para habilitar. Versiones fijas, sin secretos en el archivo.	<code>.mcp.json</code>
Herramientas	Newman para colecciones de Postman y k0lmena para web, API, mobile y performance.	<code>herramientas/</code>
Entrada y salida	Insumos del usuario y artefactos generados, ordenados por tipo.	<code>input/</code> · <code>output/</code>



Flujo de punta a punta · [hacé clic para ver en tamaño completo](#)

- 1

Poné tus insumos
En `input/`, o pasá una key de Jira, una página de Confluence o un link de Figma con el conector activo. Los agentes investigan el contexto completo y marcan lo que falta (`FI-01`).
- 2

Pedí en lenguaje natural
Claude Code elige el agente. También podés nombrarlo: *"usá el web-mapper para..."*.
- 3

Revisá el resultado
En `output/` o en `herramientas/k0lmena/`.
- 4

Automatizá lo que se repite
Y corrélo con `npm test` todas las veces que quieras, sin tokens.

#	LE PEDÍS	OBTENÉS
0	"Investigá el contexto de PROJ-12"	Ficha de contexto + Falta información
1	"Analizá la historia HU-001"	Ambigüedades y preguntas para el PO
2	"Armá el plan de pruebas de HU-001"	Plan HTML
3	"Generá los casos de HU-001"	<code>casos-HU-001.xlsx</code> + cobertura
4	"Subí los casos a Xray en 'HU-001 Registro', vinculados a PROJ-12, y creá el ciclo Sprint 5"	Casos y ciclo en Xray
5	"Automatizá en k0lmena los casos de HU-001 contra https://tu-app.com"	<code>.feature</code> + steps + locators
6	<code>npm test</code> · <code>npm run report:web</code>	Reporte HTML con evidencias
7	"Subí los resultados al ciclo del Sprint 5"	Estados y evidencias en Xray
8	"Armá una prueba de carga del login para 20 usuarios"	Script k6/Artillery/JMeter + reporte HTML
9	"¿Qué casos corro para el PR #42?"	Selección por riesgo + comando <code>npm test</code> con tags
10	"Corré una regresión visual de https://tu-app.com" · "Probá HU-003 en Chrome, Firefox y Safari"	Qué se corrió y cuántos px · matriz por navegador
11	"Repará los tests que fallaron por locators"	Locators nuevos aplicados y verificados
12	"Verificá que llegue el mail de bienvenida a ana@test.com" · "Revisá las traducciones del sitio en inglés y portugués"	Correos revisados contra lo esperado · porcentaje traducido por idioma
13	"Armá el informe de cierre de HU-001"	Go / no-go

Agentes

Claude Code elige el agente según lo que pidas. Están en `.claude/agents/` y se agrupan en cuatro familias. El catálogo completo (cuándo usar cada uno, pedido de ejemplo, entradas, salidas y qué confirma) está en `AGENTES.md`.

Análisis y diseño

analista-historias
Investiga el contexto de la historia y detecta ambigüedades, vacíos, riesgos y Falta información.

Skills investigacion-contexto
Usa formatear_tablas.py
Conectores atlassian, azure-devops, figma

`output/analisis-historias/`

estratega-pruebas
Plan de pruebas: alcance, riesgos, tipos de prueba, entorno y criterios, en HTML.

Skills investigacion-contexto
Usa generar_plan.py
Conectores atlassian, azure-devops, figma

`output/planes-de-prueba/`

generador-casos-manuales
Casos en Excel y Markdown + informe de cobertura con preguntas para el PO.

Skills investigacion-contexto, tecnicas-de-diseno
Usa generar_casos.py
Conectores atlassian, azure-devops, figma

`output/casos-de-prueba/manuales/`

generador-casos-bdd
Escenarios Gherkin (keywords en inglés, contenido en español) + cobertura.

Skills investigacion-contexto, tecnicas-de-diseno
Usa formatear_tablas.py
Conectores atlassian, azure-devops, figma

`output/casos-de-prueba/bdd/`

generador-casos-api
Casos de API positivos, negativos, de schema y de autorización.

Skills investigacion-contexto, tecnicas-de-diseno
Usa formatear_tablas.py
Conectores atlassian, azure-devops

`output/casos-api/`

generador-datos-prueba
Datos válidos, inválidos y de borde en Markdown o CSV.

Skills investigacion-contexto
Usa formatear_tablas.py
Conectores atlassian, azure-devops

`output/datos-de-prueba/`

selector-casos-de-prueba
¿Qué pruebo con este cambio? Cruza un diff, un PR, issues o notas de la release con todos los casos y arma una selección por riesgo, con el comando `npm test` listo.

Skills investigacion-contexto
Usa seleccionar_casos.py, generar_informe_seleccion.py
Conectores atlassian, azure-devops

`output/seleccion-casos/`

revisor-casos
Revisa casos que ya existen: pasos ambiguos o sin resultado esperado, duplicados, historias sin negativos, puntaje por caso y cuánto lleva ejecutarlos a mano.

Skills investigacion-contexto, tecnicas-de-diseno
Usa revisar_casos.py, generar_informe_revision.py, estimacion.py
Conectores atlassian, azure-devops

`output/revision-casos/`

generador-charters

Pruebas exploratorias por sesiones: charters con misión, foco, heurísticas y tiempo, y el informe de las sesiones a partir de las notas.

Skills investigacion-contexto, tecnicas-de-diseno, ejecucion-e2e
Usa generar_charters.py
Conectores atlassian, azure-devops, playwright

output/charters/

generador-reportes-bug

Reporte de bug profesional según la plantilla, con severidad y prioridad justificadas.

Usa plantilla de bug, formatear_tablas.py
Conectores atlassian, azure-devops

output/reportes-bug/

Ejecución en vivo **USA TOKENS EN CADA CORRIDA**

ejecutor-e2e

Ejecuta casos en un navegador real con Playwright MCP (headed o headless), con evidencia.

Skills ejecucion-e2e
Usa generar_reporte.py
Conectores playwright, playwright-headless

output/ejecuciones/

ejecutor-api

Corre una colección de Postman con Newman.

Skills ejecucion-api
Usa Newman, correr_newman.py

output/ejecuciones/

explorador-web

Pruebas exploratorias: links e imágenes rotas, errores de JavaScript y consola, y responsive en varias resoluciones, con capturas e informe.

Skills ejecucion-e2e
Usa npm run explorar, generar_informe_explorador.io.py
Conectores playwright, playwright-headless

output/exploratorias/

analista-accesibilidad

Auditoría WCAG 2.2 con axe-core, teclado y reflow, más revisión manual: criterios incumplidos, capturas y cómo corregir.

Skills ejecucion-e2e
Usa npm run accesibilidad, generar_informe_accesibilidad.py
Conectores playwright, playwright-headless

output/accesibilidad/

auditor- rendimiento-web

Velocidad de carga con Lighthouse en mobile y desktop: Core Web Vitals, oportunidades con el ahorro estimado y secuencia de carga.

Usa npm run rendimiento (Lighthouse), generar_informe_rendimiento.py

output/rendimiento/

pixel-perfect

Regresión visual contra una referencia (o el diseño): qué se corrió y cuántos px, qué falta, qué cambió de texto o color, con comparador deslizable.

Skills ejecucion-e2e
Usa npm run pixel-perfect, generar_informe_pixel_perfect.py
Conectores playwright, figma

output/pixel-perfect/

cross-browser

Chromium (Chrome y Edge), Firefox y WebKit (Safari): la suite de k0lmena en cada uno o las páginas comparadas lado a lado; separa lo que falla solo en un navegador.

Skills ejecucion-e2e
Usa npm run cross-browser, generar_informe_cross_browser.py
Conectores playwright, playwright-headless

output/cross-browser/

verificador-correos

Correos de la app con un buzón de prueba (Mailpit): que llegue lo esperado, sin variables sin reemplazar, con links e imágenes bien, compatible con Outlook y Gmail, y cómo se ve en desktop y mobile.

Skills investigacion-contexto, ejecucion-e2e, automatizacion-k0lmena
Usa Mailpit, npm run correos, generar_informe_correos.py
Conectores playwright, playwright-headless

output/correos/

analista-traductor

Cada página en cada idioma: textos sin traducir, claves a la vista, caracteres rotos, textos cortados y formatos de número y fecha de otra región, con la traducción propuesta.

Skills investigacion-contexto, ejecucion-e2e

Usa npm run traductor, generar_informe_traductor.py

Conectores playwright, playwright-headless

output/traductor/

evaluador-ux

Usabilidad con las 10 heurísticas de Nielsen: puntaje por heurística, hallazgos automáticos y del recorrido manual con captura, la página 404, desktop y mobile.

Skills ejecucion-e2e, investigacion-contexto

Usa npm run ux, generar_informe_ux.py
Conectores playwright, playwright-headless

output/ux/

probador-formularios

Validaciones de cada campo sin enviar el formulario: obligatorios, largos, formatos y límites, valores inválidos aceptados y válidos rechazados, con el mensaje y la captura.

Skills ejecucion-e2e, investigacion-contexto
Usa npm run formularios, generar_informe_formulario.py

Conectores playwright, playwright-headless

output/formularios/

generador-reporte-html

Dashboard HTML de los resultados de una ejecución.

Usa generar_reporte.py

output/ejecuciones/

generador-reporte-cierre

Informe de cierre de la ronda con recomendación go/no-go.

Usa recolectar_resultados.py, generar_informe_cierre.py

output/informes-cierre/

analista-cobertura

Matriz de trazabilidad (HU → criterios → casos → automatización → resultados) y qué falta probar.

Skills investigacion-contexto

Usa analizar_cobertura.py, generar_informe_cobertura.py

Conectores atlassian, azure-devops

output/cobertura/

Automatización con k0lmena DESPUÉS CORRE SIN TOKENS

web-mapper

Recorre la web siguiendo tus casos y genera `.feature`, steps y locators.

Skills automatizacion-k0lmena, ejecucion-e2e, investigacion-contexto

Usa k0lmena (Playwright + Cucumber)

Conectores playwright, playwright-headless

k0lmena/web/

api-mapper

Lee Postman o Swagger/OpenAPI, verifica los endpoints y genera los `.feature`.

Skills automatizacion-k0lmena, investigacion-contexto, tecnicas-de-diseno

Usa k0lmena (axios + Cucumber)

k0lmena/api/

mobile-mapper

Recorre la app con Appium y genera `.feature`, steps y locators.

Skills automatizacion-k0lmena, investigacion-contexto

Usa k0lmena (WebdriverIO + Appium)
Conectores appium-mcp

k0lmena/mobile/

performance-mapper

Carga, estrés, soak y picos con k6, Artillery o JMeter; pregunta carga y umbrales.

