

SESIÓN FINAL · GRADUACIÓN

KATA Bridge

Del Page Object hecho a mano al **framework profesional**.
Hoy no aprendes piezas nuevas — aprendes el mapa que las
ensambla.

KATA: 4 capas

ATCs

kata-manifest

Agentic QA

graduación 🏆

MIRA HACIA ATRÁS

El viaje que ya hiciste



Coding Base

valores → tu primer test



Coding Classes

clases → Page Object



Dojo Lab I

Playwright real + dojo



Playwright Pro

hooks, POM, fixtures, API



Dev Craft

terminal, git, tu primer PR



KATA Bridge

estás aquí



Hace 5 sesiones, "Ana" era tu primer valor. Hoy tienes: suite multi-browser con Page Objects y fixtures propios, publicada en GitHub por PR. **Lo que falta no es conocimiento — es el mapa del territorio profesional.**

Tu mini-framework, a escala industrial 🏭

Tu proyecto (3 páginas, 1 QA)

pages/ + fixtures.ts + tests/. Funciona hermoso. Pero ahora imagina:

- 📈 **40 páginas** y **30 endpoints** de API
- 👥 **5 QAs** escribiendo a la vez
- 🌐 **3 ambientes** (local, staging, prod)
- 🎲 datos fake distintos en cada corrida
- 🔍 trazabilidad: cada test ↔ su test case del TMS

😞 Dolor 1

Cada POM repite los mismos helpers (waits, navegación, formularios)
→ **capa de bases heredables**.

😞 Dolor 2

Credenciales y URLs hardcodeadas por ambiente → **capa de configuración**.

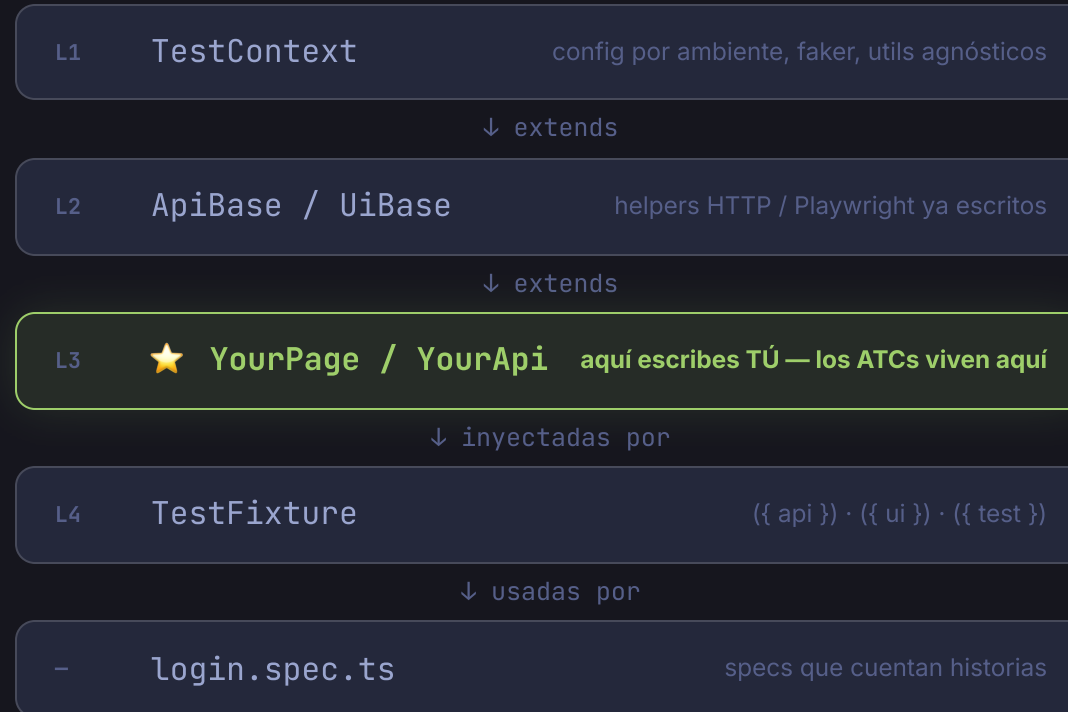
😞 Dolor 3

Con 5 QAs, nadie sabe qué ya existe → duplicados → **un registro central**.

💡 KATA

Cada dolor de escala tiene una pieza con nombre. Eso ES el framework.

KATA: Component Action Test Architecture



KATA como en artes marciales: una *forma* ensayada hasta que sale sola. Dos mecanismos que ya dominas la sostienen: **extends** (Coding Classes) y **fixtures** (Playwright Pro). Nada más.

L1 + L2: todo lo que ya no tendrás que escribir

L1 · TestContext — la mochila común

- Config por ambiente: URLs y credenciales salen de `.env` + config — **mata tus credenciales hardcodedas**
- Faker: datos fake frescos en cada corrida (¿recuerdas tu `Date.now()` del email único? esto, industrializado)
- Utilidades agnósticas compartidas por API y UI

L2 · UiBase / ApiBase — los helpers heredables

- **UiBase**: navegación, waits inteligentes, helpers de formularios — todo lo que tus POMs repetían
- **ApiBase**: cliente HTTP con auth, headers y manejo de errores resuelto

```
// Tu LoginPage de Playwright Pro:  
export class LoginPage {           // extendía... nada  
    