Skills guia-performance, automatizacion-k0lmena

Usa k6, Artillery, JMeter, generar_informe_performance.py

k0lmena/performance/

verificador-datos

Verifica datos en PostgreSQL, MySQL, SQL Server o MongoDB y suma verificaciones de BD a los tests.

Skills automatizacion-k0lmena

Usa npm run bd

Chat · steps en

`.feature`

analista-fallos

Cuando `npm test` falla: separa bugs reales de tests rotos, inestables, de datos o de ambiente, con la evidencia.

Skills ejecucion-e2e

Usa analizar_fallos.py, generar_informe_fallos.py

`output/fallos/`

analista-logs

Logs del servidor (app, nginx, Apache, JSON, syslog) agrupados por firma con su stack trace y cruzados con la corrida: qué error del servidor hubo mientras fallaba cada escenario.

Usa analizar_logs.py, generar_informe_logs.py

`output/logs/`

reparador-automatizacion

Cuando un test falla por un locator o una espera: diagnostica con el DOM guardado al fallar, propone el locator nuevo, lo aplica con respaldo y lo verifica. Si es un bug, no toca el test.

Skills automatizacion-k0lmena, ejecucion-e2e

Usa npm run reparar, generar_informe_reparacion.py

Conectores playwright, playwright-headless

`output/reparaciones/`

analista-seguridad

Escaneo de seguridad web pasivo con OWASP ZAP e informe con hallazgos, evidencia y cómo corregirlos.

Usa OWASP ZAP (Docker), generar_informe_seguridad.py

`output/seguridad/`

Gestión de pruebas

gestor-pruebas

Carpetas, casos, vínculos con la historia y ciclos en Xray, QMetry (QTM4J), AIO Tests o Azure DevOps Test Plans.

Usa scripts/gestion/gestion.py
Conectores azure-devops (Azure Test Plans)

La herramienta +

`output/gestion/`

publicador-resultados

Sube los resultados de `npm test` al ciclo: estado, error y evidencias.

Usa scripts/gestion/gestion.py

La herramienta

Skills: el cómo que usan los agentes

SKILL	QUÉ ENSEÑA	LO USAN

investigacion-contexto	Investigar la historia completa (comentarios, subtareas, épica, Confluence, Figma, contratos) y registrar la Falta información (FI-XX)	analista-historias, estrategia-pruebas, generadores de casos y datos, web-mapper, api-mapper, mobile-mapper, analista-cobertura, selector-casos-de-prueba, verificador-correos, analista-traductor, revisor-casos, generador-charters, evaluador-ux, probador-formularios
tecnicas-de-diseno	Equivalencia, valores límite, tabla de decisión, transición de estados y pairwise	generadores de casos manuales, BDD y API; api-mapper, revisor-casos, generador-charters
ejecucion-e2e	Ejecutar en el navegador con Playwright MCP: modos, snapshots, esperas y evidencia	ejecutor-e2e, web-mapper, explorador-web, analista-accesibilidad, analista-fallos, pixel-perfect, cross-browser, reparador-automatizacion, verificador-correos, analista-traductor, generador-charters, evaluador-ux, probador-formularios
ejecucion-api	Correr colecciones con Newman: base_url, token, validaciones y resultados	ejecutor-api
automatizacion-k0lmena	Convenciones de k0lmena: features, steps, locators, tags, steps de BD y scripts de performance	los mappers, verificador-datos, reparador-automatizacion y verificador-correos
guia-performance	Planificar con la persona una prueba de performance paso a paso	la conversación principal y performance-mapper

Requisitos

Instalá solo lo que vayas a usar.

PARA...	NECESITÁS
Usar los agentes	Claude Code y una cuenta de Claude (Pro, Max, Team o Enterprise) o acceso por API de Anthropic. VS Code recomendado.
Casos, reportes y gestión	Python 3 con <code>openpyxl</code> , <code>tabulate</code> y <code>requests</code> .
Ejecutar E2E en vivo	Node.js 18+ y los navegadores de Playwright.
Ejecutar colecciones de Postman	Newman .
Automatizar con k0lmena	Node.js 20+ .
Automatizar mobile	Node.js 22+ , JDK y Android SDK, o macOS con Xcode para iOS; o una cuenta de BrowserStack.

Instalación

En cinco minutos tenés los agentes andando; k0lmena y el resto se suman cuando los necesites.

k0lmenIA · Puesta en marcha

Instalá, configurá tus tokens en el `.env` y empezá a pedirle cosas a los agentes en español

QARMY · UNDERCODE

1 Poner en marcha

Unos minutos, una sola vez

1 Requisitos

Python 3.10+ · Node.js 20+ (22 LTS recomendado) · VS Code + Claude Code (cuenta de Claude o API de Anthropic). Opcionales: Java 8+ (JMeter), Docker (seguridad), JDK + Android SDK o Xcode (mobile).

2 Cloná el repositorio

```
git clone https://github.com/undercodelabs/k0lmenIA.git
cd k0lmenIA
```

3 Instalá las dependencias

```
pip install -r requirements.txt
# Mac/Linux: pip3

npm install
# instala k0lmena
```

4 Creá tu `.env`

```
cp .env.example .env
# PowerShell: copy
```

Completá solo lo que vayas a usar (columna 2).

5 Revisá que esté todo listo

```
npm run doctor
# dice qué falta y cómo
```

6 Abrió Claude Code

```
code .
claude
# la 1ª vez pide autenticarte
```

Para navegadores extra: `cd herramientas/k0lmena · npx playwright install chromium firefox webkit`

2 Configurar tokens en el `.env`

Un solo archivo, en la raíz · nunca se versiona · los agentes nunca piden tokens por el chat

App bajo prueba

EZE, mappers, exploratorias, visual

```
APP_URL=https://staging.tu-app.com
APP_USER=_ · APP_PASSWORD=_
BASEURL=(si queda vacío, usa APP_URL)
```

API

Newman, suite de API k6

```
API_BASEURL=https://api.tu-app.com · API_TOKEN=ctu-token>
```

Jira y Confluence

conector atlassian (activo)

OAuth desde `/mcp`, o con token:

```
ATLASSIAN_MCP_AUTENTICACION=token
JIRA_BASE_URL · JIRA_EMAIL · JIRA_API_TOKEN
```

Gestión de pruebas

scripts/gestion

```
GESTION_HERRAMIENTA=xray-cloud | xray-dc | qtm4j | aio
GESTION_PROYECTO=PROJ
Xray Cloud: XRAY_CLOUD_CLIENT_ID · XRAY_CLOUD_CLIENT_SECRET
Xray DC: XRAY_DC_URL · XRAY_DC_TOKEN
Qlmetry: QlM4J_API_KEY · AIO: AIO_API_TOKEN
Opcional: ESTIMACION_FACTOR=1.3 (tiempo estimado × 1,3)
```

Azure DevOps

conector azure-devops

```
ADO_ORGANIZACION · ADO_PAT · ADO_PROYECTO
```

Bases de datos y seguridad

```
DB_CONEXIONES=principal + DB_PRINCIPAL_MOTOR · _HOST · _BASE ·
_USUARIO · _CLAVE
SEGURIDAD_URLS_AUTORIZADAS=https://staging.tu-app.com
```

Correos e idiomas

verificador-correos, analista-traductor

```
MAILPIT_URL=http://localhost:8025
TRADUCTOR_IDIOMAS=es-AR,en-US,pt-BR
Mailpit: docker compose -f herramientas/mailpit/docker-
compose.yml up -d
```

Conectores opcionales

(Azure DevOps, Figma, Qlmetry, AIO Tests, k0lmenaMT, Appium): `npm run conector -- activar <nombre>`, reiniciá Claude Code y verificá con `/mcp`. Usá credenciales de prueba, nunca de producción.

3 Pedírselo a los agentes

En lenguaje natural: Claude Code elige el agente (o nombralo vos)

ANÁLISIS Y DISEÑO

"Analizó la historia HU-001 y decime qué falta definir"

analista-historias

"Generá los casos de prueba de PROJ-12"

generador-casos-manuales · Excel con tiempo estimado + cobertura

"Revisá los casos de HU-001 y decime cuánto lleva ejecutarlos"

revisor-casos · **NUEVO** · puntaje, duplicados y minutos

"¿Qué casos corro para el PR #42?"

selector-casos-de-prueba · selección por riesgo + comando

EJECUCIÓN EN VIVO

"Corré una regresión visual de qarmyar"

pixel-perfect · qué se corrió y cuántos px

"Evaluá la usabilidad del checkout"

evaluador-ux · **NUEVO** · heurísticas de Nielsen

"Probá las validaciones del formulario de registro"

probador-formularios · **NUEVO** · campo por campo, sin enviarlo

"Verificá que llegue el mail de bienvenida a ana@test.com"

verificador-correos · con Mailpit

"Hacé pruebas exploratorias del sitio del .env"

explorador-web · también accesibilidad y velocidad

AUTOMATIZACIÓN CON K0LMENA

"Automatizá en k0lmena los casos de HU-001"

web-mapper · después `npm test`, sin tokens

"Repará los tests que fallaron por locators"

reparador-automatizacion · propone, aplica y verifica

GESTIÓN Y CIERRE

"Subí los casos de HU-001 a Xray y creá el ciclo Sprint 5"

gestor-pruebas

"Armá el informe de cierre de HU-001"

generador-reporte-cierre · go / no-go

SI FALTA UN DATO

El agente no inventa: devuelve "Necesito que confirmes" y Claude te pregunta.

ACCIONES CON EFECTO

Escribir en una base, crear datos en Jira/Xray o correr carga: siempre con tu ok.

AHORRA TOKENS

Lo automatizado corre con `npm test`, sin agentes.

TODO QUEDA EN

output/ (reportes HTML, con las últimas 5 corridas en `historial/`) y herramientas/k0lmena/ (automatización).

Puesta en marcha: instalación, tokens del `.env` y pedidos de ejemplo · hacé clic para ver en tamaño completo

1 Instalá Claude Code

```
curl -fsSL https://claude.ai/install.sh | bash # macOS / Linux
npm install -g @anthropic-ai/claude-code # alternativa (con Node.js 22 LTS cubrís todo)
claude --version
```

En Windows, seguí la [guía oficial](#).

k0lmenIA · Documentación

15 / 52

2

Cloná el repositorio

```
git clone https://github.com/underc0delabs/k0lmenIA.git
cd k0lmenIA                                # o la carpeta donde lo clonaste
pip install -r requirements.txt              # Mac/Linux: pip3
npm install                                 # k0lmena
cp .env.example .env                       # y completalo (PowerShell: copy)
```

En Windows el comando es `python`; en Mac y Linux, `python3` y `pip3` (si `pip3` responde *externally-managed-environment*, usá un entorno virtual: `python3 -m venv .venv`).

3

Revisá que esté todo listo

```
npm run doctor                                # dice qué falta y cómo resolverlo
```

4

Abri VS Code y lanzá Claude Code

```
code .
claude                                     # la primera vez pide autenticarte
```

5

Sumá lo opcional

```
npx playwright install                      # ejecución E2E en vivo
npm install -g newman                      # colecciones de Postman
cd herramientas/k0lmena
npx playwright install chromium            # + firefox webkit para el cross-browser
npm run bootstrap:k6                       # performance de APIs con k6
npm run bootstrap:jmeter                  # performance con JMeter (Java 8+)
```

Configuración: el archivo .env

Hay **un solo** `.env`, **en la raíz del repo**. Lo usan los agentes, k0lmena y los scripts de gestión. Está en `.gitignore` y la plantilla comentada es `.env.example`.

GRUPO	VARIABLES	PARA QUÉ
App bajo prueba	APP_URL APP_USER APP_PASSWORD	Ejecución E2E y logins
API	API_TOKEN API_BASEURL	Newman, suite de API y k6
Web	BASEURL BROWSER HEADLESS VIEWPORT_WIDTH VIEWPORT_HEIGHT LOCALE TIMEZONE	Navegador de la suite web (vacías: Chromium, Firefox y WebKit a 1366×768)
Ejecución	TAGS PARALLEL	Qué escenarios y cómo se corren
Evidencias	EVIDENCE (captura · ambos · off) VIDEO TRACE	Qué se adjunta al reporte
Auto-healing	KOLMENA_AUTO_HEALING	Recuperar locators rotos
Mobile	MOBILE_TARGET MOBILE_PLATFORM MOBILE_DEVICE_NAME MOBILE_APP MOBILE_UDID BROWSERSTACK_*	Dónde corre la suite mobile
Performance	PERF_VUS PERF_DURACION	Pisar la carga de los scripts k6
Bases de datos	DB_CONEXIONES + DB_<NOMBRE>_MOTOR _HOST _PUERTO _BASE _USUARIO _CLAVE (o _URL) _ESCRITURA _PRODUCCION	Verificación en base de datos
Gestión	GESTION_HERRAMIENTA GESTION_PROYECTO + credenciales de la herramienta	Xray, QMetry y AIO Tests
Azure DevOps (MCP)	ADO_ORGANIZACION ADO_PAT ADO_PROYECTO	Boards, Wiki y Test Plans
Correos	MAILPIT_URL MAILPIT_USUARIO MAILPIT_CLAVE CORREO_TIMEOUT_S	Buzón de prueba Mailpit (verificador-correos y steps de correo)
Idiomas	TRADUCTOR_IDIOMAS TRADUCTOR_URL_IDIOMA	Idiomas a revisar y cómo se cambia de idioma (analista-traductor)
Tiempo estimado	ESTIMACION_FACTOR GESTION_TIEMPO_NATIVO	Opcionales: multiplicar el tiempo estimado de los casos (por ejemplo 1,3) y no usar el campo de tiempo de la herramienta de gestión (0)

! Usá credenciales de un entorno de prueba, nunca de producción. Los agentes leen los secretos del `.env` : nunca pegues un token en el chat ni lo escribas en un archivo versionado.

Carpetas de entrada

CARPETA	QUÉ VA
input/historias/	Historias de usuario con sus criterios de aceptación (HU-001.md , ...).
input/documentacion/	Documentación del producto que sirva de contexto.
input/api/	Contratos, colecciones de Postman y environments.
input/bugs/	Observaciones sueltas de errores para convertir en reportes.
input/correos/	Lo que tiene que llegar por mail en cada caso (espera-HU-001.json , para el verificador-correos).
input/logs/	Logs de la aplicación para el analista-logs (app.log , access.log). No se versionan.

Investigación de contexto y "Falta información"

Antes de analizar, planificar, escribir casos o automatizar, los agentes **investigan el contexto completo** de la historia en lugar de suponer. Lo que no encuentran en ninguna fuente queda marcado como **Falta información** en todos los reportes.

Dónde buscan

FUENTE	QUÉ REVISAN
Historia de Jira	Descripción, criterios, estado, labels, componentes y adjuntos.
Comentarios	Todos. Las decisiones que se tomaron después ("acordamos que el límite es 50") reemplazan a la descripción, citando quién y cuándo.
Subtareas, épica e issues vinculados	Detalle de cada subtarea, reglas generales e historias hermanas de la épica, dependencias y bugs conocidos del flujo.
Confluence	Páginas enlazadas o, si no hay, las más relevantes del espacio del proyecto.
Figma	Textos exactos, mensajes de error, estados y campos obligatorios.
Contratos e <code>input/</code>	Swagger, Postman y documentación local.

El resultado es una **ficha de contexto** reutilizable, `output/contexto/contexto-HU-XXX.md` : fuentes consultadas, hallazgos por criterio, decisiones, contradicciones y lo que falta. Si la historia no cambió, los demás agentes la reusan sin volver a investigar. Si un conector no está activo, la fuente queda como "No disponible" en lugar de suponerla.

PEDIDO · *Investigá el contexto de PROJ-12 y generá los casos de prueba.*

Falta información (FI-XX)

Cada dato que no aparece en ninguna fuente (o aparece contradictorio) recibe un ID por historia, con qué falta, dónde se buscó, a qué afecta y la pregunta para el PO. El mismo ID viaja a todos los artefactos. La app no es fuente de requisitos: si el resultado esperado depende de un dato faltante, el escenario no se "acomoda" a lo que hace la aplicación; se automatiza con el supuesto marcado o queda `@bloqueado` hasta que el PO responda.

DÓNDE	CÓMO SE VE
Planilla de casos	Casos resaltados en ámbar, etiqueta <code>@falta-info</code> , detalle en Comentarios y hoja Falta información .
Análisis y coberturas	Sección Falta información ; criterios con cobertura <code>Parcial (falta información)</code> .
<code>.feature</code> de k0lmena	Tags <code>@falta-info @FI-01</code> y comentario <code># FALTA INFORMACIÓN (FI-01): ...</code> .
Reporte de mapeo	Estado <code>Falta información</code> . El mapper no usa la app para completar el dato: lo que muestra la app queda como observación.
Reportes de k0lmena y de ejecución	Nota visible en el escenario y aviso en el resumen.
Informe de cierre	Sección Falta información ; los ítems abiertos cuentan como riesgo para el go/no-go.
Xray · QMetric · AIO	El comentario del resultado aclara que el caso depende de información faltante.

Diseño y ejecución

Qué pedir, qué hace cada agente y qué deja.

Analizar una historia ANALISTA-HISTORIAS

PEDIDO · Analizá la historia HU-001 y decime qué ambigüedades tiene.

Revisa cada criterio y entrega `analisis-HU-001.md` con ambigüedades, vacíos, riesgos, casos no contemplados y preguntas para el PO.

Armar el plan de pruebas ESTRATEGA-PRUEBAS

PEDIDO · Armá el plan de pruebas de HU-001.

Dashboard HTML con alcance, objetivos, tipos de prueba, riesgos, datos y entorno, y criterios de entrada y salida.

Generar casos manuales GENERADOR-CASOS-MANUALES

PEDIDO · Generá los casos de prueba de HU-001.

Entrega `casos-HU-001.xlsx`, `casos-HU-001.md` y `casos-HU-001-cobertura.md`. Cubre camino feliz, negativos, bordes y validaciones. Cada caso lleva su **tiempo estimado de ejecución manual**, con el total al pie (ver [Revisión de casos y tiempo estimado](#)). Esa planilla es la que después se sube a la herramienta de gestión o se automatiza.

Generar escenarios BDD GENERADOR-CASOS-BDD

PEDIDO · Pasá la historia HU-001 a escenarios BDD.

Genera `HU-001-<slug>.feature`, con un tag `@CP-XXX` por escenario, y `HU-001-cobertura.md`, con el tiempo estimado si se ejecutan a mano.

Generar casos de API GENERADOR-CASOS-API

PEDIDO · Generá los casos de la API de autenticación a partir de `input/api/auth-endpoints.md`.

Tabla resumen y detalle por caso: método, endpoint, headers, body, status y schema esperado.

Datos de prueba y bugs GENERADOR-DATOS-PRUEBA GENERADOR-REPORTES-BUG

PEDIDO · Generá datos de prueba para el formulario de registro, en CSV.

Datos válidos, inválidos y de borde. Los bugs siguen `plantillas/plantilla-reporte-bug.md` (editala para usar tu formato) y marcan lo que falta en vez de inventarlo.

Ejecutar en vivo

[EJECUTOR-E2E](#)[EJECUTOR-API](#)[USA TOKENS](#)

PEDIDO · Ejecutá SOLO el escenario de registro válido de HU-001 contra <https://tu-app.com>.

El ejecutor E2E pregunta si querés ver el navegador, ejecuta con Playwright MCP, saca evidencia y genera `reporte-HU-001-<fecha-hora>.html`. Para recorrer un sitio entero buscando fallas, está el **explorador-web** (ver [Pruebas exploratorias](#)). El de API corre la colección con Newman. Ideal para pruebas puntuales: lo que se repite conviene automatizarlo.

Reportes e informe de cierre

[GENERADOR-REPORTE-HTML](#)[GENERADOR-REPORTE-CIERRE](#)

PEDIDO · Armá el informe de cierre de las pruebas de HU-001.

Dashboards con indicadores, gráficos y detalle; el informe de cierre suma bugs abiertos, riesgos y la recomendación go/no-go.

Automatización con k0lmena

El framework integrado en `herramientas/k0lmena/`. Un agente *mapper* recorre la aplicación **una sola vez** y escribe la automatización; desde ahí la suite corre con `npm`, sin agentes y sin tokens.



Web

Playwright + Cucumber, video, trace y GIF. Con `K0LMENA_AUTO_HEALING=on` recupera locators rotos.



API

axios + Cucumber con steps genéricos en español: no hace falta programar.



Mobile

WebdriverIO + Appium en dispositivo, emulador o BrowserStack.



Performance

k6 para APIs, Artillery + Playwright para flujos web y JMeter para quien ya trabaja con .jmx.

i Las carpetas de k0lmena vienen vacías: todo lo que hay adentro lo generan los mappers para tu aplicación.

web-mapper

PEDIDO · Automatizá en k0lmena los casos de output/casos-de-prueba/manuales/casos-HU-001.xlsx contra <https://tu-app.com>.

1. Lee los casos (manuales o BDD).
2. Navega la web real con Playwright MCP y comprueba que cada paso se puede ejecutar.
3. Genera `web/features/HU-001-<slug>.feature`, `web/steps/HU-001.steps.ts` y `web/locators/HU-001.locators.ts`, reutilizando steps existentes.
4. Valida con `npm run test:web` y deja `output/mapeos/mapeo-HU-001-web.md`.

Si un paso no se puede ejecutar, el escenario queda `@bloqueado` con el motivo y no corre hasta que se resuelva.

api-mapper

PEDIDO · Pasá a k0lmena los endpoints de /pet del Swagger <https://petstore.swagger.io/v2/swagger.json>.

Lee Postman (v2.x) o Swagger/OpenAPI, te deja elegir los endpoints, diseña escenarios positivos, negativos y de autorización, los verifica contra la API real y escribe los `.feature` con los steps genéricos. Antes de un POST, PUT, PATCH o DELETE pide autorización.

mobile-mapper

PEDIDO · Automatizá en k0lmena los casos de login de la app en el emulador.

Necesita el conector Appium MCP y la app en `mobile/apps/` (no se versiona). Después la suite corre sin MCP según `MOBILE_TARGET`:

DESTINO	QUÉ NECESITÁS
<code>device</code>	Dispositivo por USB con depuración y <code>MOBILE_UDID</code> (de <code>adb devices</code>).
<code>emulator</code>	Emulador o simulador encendido y <code>MOBILE_DEVICE_NAME</code> .
<code>browserstack</code>	<code>BROWSERSTACK_USER</code> , <code>BROWSERSTACK_KEY</code> y <code>BROWSERSTACK_APP</code> .

Correr la suite

```
cd herramientas/k0lmena
npm test                # web + API
npm run test:web        # también: test:api · test:mobile · test:all
TAGS=@HU-001 npm test  # filtrar por tag (bash)
$env:TAGS="@HU-001"; npm test # filtrar por tag (PowerShell)
npm run report:web      # también: report:api · report:mobile
```

Si `EVIDENCE` está vacía, `npm test` pregunta al arrancar qué guardar de los tests que pasan: solo captura, o captura + GIF del recorrido.

Evidencias en el reporte

SUITE	ESCENARIO QUE PASA	ESCENARIO QUE FALLA
Web	Captura final (+ GIF opcional)	Video completo, captura, error con stack, URL y título, logs del navegador (consola, errores JS, requests fallidos, HTTP $\geq$ 400), logs de Node, trace de Playwright y el DOM de la página al fallar (<code>reports/web/dom/</code>), lo usa el reparador)
API	Request y response (el <code>Authorization</code> se oculta)	Lo mismo, más la validación que falló
Mobile	Captura final (+ GIF opcional)	Video completo, captura, page source y logs del dispositivo

Herramientas extra

COMANDO	PARA QUÉ
<code>npm run crawler [url]</code>	Genera un archivo de locators recorriendo una página.
<code>npm run link-tester</code>	Busca enlaces e imágenes rotas en <code>BASEURL</code> .
<code>npm run explorar</code>	Recorrido exploratorio con navegador: links, imágenes, errores, responsive y tiempos de carga, con capturas, videos e informe HTML (lo usa el explorador-web).
<code>npm run accesibilidad</code>	Auditoría WCAG 2.2 con axe-core, navegación con teclado y reflow a 320px, con capturas, video e informe HTML (lo usa el analista-accesibilidad).
<code>npm run rendimiento</code>	Lighthouse en mobile y desktop: Core Web Vitals, oportunidades de mejora y secuencia de carga, con informe HTML (lo usa el auditor-rendimiento-web).

<code>npm run pixel-perfect</code>	Regresión visual contra la referencia: píxeles y elementos corridos, faltantes, nuevos o con otro estilo, con informe HTML (lo usa el pixel-perfect).
<code>npm run cross-browser</code>	La suite web o las páginas en Chromium, Firefox y WebKit, con matriz y comparación lado a lado (lo usa el cross-browser).
<code>npm run reparar</code>	Diagnostica los locators rotos con el DOM guardado al fallar, propone, aplica y verifica (lo usa el reparador-automatizacion).
<code>npm run correos</code>	Revisa los correos del buzón de prueba Mailpit contra lo esperado, con informe HTML (lo usa el verificador-correos).
<code>npm run traductor</code>	Cada página en cada idioma: textos sin traducir, cortados y formatos, con informe HTML (lo usa el analista-traductor).
<code>npm run ux</code>	Usabilidad con las 10 heurísticas de Nielsen en desktop y mobile, con la página 404, capturas e informe HTML (lo usa el evaluador-ux).
<code>npm run formularios</code>	Validaciones de cada campo sin enviar el formulario, con matriz campo × prueba, capturas e informe HTML (lo usa el probador-formularios).
<code>npm run record</code>	Graba un flujo con Playwright codegen.
<code>npm run debug</code>	Corre la suite web en modo debug.

Pruebas de performance

El **performance-mapper** arma pruebas de carga, estrés, soak y picos. Nunca usa valores por defecto: la carga y los umbrales los definís vos.

! Una prueba de carga genera tráfico real. Corredla solo contra ambientes de prueba o con autorización de los dueños del sistema.

performance-mapper

PEDIDO · Armá una prueba de carga de los endpoints de /pet para 50 usuarios.

1. **Te guía paso a paso**, una pregunta por vez y con una sugerencia justificada en cada una: objetivo y tipo de prueba, herramienta, ambiente (y que no sea producción) y endpoints, usuarios concurrentes, rampa, duración y tiempo de pensamiento (te ayuda a calcularlos desde el volumen de negocio), umbrales y perfiles a correr.
2. **Deja el plan** en `output/performance/<HU>/` y te pide confirmación.
3. **Te recomienda la herramienta** (la decisión es tuya): k6 para APIs (escala a miles de usuarios) Artillery + Playwright para flujos de navegador (pocos usuarios: cada uno es un Chromium) o JMeter si tu equipo ya lo usa, traés un `.jmx` o necesitás otros protocolos.
4. **Escribe el script** una vez y lo valida con un smoke.
5. **Corre con carga** solo con tu confirmación explícita para cada corrida.
6. **Arma el informe de performance**: dashboard con veredicto, capacidad observada, métricas y gráficos por corrida, umbrales, p95 por endpoint, hallazgos y recomendaciones.

Comandos

```
npm run perf # lista los scripts
npm run perf -- <script> smoke # valida el script con carga mínima
npm run perf -- <script> load # carga objetivo (pide confirmación)
npm run perf -- <script> stress --confirmar # sin pregunta: CI o con autorización
npm run perf -- <script> load --vus 20 --duracion 2m # k6 y JMeter: pisa la carga
```

PERFIL	QUÉ HACE
<code>smoke</code>	Carga mínima para validar que el script funciona.
<code>load</code>	Sube hasta la carga objetivo, la sostiene y baja.
<code>stress</code>	1x, 2x y 3x la carga objetivo: busca el punto de quiebre.
<code>soak</code>	Carga objetivo durante mucho tiempo: fugas de memoria y degradación.
<code>spike</code>	Salto brusco a un pico y vuelta: cómo se recupera.

Cada corrida deja en `reports/performance/<k6|artillery|jmeter>/` un reporte HTML con el veredicto, requests, req/s, errores, latencias p50/p95/p99, umbrales con el valor medido, detalle por endpoint o paso, códigos de respuesta, checks y, en Artillery, la evolución en el tiempo. Si un umbral no se cumple, el comando termina con error.

Xray, QMetry, AIO Tests y Azure DevOps

gestor-pruebas y **publicador-resultados** trabajan con Xray Cloud, Xray Server/Data Center, QMetry para Jira (QTM4J) y AIO Tests, con los mismos comandos para todas.

Se configura en el `.env`: `GESTION_HERRAMIENTA` (`xray-cloud`, `xray-dc`, `qtm4j` o `aio`), `GESTION_PROYECTO` (la key del proyecto de Jira) y las credenciales de tu herramienta, explicadas en `.env.example`.

gestor-pruebas

PEDIDO · Subí los casos de HU-001 a Xray en la carpeta 'HU-001 Registro', vinculados a PROJ-12, y creá el ciclo Sprint 5.

Crea la carpeta, sube los casos (manuales desde el `.xlsx` o BDD desde el `.feature`), los vincula a la historia, crea ciclos y les agrega casos. La primera vez hace una prueba en seco (`--dry-run`). Guarda la trazabilidad en `output/gestion/HU-001-<herramienta>.json`: un caso nunca se sube dos veces.

i Tiempo estimado. Cada caso se sube con su tiempo de ejecución manual: en la descripción y en el campo de tiempo de la herramienta — *Original Estimate* de Jira en Xray (si el proyecto tiene time tracking), `estimatedTime` en QTM4J y `estimatedEffort` en AIO Tests. En Azure DevOps viaja en el JSON de `para_azure_devops.py` y como etiqueta `tiempo-estimado-Xmin`. Si la herramienta rechaza el campo, el caso se crea igual con el tiempo en la descripción (`GESTION_TIEMPO_NATIVO=0` lo desactiva). Ver [la fórmula](#).

publicador-resultados

PEDIDO · Subí los resultados de la última corrida web al ciclo PROJ-60.

Asocia cada escenario a su caso por el tag `@CP-XXX` y carga el estado, un comentario con el error y las evidencias: captura siempre, GIF si se generó y video solo si falló. Nunca asigna un resultado por aproximación.

Equivalencias

CONCEPTO	XRAY	QTM4J	AIO TESTS
Caso	Issue Test (Manual o Cucumber)	Test case	Caso (Classic o BDD)
Ciclo	Test Execution	Test cycle	Test cycle
Caso ↔ historia	Link "Test" (cobertura)	Requirement link	jiraRequirementIDs
Ciclo ↔ historia	Link "Relates"	Requirement link	jiraTaskIDs

Comandos manuales

```
python scripts/gestion/gestion.py --dry-run carpeta --ruta "HU-001 Registro"
python scripts/gestion/gestion.py subir-casos --origen <casos.xlsx> --carpeta "HU-001 Registro" --historia PROJ-12 --traza HU-001
python scripts/gestion/gestion.py crear-ciclo --nombre "Sprint 5" --historia PROJ-12 --traza HU-001
python scripts/gestion/gestion.py extraer-resultados --reporte <cucumber-report.json>
python scripts/gestion/gestion.py publicar-resultados --ciclo PROJ-60 --resultados <resultados.json> --traza HU-001
```

Las carpetas van sin barra inicial (en Git Bash, `/HU-001` se convierte en una ruta de Windows). Referencia completa en `scripts/gestion/README.md`.

Azure DevOps (Test Plans)

Azure DevOps no pasa por `gestion.py`: se integra por su conector MCP oficial (`azure-devops`), con `ADO_ORGANIZACION`, `ADO_PAT` y `ADO_PROYECTO` en el `.env`. Los agentes leen las historias (con comentarios, tareas, padre y vínculos) y la Wiki, y **gestor-pruebas** crea los *Test Cases* con sus pasos (los arma `scripts/gestion/para_azure_devops.py`), los vincula a la historia y los agrega a una suite. Todavía no se publican resultados de ejecución en Azure DevOps.

PEDIDO · Subí los casos de *casos-HU-001.xlsx* a Azure Test Plans, plan 'Sprint 5', suite 'HU-001 Registro', vinculados a la historia 1234.

Verificación en base de datos

El **verificador-datos** revisa lo que las pruebas no ven: qué quedó guardado, en qué estado y con qué valores. Soporta **PostgreSQL**, **MySQL / MariaDB**, **SQL Server** y **MongoDB**, con varias conexiones con nombre.



A pedido

"Verifiqué si existe el usuario ana@test.com y en qué estado quedó." Mira el esquema, arma la consulta con parámetros y te responde.



Dentro de los tests, si lo pedís

Suma steps de base de datos a los `.feature` de web, API o mobile solo cuando se lo pedís; después corren con `npm test`, sin tokens.



Seguro por defecto

Solo lectura, en transacciones que se descartan; datos sensibles enmascarados. Escribir requiere habilitarlo y confirmar cada operación.

Configuración

```
DB_CONEXIONES=principal,pagos           # la primera es la de por defecto
DB_PRINCIPAL_MOTOR=postgres              # postgres | mysql | mariadb | sqlserver | mongodb
DB_PRINCIPAL_HOST=localhost
DB_PRINCIPAL_PUERTO=5432
DB_PRINCIPAL_BASE=tienda
DB_PRINCIPAL_USUARIO=qa_lectura
DB_PRINCIPAL_CLAVE=...
DB_PRINCIPAL_ESCRITURA=no               # si = permite escribir con confirmación
DB_PRINCIPAL_PRODUCCION=no              # si = nunca escribe
```

Steps para los tests

i Solo cuando lo pedís. Los mappers (web, API y mobile) no agregan verificaciones de base de datos por su cuenta: si un caso menciona datos guardados, lo dejan como sugerencia en el reporte de mapeo. El verificador-datos, ante una consulta puntual, responde sin tocar los `.feature`.

```
When envío un POST a "/usuarios" con el body: ...
And guardo el campo "id" de la respuesta como "idUsuario"
Then existe en la base de datos un registro en "usuarios" donde "id" es "{idUsuario}"
And el registro de "usuarios" donde "id" es "{idUsuario}" tiene:
  | campo | valor |
  | estado | ACTIVO |
  | baja   | (nulo) |
And la base de datos "pagos" tiene 1 registro en "movimientos" donde:
  | campo      | valor      |
  | usuario_id | {idUsuario} |
```

También: `no existe en la base de datos ...`, `consulto en la base de datos:` (SQL, o JSON en MongoDB), `la consulta devuelve N filas`, `el campo "x" de la consulta es "y"` y `guardo el campo "x" de la consulta como "var"`. Si el dato se guarda de forma asíncrona, los steps reintentan hasta `DB_ESPERA_MS`. Cada verificación deja en el reporte la consulta y las filas encontradas.

Sin agente

```
cd herramientas/k0lmena
npm run bd -- conexiones           # lista y prueba las conexiones
npm run bd -- esquema --tabla usuarios   # columnas de una tabla
npm run bd -- existe --tabla usuarios --donde email=ana@test.com
npm run bd -- consultar --sql "SELECT id, estado FROM usuarios WHERE email = ?" --param ana@test.com
```

! Para preparar o limpiar datos, la conexión necesita `DB_<NOMBRE>_ESCRITURA=si` y el agente muestra la sentencia y pide confirmación antes de cada operación. Con `DB_<NOMBRE>_PRODUCCION=si` nunca escribe. Recomendado: un usuario de base de datos con permisos de solo lectura.

Pruebas exploratorias

El **explorador-web** recorre tu sitio (el `APP_URL` del `.env` o la URL que le digas) y busca lo que se rompe sin que nadie lo note. Primero un **recorrido automático** con un script, sin gastar tokens por página; después explora a mano con Playwright MCP lo que necesita criterio.

QUÉ REVISAS	CÓMO
Links e imágenes	Links rotos internos y externos (aparte, los que no se pueden verificar porque el sitio bloquea robots), imágenes que no cargan e imágenes sin <code>alt</code> .
Errores	Errores de JavaScript, errores de consola y recursos que fallan (scripts, CSS, fetch con 4xx/5xx o sin respuesta).
Responsive	En 360×740, 390×844, 768×1024, 1366×768 y 1920×1080: scroll horizontal y qué elemento se sale, texto de menos de 12px, objetivos táctiles chicos y muy juntos (WCAG 2.5.8) y falta de <code>meta viewport</code> .
Rendimiento	Tiempo de carga, primer byte, DOM listo, requests y peso de cada página; las lentas son hallazgos.
Exploración manual	Menús (también el mobile), formularios sin enviar, búsquedas, la página 404, recargar a mitad de un flujo. Con misión y tiempo acotado.

explorador-web USA TOKENS

PEDIDO · *Hacé pruebas exploratorias del sitio del `.env`.*

1. **Recorre** el sitio con `npm run explorar`: captura cada página y cada resolución con los problemas **marcados en rojo** y graba un video del recorrido y de cada resolución.
2. **Revisa** los hallazgos: reintenta los dudosos, mira las capturas y ajusta la severidad con su motivo.
3. **Explora a mano** con Playwright MCP lo que un recorrido automático no ve.
4. **Arma el informe** HTML: indicadores, gráficos, cada hallazgo con su captura (clic para ampliar), tiempos de carga, matriz de responsive, videos, links rotos, detalle por página y recomendaciones.

```
npm run explorar -- --url https://staging.mi-app.com --max-paginas 50 # sin agentes ni tokens
npm run explorar -- --resoluciones 360x740,768x1024,1366x768 --video no
```

i Solo navega con GET: no envía formularios y no abre ni verifica links de cierre de sesión o borrado. Todo queda en `output/exploratorias/<HU o sitio>/` y la corrida anterior pasa a `historial/` (se guardan las últimas 5) y el informe la compara.

Accesibilidad

El **analista-accesibilidad** audita tu sitio contra **WCAG 2.2** (nivel AA por defecto): primero una auditoría automática, sin gastar tokens por página, y después una revisión manual de lo que una herramienta no puede juzgar.

QUÉ REVISAS	CÓMO
Reglas WCAG	axe-core en cada página, con los textos en español: contraste, textos alternativos, etiquetas de formularios, nombres de botones y links, idioma, estructura, ARIA. También las buenas prácticas.
Teclado	Recorre la página con Tab: elementos que reciben foco sin indicador visible (2.4.7), trampas de foco (2.1.2) y enlace para saltar al contenido (2.4.1). Muestra el orden del foco.
Reflow	A 320px de ancho no tiene que haber scroll horizontal (1.4.10, equivale a un zoom del 400 %).
Revisión manual	Con el snapshot de accesibilidad de Playwright (lo que "oye" un lector de pantalla): si los textos alternativos describen, encabezados y regiones, formularios y mensajes de error, menús y modales con teclado, zoom, color y movimiento.

analista-accesibilidad USA TOKENS

PEDIDO · *Revisá la accesibilidad del sitio del .env.*

1. **Audita** con `npm run accesibilidad`: captura cada página con los elementos que fallan **marcados en rojo**, cada elemento por separado y un video del recorrido.
2. **Revisa** los hallazgos: confirma con las capturas, ajusta la severidad y decide lo que axe no pudo.
3. **Revisa a mano** con Playwright MCP los flujos más importantes.
4. **Arma el informe** HTML: cumplimiento, criterios WCAG incumplidos con su nombre y nivel, cada problema con sus elementos, capturas y cómo corregirlo, teclado y orden del foco, y recomendaciones.

```
npm run accesibilidad -- --url https://staging.mi-app.com --nivel AA # sin agentes ni tokens
npm run accesibilidad -- --paginas https://mi-app.com/registro,https://mi-app.com/login --historia HU-001
```

! Una auditoría automática detecta entre un 30 y un 50 % de los problemas de accesibilidad: el agente nunca dice que el sitio "cumple", dice qué no cumple y qué faltó revisar. Todo queda en `output/accesibilidad/<HU o sitio>/` y la corrida anterior pasa a `historial/` (se guardan las últimas 5) y el informe la compara.

Rendimiento web

El **auditor-rendimiento-web** mide qué tan rápido carga cada página para una persona y por qué, con **Lighthouse** en mobile y desktop. No es una prueba de carga con muchos usuarios: eso es el **performance-mapper**.

auditor-rendimiento-web USA TOKENS

PEDIDO · *Revisá la velocidad del sitio del .env.*

1. **Mide** cada página con `npm run rendimiento`: puntajes, Core Web Vitals (LCP, CLS y TBT) con los umbrales de Google, FCP y Speed Index.
2. **Encuentra** las oportunidades de mejora con el ahorro estimado y los recursos que las causan, el elemento LCP, el peso por tipo de recurso y el código de terceros.
3. **Arma el informe** HTML con la secuencia de carga en capturas, el reporte nativo de Lighthouse de cada página y las recomendaciones priorizadas.

```
npm run rendimiento -- --url https://staging.mi-app.com --max-paginas 5 # sin agentes ni tokens
npm run rendimiento -- --paginas https://mi-app.com/,https://mi-app.com/login --repeticiones 3
```

i Son datos de laboratorio (una carga simulada en condiciones fijas): sirven para encontrar y comparar problemas, no para afirmar cómo lo vive cada persona. Para comparar antes y después, usá `--repeticiones 3`. Necesita Node 22.19 o superior.

Análisis de fallos

Cuando `npm test` falla, el **analista-fallos** separa los bugs reales de los tests rotos, inestables, de datos o de ambiente, para no revisar falla por falla.

analista-fallos USA TOKENS

PEDIDO · *Analizá por qué falló npm test.*

1. **Clasifica** cada falla con `scripts/analizar_fallos.py`: posible bug, locator o espera, ambiente, datos de prueba, error en la automatización o paso sin definir, y rescata captura, video y trace.
2. **Reintenta** los fallidos (con tu ok) y suma el reintento: lo que falló y después pasó queda como inestable.
3. **Confirma** la causa con la evidencia, redacta los reportes de bug de los reales y propone cómo arreglar el resto.

```
python scripts/analizar_fallos.py           # la última corrida de npm test
python scripts/analizar_fallos.py --agregar-reintento # después de volver a correr los fallidos
```

- i Volver a correr los fallidos corre la suite contra el ambiente: se hace solo con tu confirmación. Los cambios en la automatización los hacen los mappers.

Logs del servidor

El **analista-logs** lee los logs de la aplicación y los cruza con la corrida de `npm test`: qué errores hubo, cuándo, y qué error del servidor había detrás de cada escenario que falló.

analista-logs USA TOKENS

PEDIDO · ¿Por qué falló CP-004 del lado del servidor? Los logs están en `input/logs/`.

Entiende logs de la app (ISO, `YYYY/MM/DD`), accesos de nginx y Apache, JSON por línea, syslog y `.gz`. Agrupa los errores por firma (ignorando números e ids), les pega su stack trace y los ubica en el tiempo. Los steps de k0lmena guardan el inicio y el fin de cada escenario, así cada falla queda con los errores del servidor mientras corría. Entrega `informe-logs.html` con errores y advertencias por minuto, las firmas y cada escenario fallido con su contexto.

```
python scripts/analizar_logs.py --logs "input/logs/*.log" --corrida web
python scripts/analizar_logs.py --logs input/logs/app.log --desde 2026-10-11T10:00 --hasta 2026-10-11T11:00
```

i Sin `--desde` ni `--hasta` analiza la ventana de la corrida más 5 minutos antes y después. `input/logs/` no se versiona. Bajar logs de un servidor compartido se confirma antes. Salida en `output/logs/<nombre>/`.

Cobertura y trazabilidad

El **analista-cobertura** responde "¿qué nos falta probar?": cruza criterios, casos, automatización, gestión, resultados y bugs de cada historia y marca dónde se corta la cadena.

analista-cobertura USA TOKENS

PEDIDO · ¿Qué nos falta probar de HU-001?

1. **Cruza** con `scripts/analizar_cobertura.py` los criterios (historia, ficha de contexto, tablas de cobertura o Jira), los casos manuales y BDD, la automatización de k0lmena, la trazabilidad con Xray/QMetry/AIO, los últimos resultados y los bugs.
2. **Marca los huecos**: criterios sin casos, casos sin criterio, sin automatizar, bloqueados o sin ejecutar, fallidos, sin subir a la herramienta de gestión y falta información abierta.
3. **Arma el informe** HTML con el embudo de la historia al resultado, la cobertura por criterio, la matriz caso por caso y qué hacer con cada hueco.

```
python scripts/analizar_cobertura.py --historia HU-001
python scripts/analizar_cobertura.py                # todas las historias del repo
```

i Solo lee: no crea casos ni automatiza. El par `@HU` + `@CP` es lo que traza cada caso; un escenario sin `@CP` aparece como hueco.

Regresión visual (pixel perfect)

El **pixel-perfect** detecta **qué cambió visualmente** entre la versión aceptada de tu sitio (la referencia) y la actual: qué se corrió y cuántos píxeles, qué cambió de tamaño, de texto o de color, qué falta y qué apareció.

QUÉ COMPARA	CÓMO
Píxeles	Porcentaje de la pantalla que cambió y las regiones con cambios, con el mapa de diferencias en rojo.
Elementos	La posición y el tamaño de cada título, link, botón, campo e imagen: corridos (con los px), faltantes, nuevos y con otro texto.
Bloques desplazados	Si muchos elementos se movieron igual, es una sola causa (un banner nuevo empujó todo): informa el bloque y la causa probable.
Estilos	Color de fondo, color del texto, tamaño y grosor de la letra.

pixel-perfect USA TOKENS

PEDIDO · Corré una regresión visual de <https://staging.tu-app.com> y decime si algo se corrió.

1. **La primera corrida crea la referencia:** una captura estable de cada página en cada resolución (sin animaciones, con fuentes e imágenes cargadas y las zonas dinámicas ocultas).
2. **Las siguientes comparan** contra ella, píxel a píxel y elemento por elemento.
3. **Separa lo intencional** (lo que pidió la historia) de las regresiones.
4. **Arma el informe:** comparador deslizable referencia/actual, mapa de diferencias, regiones de cerca y el antes y después de cada hallazgo.

```
npm run pixel-perfect -- --url https://staging.tu-app.com --resoluciones 390x844,1366x768 # sin agentes ni tokens
npm run pixel-perfect -- --ocultar ".banner,.fecha" # zonas que cambian solas
npm run pixel-perfect -- --aceptar todas # lo actual pasa a ser la referencia (con tu ok)
```

- i** La referencia también puede ser un diseño de Figma exportado como PNG. Todo queda en `output/pixel-perfect/<HU o sitio>/`: la `referencia/` se conserva y, en cada comparación, la corrida anterior pasa a `historial/` (se guardan las últimas 5) y el informe la compara.

Cross-browser

El **cross-browser** encuentra lo que funciona o se ve distinto según el navegador: **Chromium** (Chrome y Edge), **Firefox** y **WebKit** (el motor de Safari), en desktop y mobile.



Modo suite

Corre la suite web de k0lmena una vez por navegador (cada uno en su carpeta) y arma la matriz escenario × navegador: pasa en todos, falla solo en algunos o falla en todos.



Modo páginas

Sin automatización: recorre las páginas en cada navegador y las compara contra el base: errores de JavaScript y de consola solo en uno, desborde, elementos que faltan o se corren y estilos distintos.

cross-browser USA TOKENS

PEDIDO · Probá HU-003 en Chrome, Firefox y Safari.

El informe trae la matriz con la captura de cada navegador, las páginas lado a lado con el comparador deslizable, y cada diferencia con su error y causa probable. Lo que falla en todos no es un tema de navegador: lo deriva al **analista-fallos**.

```
npm run cross-browser -- --historia HU-003 # modo suite (corre contra el ambiente: con tu ok)
npm run cross-browser -- --modo paginas --url https://staging.tu-app.com # solo GET
cd herramientas/k0lmena && npx playwright install firefox webkit # si faltan los navegadores
```

! WebKit es el motor de Safari y encuentra casi todo, pero no reemplaza un Safari real de Mac o iPhone. Salida en `output/cross-browser/<HU, sitio o suite>/`; la corrida anterior pasa a `historial/` (se guardan las últimas 5) y el informe la compara.

Reparación de la automatización

Cuando un test falla porque **la pantalla cambió** (un botón con otro nombre, un selector que ahora encuentra varios elementos), el **reparador-automatizacion** propone el locator nuevo, lo aplica y verifica que el test vuelva a pasar. Si falla porque la aplicación tiene un bug, no toca el test.

DIAGNÓSTICO	QUÉ HACE
Ya no existe	Busca el elemento más parecido (rol, nombre, texto, label, placeholder, data-testid) y propone el locator más robusto que encuentre un solo elemento. En un paso de verificación (Then) lo marca como posible bug.
Coincide con varios	Prueba con <code>exact: true</code> o lo acota al contenedor de cada elemento (la tarjeta del producto) y te deja elegir.
Existe pero no se ve	Falta un paso previo (abrir un menú) o la app no lo muestra.
Existe y se ve	El locator está bien: propone la espera adecuada.

reparador-automatizacion USA TOKENS

PEDIDO · *Repará los tests de HU-001 que fallaron por locators.*

1. **Analiza** las fallas del último `npm test` (o del analista-fallos) contra el DOM que k0lmena guardó al fallar, sin tocar archivos.
2. **Aplica** las reparaciones seguras (o las que confirmes), con copia de respaldo y `cambios.diff`.
3. **Verifica** corriendo solo los escenarios reparados (con tu ok) y arma el informe con el diagnóstico, el cambio y las capturas.

```
npm run reparar -- --historia HU-001           # diagnóstico y propuestas
npm run reparar -- --aplicar seguras --historia HU-001 # o --aplicar R-01,R-03
npm run reparar -- --verificar --historia HU-001      # vuelve a correr los reparados
```

i Desde esta versión, k0lmena guarda el DOM de la página al fallar (`reports/web/dom/`) y se corrigió el viewport: si `VIEWPORT_WIDTH` y `VIEWPORT_HEIGHT` quedan vacíos, la ventana usa 1366×768 (antes quedaba en 0 y los clics fallaban). Salida en `output/reparaciones/<HU o suite>/`.

Selección de casos para un cambio

El **selector-casos-de-prueba** responde “¿qué pruebo con este cambio?” sin correr toda la regresión: cruza lo que cambió con todos los casos y entrega una lista priorizada, explicada y lista para ejecutar.

ENTRADA	QUÉ USA
El cambio	Un rango de git (archivos, commits con sus keys de Jira y funciones tocadas), los archivos de un PR, historias o issues de Jira y Azure DevOps, o las notas de la release.
El inventario	Casos manuales (Excel), BDD y automatizados en k0lmena, con su último resultado, los inestables y los bugs abiertos.
El resultado	Imprescindibles, recomendados y regresión mínima (smoke y prioridad Crítica), con el porqué, el tiempo estimado, el ahorro y los cambios que ningún caso cubre.

selector-casos-de-prueba USA TOKENS

PEDIDO · ¿Qué casos corro para el PR #42? Toca el pago con tarjeta (PROJ-55).

Entrega `informe-seleccion.html` con los comandos listos para copiar (bash y PowerShell) para lo automatizado y `seleccion.csv` con lo manual. No corre las pruebas: lo automatizado va con `npm test`, sin tokens.

```
python scripts/seleccionar_casos.py --nombre release-2.4 --git main..feature/pagos --repo ../mi-app
python scripts/seleccionar_casos.py --nombre pr-42 --historias HU-003 --presupuesto-min 60
TAGS="(@HU-003 and (@CP-001 or @CP-003))" npm test -- web # el comando que arma el informe
```

i Opcional: `input/mapa-componentes.json` relaciona carpetas del código con historias para afinar la selección. Salida en `output/seleccion-casos/<nombre>/`.

Revisión de casos y tiempo estimado

El **revisor-casos** revisa casos que ya existen (manuales en Excel o escenarios BDD) antes de ejecutarlos, subirlos o automatizarlos, y calcula cuánto lleva ejecutarlos a mano.

revisor-casos USA TOKENS

PEDIDO · Revisá los casos de HU-001 y decime cuánto lleva ejecutarlos.

Un script revisa cada caso (título, precondiciones, pasos accionables, resultado esperado verificable, datos, prioridad, trazabilidad) y le da un puntaje de 0 a 100; detecta duplicados y casi duplicados y las historias sin casos negativos o de borde. El agente suma lo que pide criterio y propone la versión mejorada de los peores. Entrega `informe-revision.html`; si lo pedís, la planilla corregida va en un archivo nuevo, nunca pisa la original.

```
python scripts/revisar_casos.py --historia HU-001
python scripts/revisar_casos.py --origen output/casos-de-prueba/manuales/casos-HU-001.xlsx,output/casos-de-prueba/bdd/HU-001-registro.feature
python scripts/estimacion.py output/casos-de-prueba/manuales/casos-HU-001.xlsx # solo el tiempo
```

Cómo se calcula el tiempo estimado

Cada caso (manual o BDD) lleva el tiempo que tarda una persona que conoce la aplicación en ejecutarlo de punta a punta. Lo calcula `scripts/estimacion.py` y aparece en la planilla (columna **Tiempo estimado (min)** con el total), en el `.md`, en la herramienta de gestión, en la selección de casos y en esta revisión.

```
tiempo del caso = (preparación + Σ pasos + registro) × factor # hacia arriba a 0,5 min, mínimo 1
tiempo del paso = acción + datos + verificación
```

PARTE	CÓMO SE CALCULA
Acción	Según el verbo: navegar 0,5 · clic 0,25 · completar 0,3 por campo · iniciar sesión 1 · subir o bajar un archivo 1 · esperar un correo o un proceso 2 · consultar una base, una API o logs 2 · leer 0,5 · otra 0,5 min.
Datos	0,15 min por cada dato extra del paso.
Verificación	0,5 min si hay resultado esperado (o es un <code>Then</code>) + 0,25 por cada verificación extra (hasta 4).
Preparación	1 min + 0,5 por precondición (hasta 3) + 2 si hay que crear datos o usuarios.
Registro	0,5 min + 0,1 por paso (marcar el resultado y guardar la evidencia).
Factor	1,1 para casos negativos o de borde; <code>ESTIMACION_FACTOR</code> del <code>.env</code> multiplica todo (por ejemplo 1,3 para alguien que recién conoce la aplicación).

i Si la planilla ya trae un valor en **Tiempo estimado (min)**, ese manda: se respeta lo que cargó la persona. Fórmula completa, con ejemplos, en `plantillas/estimacion-tiempos.md`. Salida de la revisión en `output/revision-casos/<HU o nombre>/`.

Charters y sesiones exploratorias

El **generador-charters** planifica pruebas exploratorias por sesiones (session-based testing): cada charter dice qué explorar, con qué foco y heurísticas, con qué datos y en cuánto tiempo. Después arma el informe con lo que encontró cada sesión.

generador-charters USA TOKENS

PEDIDO · Armá charters para explorar el checkout de la release 2.4.

Entrega las hojas de sesión (`charters.html` y `.md`) y una plantilla de notas por sesión en `sesiones/CH-XX.md` . Quien explora anota con prefijos (`BUG:` `PREGUNTA:` `IDEA:` `RIESGO:` `CUBIERTO:` `SIN CUBRIR:` `NOTA:` y `Tiempo:`) y el informe sale de esas notas: bugs, preguntas, riesgos, reparto del tiempo y áreas cubiertas.

```
python scripts/generar_charters.py --nombre HU-003          # hojas de sesión
python scripts/generar_charters.py --nombre HU-003 --informe # informe-sesiones.html desde sesiones/
```

i Si le pedís que explore él, usa Playwright MCP y completa las notas de la sesión. Salida en `output/charters/<HU o nombre>/` .

Correos

El **verificador-correos** comprueba que los mails que manda tu aplicación (registro, confirmación, recuperar contraseña, pedidos, notificaciones) **lleguen bien**: a la persona correcta, con el contenido correcto y viéndose bien en los clientes de correo. Usa **Mailpit**, un buzón de prueba que atrapa los mails del ambiente de pruebas sin entregarlos a nadie.

QUÉ REVISAS	CÓMO
Lo esperado	Por cada caso: destinatario, remitente, asunto, textos que tiene (y que no), links y adjuntos. Marca los que no llegaron.
Contenido	Variables sin reemplazar (<code>{{nombre}}</code>), <code>undefined</code> , <code>null</code>), asunto, versión de texto plano y tamaño (Gmail recorta desde 102 KB).
Links e imágenes	Links rotos, a <code>localhost</code> o sin https; imágenes sin texto alternativo o que apuntan a rutas locales.
Cómo se ve	Compatibilidad del HTML con Outlook, Gmail, Apple Mail y otros (revisión de Mailpit) y captura en desktop (700 px) y mobile (375 px).

verificador-correos USA TOKENS

PEDIDO · Verificá que llegue el mail de bienvenida a `ana@test.com`.

1. **Levantá Mailpit** y configurá el SMTP de la app de pruebas a `localhost:1025`.
2. **Disparás el envío** (vos, `npm test` o el agente con tu ok: crea datos).
3. **Revisa el buzón** contra lo esperado de cada caso y arma el informe con la vista de cada correo.
4. **Si lo pedís**, suma steps de correo a los `.feature` de k0lmena, que corren sin tokens.

```
docker compose -f herramientas/mailpit/docker-compose.yml up -d      # web y API en :8025, SMTP en :1025
npm run correos -- --para ana@test.com --espera input/correos/espera-HU-001.json
npm run correos -- --para ana@test.com --esperar 1 --timeout-s 90    # espera a que llegue
```

```
Then llega un correo a "ana@test.com" con el asunto "Confirmá tu cuenta"
And el correo contiene "Hola Ana"
And el correo no contiene "undefined"
When guardo el link del correo que contiene "/confirmar" como "linkConfirmacion"
```

! Solo buzones de prueba: nunca casillas reales ni el servidor de correo de producción. Vaciar el buzón (`--limpiar-buzon`) se confirma. Salida en `output/correos/<HU o buzón>/`; la corrida anterior pasa a `historial/` (se guardan las últimas 5) y el informe la compara.

Traducciones e idiomas

El **analista-traductor** recorre cada página en cada idioma (el primero es la referencia) y encuentra lo que falta traducir o se ve mal en otro idioma. Propone la traducción de lo que falta.

QUÉ BUSCA	EJEMPLO
Textos sin traducir	Un párrafo en español en la versión en inglés, o igual al de la referencia; también en placeholders, textos alternativos y <code>aria-label</code> .
Claves a la vista	<code>checkout.add_to_cart</code> , <code>MISSING_TRANSLATION</code> .
Caracteres rotos	<code>EnvÃo</code> en lugar de <code>Envío</code> .
Textos cortados	Un botón que no deja lugar para la traducción, más larga.
Formatos	<code>1.234,56</code> en la versión en inglés; "octubre" en una fecha en inglés; el atributo <code>lang</code> equivocado.

analista-traductor USA TOKENS

PEDIDO · *Revisá las traducciones del sitio en inglés y portugués.*

Cambia de idioma con los `hreflang` del sitio, con `--url-idioma` (por ejemplo `https://tu-app.com/{idioma_corto}/`) o pidiendo el idioma al navegador. El informe trae el porcentaje traducido por idioma, cada página en todos los idiomas lado a lado y cada problema con su recorte.

```
npm run traductor -- --url https://staging.tu-app.com --idiomas es-AR,en-US,pt-BR # sin agentes ni tokens
npm run traductor -- --url https://tu-app.com --url-idioma "https://tu-app.com/{idioma_corto}/"
```

- i La detección de idioma es una heurística: el agente confirma cada caso y descarta nombres propios y marcas. Salida en `output/traductor/<HU o sitio>/`; la corrida anterior pasa a `historial/` (se guardan las últimas 5) y el informe la compara.

Usabilidad

El **evaluador-ux** revisa qué confunde, frena o hace equivocar a la persona, con las **10 heurísticas de Nielsen**: un recorrido automático por las páginas y uno manual por los flujos importantes.

evaluador-ux USA TOKENS

PEDIDO · *Evaluá la usabilidad del checkout.*

El recorrido automático revisa, en desktop y mobile, lo que se puede medir: la página 404 (si ayuda a volver), menús muy largos o sin la sección actual marcada, nombres distintos para la misma acción, links que abren otra pestaña sin avisar, botones solo con ícono, campos con el placeholder como única etiqueta, mensajes técnicos a la vista, la acción principal abajo del pliegue en mobile y más. El agente recorre los flujos con Playwright MCP y suma lo que solo se ve usándolos. Entrega `informe-ux.html` con el puntaje por heurística, cada hallazgo con su captura y recomendaciones.

```
npm run ux -- --url https://staging.tu-app.com # sin agentes ni tokens
npm run ux -- --url https://tu-app.com/checkout --max-paginas 4 --resoluciones 1440x900,390x844
```

i No avanza en un flujo que crea datos (un pedido, una cuenta) sin confirmación. Salida en `output/ux/<HU o sitio>/`; la corrida anterior pasa a `historial/` y el informe la compara.

Formularios

El **probador-formularios** prueba las validaciones de cada campo de un formulario **sin enviarlo**: obligatorios, largos, formatos, límites y mensajes de error.

probador-formularios USA TOKENS

PEDIDO · *Probá las validaciones del formulario de registro.*

Encuentra los formularios, deduce qué es cada campo (tipo, etiqueta, `required`, `maxlength`, `min / max`, `pattern`) y le prueba lo que corresponde: vacío, solo espacios, más largo que el máximo, emails y teléfonos inválidos, números fuera de rango, nombres con acentos y ñ. Marca los **valores inválidos aceptados** y los **válidos rechazados**, campos sin etiqueta y mensajes que no dicen qué corregir. Si la historia define reglas, el agente las compara. Entrega `informe-formularios.html` con la matriz campo × prueba y capturas con los errores a la vista.

```
npm run formularios -- --url https://staging.tu-app.com/registro # sin agentes ni tokens
npm run formularios -- --url https://staging.tu-app.com/registro --enviar-vacio # solo con confirmación
```

! Nunca envía un formulario con datos: bloquea los envíos que no son GET. Lo único que puede enviar, con confirmación, es el formulario vacío (`--enviar-vacio`). Salida en `output/formularios/<HU o sitio>/` .

Historial de corridas

Cada herramienta guarda sus **últimas 5 corridas** con su evidencia, así podés comparar antes y después de un arreglo sin perder nada.

DÓNDE	QUÉ GUARDA
<code>output/<herramienta>/<HU o sitio>/historial/<fecha-hora>/</code>	El JSON, el análisis, el informe y la evidencia (capturas, videos, recortes) de cada corrida anterior: su informe se sigue abriendo igual.
<code>herramientas/k0lmena/reports/<suite>/historial/</code>	El reporte de Cucumber y la evidencia de cada <code>npm test</code> anterior (capturas, videos, traces y el DOM al fallar).



Nuevos
Lo que aparece en esta corrida y no estaba en la anterior.



Resueltos
Lo que estaba y ya no aparece.



Evolución
Los indicadores principales de cada corrida en el tiempo, con el link al informe de cada una.

i Cada informe (exploratorias, accesibilidad, rendimiento, regresión visual, cross-browser, fallos, cobertura, reparaciones, selección, correos, traducciones, revisión de casos, charters, usabilidad, formularios y logs) trae la sección **Evolución de las corridas**. Para guardar más corridas: `HISTORIAL_CORRIDAS=10` en el entorno de las herramientas de k0lmena.

Seguridad web

El **analista-seguridad** revisa la seguridad de tu aplicación con **OWASP ZAP** (libre y gratuito) en modo **pasivo**: recorre el sitio y analiza las respuestas sin enviar ataques ni modificar datos.

! Solo se escanean sitios propios o con autorización por escrito, declarados en `SEGURIDAD_URLS_AUTORIZADAS` del `.env`. Un escaneo pasivo no reemplaza una prueba de penetración.

analista-seguridad

PEDIDO · Revisá la seguridad de `https://staging.mi-app.com` para `HU-001`.

1. **Confirma** la URL (tiene que estar autorizada) y la duración del recorrido.
2. **Escanea** con ZAP baseline en Docker (abrí Docker Desktop antes).
3. **Analiza** los hallazgos: los prioriza, descarta falsos positivos con su motivo y los explica en español.
4. **Arma el informe** HTML: hallazgos por riesgo, dónde aparece cada uno, evidencia, cómo se corrige y recomendaciones.

```
python herramientas/zap/escanear.py --url https://staging.mi-app.com --historia HU-001
```

Salida en `output/seguridad/<HU o host>/<fecha>/`. Detalle en `herramientas/zap/README.md`.

Conectores MCP

Conectan a Claude Code con sistemas externos. Claude Code arranca directo: vienen activos Playwright y Atlassian, y cada QA activa el resto con `npm run conector`. Sin secretos: las credenciales se leen del `.env`. Vienen activos Playwright y Atlassian (Jira y Confluence); el `.env.example` trae un bloque comentado por herramienta para completar solo los de tu equipo.

CONECTOR	PARA QUÉ	AUTENTICACIÓN
<code>playwright</code>	Navegador para el ejecutor E2E y el web-mapper (headed y headless).	Activo
<code>atlassian</code>	Jira y Confluence: historias, criterios y documentación funcional directamente.	Activo · OAuth o API token
<code>figma</code>	Diseños desde un link a un frame o archivo: textos, estados y componentes. Requiere asiento Full o Dev.	Cuenta (OAuth)
<code>appium-mcp</code>	Lo usa el mobile-mapper para recorrer la app.	<code>ANDROID_HOME</code>
<code>qmetry</code> · <code>qtm4j</code>	Consultar QMetry desde el chat.	API key
<code>aio-tests</code>	Consultar AIO Tests desde el chat.	Token
<code>azure-devops</code>	Azure DevOps: historias, Wiki y casos en Test Plans.	PAT
<code>k0lmena-tmt</code>	Cargar y consultar casos en k0lmenaTMT.	Token personal

- 1

Completá el .env
Si usa token, cargalo en el `.env` de la raíz. Atlassian y Figma se autorizan con tu cuenta desde `/mcp`.
- 2

Activalo
`npm run conector -- activar azure-devops` (queda solo para vos; `npm run conector` lista los disponibles).
- 3

Verificá
Reiniciá Claude Code y revisá la conexión con `/mcp`.

PEDIDO · *Generá los casos de PROJ-12 usando la página de Confluence 'Reglas de facturación' y este diseño de Figma.*

Para Xray no hace falta MCP: se usa `scripts/gestion/`. Guía completa en `CONECTORES.md`.

Convenciones

ELEMENTO	FORMATO
Historias	HU-001
Criterios de aceptación	CA1 · CA2
Casos manuales	CP-001
Casos de API	CP-API-001
Bugs	BUG-001
Falta información	FI-01 · @falta-info @FI-01

ESCALA	VALORES
Severidad y prioridad	Crítica · Alta · Media · Baja
Estado de un caso	N/A · Pendiente · En ejecución · Aprobado · Fallido · Bloqueado

Tags en k0lmena

DÓNDE	TAGS
Feature	La historia y el tipo: @HU-001 @web · @api · @mobile
Scenario	El ID del caso (@CP-001 , @CP-API-001) y @smoke si es crítico
Excluir	@bloqueado : un paso no se pudo verificar y el escenario no se ejecuta

Nombres de archivos

ARTEFACTO	NOMBRE
Análisis	analisis-HU-001.md
Casos manuales	casos-HU-001.xlsx + .md + -cobertura.md
Casos BDD	HU-001-registro.feature + HU-001-cobertura.md
Bug	BUG-001.md
Automatización	<tipo>/features/HU-001-<slug>.feature · steps/HU-001.steps.ts · locators/HU-001.locators.ts
Performance	performance/k6/http/HU-001-<slug>.ts · performance/artillery/HU-001-<slug>.yaml · performance/jmeter/HU-001-<slug>.jmx
Mapeo	output/mapeos/mapeo-HU-001-<tipo>.md
Exploratorias	output/exploratorias/<HU o sitio>/informe-exploratorio.html + exploracion.json · capturas/ · videos/ · historial/ (últimas 5)
Rendimiento web	output/rendimiento/<HU o sitio>/informe-rendimiento.html + rendimiento.json · capturas/ · lighthouse/
Análisis de fallos	output/fallos/<HU o suite>/informe-fallos.html + fallos.json · corridas/ · evidencia/
Cobertura	output/cobertura/<HU o todas>/informe-cobertura.html + cobertura.json

Regresión visual	output/pixel-perfect/<HU o sitio>/informe-pixel-perfect.html + pixel-perfect.json · referencia/ (se conserva) · actual/ · diferencias/ · recortes/
Cross-browser	output/cross-browser/<HU, sitio o suite>/informe-cross-browser.html + cross-browser.json · capturas/ · suite/
Reparaciones	output/repaciones/<HU o suite>/informe-reparacion.html + reparaciones.json · cambios.diff · respaldo/
Selección de casos	output/seleccion-casos/<nombre>/informe-seleccion.html + seleccion.json · seleccion.csv
Accesibilidad	output/accesibilidad/<HU o sitio>/informe-accesibilidad.html + auditoria.json · capturas/ · videos/ · historial/ (últimas 5)
Correos	output/correos/<HU o buzón>/informe-correos.html + correos.json · capturas/ · eml/
Traducciones	output/traductor/<HU o sitio>/informe-traductor.html + traductor.json · capturas/ · recortes/
Revisión de casos	output/revision-casos/<HU o nombre>/informe-revision.html + revision.json
Charters	output/charters/<HU o nombre>/charters.html + charters.md · sesiones/CH-XX.md · informe-sesiones.html
Usabilidad	output/ux/<HU o sitio>/informe-ux.html + ux.json · capturas/ · recortes/
Formularios	output/formularios/<HU o sitio>/informe-formularios.html + formularios.json · capturas/
Logs	output/logs/<nombre>/informe-logs.html + logs.json
Historial	<carpeta de cada herramienta>/historial/<fecha-hora>/ (últimas 5, con su evidencia) · herramientas/k0lmena/reports/<suite>/historial/ para npm test

Estructura del repositorio

```

k0lmenIA/
├─ CLAUDE.md           Contexto y estándares (Claude Code lo lee siempre)
├─ ARQUITECTURA.md    Cómo crece el repo
├─ CONECTORES.md       Cómo activar los conectores MCP
├─ CHANGELOG.md        Cambios de cada versión
├─ LICENSE             Licencia MIT
├─ .env.example        Plantilla de variables (copiar a .env)
├─ .mcp.json           Conectores MCP (sin secretos; activos según settings)
├─ package.json        Atajos para correr k0lmena desde la raíz (npm test...)
├─ docs/               Este sitio, el PDF y los diagramas
├─ tests/              Tests de los scripts (pytest y node --test)
├─ .github/workflows/  CI: los tests en Ubuntu y Windows
├─ .claude/
│   └─ agents/         Los 35 agentes
│   └─ skills/         Contexto, diseño, guía de performance, ejecución, k0lmena
│   └─ settings.json   Conectores habilitados y permisos
├─ input/              historias/ documentacion/ api/ bugs/ correos/ logs/
├─ output/             Lo que generan los agentes
├─ plantillas/         Bug, planilla de casos, cobertura, tiempo estimado
├─ scripts/            Casos, tiempo estimado, revisión, charters, logs, informes y doctor
│   └─ gestion/        Xray, QMetry (QTM4J), AIO Tests y casos para Azure DevOps
│   └─ mcp/            Lanzadores de los conectores MCP (leen el .env)
└─ herramientas/
    ├─ newman/          Colecciones de Postman
    ├─ zap/             OWASP ZAP: seguridad web (pasivo)
    ├─ mailpit/         Mailpit: buzón de prueba para los correos
    └─ k0lmena/
        ├─ web/         features/ steps/ locators/ + framework
        ├─ api/         features/ steps/ (steps genéricos)
        ├─ mobile/      features/ steps/ locators/ support/ apps/
        ├─ performance/ k6/http/, artillery/ y jmeter/
        ├─ reports/     Reportes HTML
        ├─ run-tests.js  Runner de npm test
        └─ run-perf.js   Runner de npm run perf

```

Problemas frecuentes

¿Qué pasa si a la historia le falta información?

No sé qué me falta instalar o configurar

Un agente devuelve "Necesito que confirmes"

Un agente nuevo no aparece

La suite web falla sin URL o los clics dicen "outside of the viewport"

```
npm test
```

corre 0 escenarios

Un escenario nunca se ejecuta

"No encuentro k6"

"No encuentro JMeter" o "JMeter necesita Java"

```
npm run perf
```

no corre un perfil con carga desde el agente

Una carpeta de Xray/QMetry/AIO aparece como

```
C:/Program Files/Git/...
```

Un conector MCP no conecta

¿Los tests verifican la base de datos siempre?

Falta una librería de Python

Extender el proyecto

Agregar un agente

- 1

Creá el archivo
`.claude/agents/mi-agente.md` con frontmatter `name` y `description`. La descripción es lo que usa Claude Code para decidir cuándo invocarlo.
- 2

Escribí el cuerpo
En español: rol, entradas, proceso, salida y reglas.
- 3

Definí el formato
Si genera un formato propio, sumá su plantilla en `plantillas/` o su script en `scripts/`, y su carpeta en `output/`.
- 4

Normalizá las tablas
Si genera tablas en un `.md`, pasalas por `scripts/formatear_tablas.py`.
- 5

Si necesita una respuesta de la persona
Copiá la sección "Cuando te falta un dato o una confirmación" de cualquier agente: los subagentes no pueden preguntar a mitad del trabajo, así que devuelven "Necesito que confirmes".

Skills, conectores y herramientas

Skill
Una carpeta en `.claude/skills/` con su `SKILL.md`; se carga solo cuando hace falta.

Conector MCP
Una entrada en `.mcp.json`, sin secretos; los tokens se leen del `.env` con un script de `scripts/mcp/`.

Herramienta
Una subcarpeta en `herramientas/` con su README.

Tests del repo
`python -m pytest tests` y `npm run test:repo`; corren en CI (Ubuntu y Windows) en cada cambio. Sumá un test cuando cambies un script.

Regenerar esta documentación

```
node docs/generar.js # diagramas (PNG) y PDF, con el Playwright de k0lmena
```

El sitio es `docs/index.html` (publicado con GitHub Pages en underc0delabs.github.io/k0lmenIA); el PDF sale del mismo HTML con estilos de impresión, y los diagramas de `docs/diagramas/`.

Licencia

Licencia MIT — © 2026 QARMY. k0lmena es un proyecto open source de Danilo Vezzoni. Hecho por la comunidad de Underc0de.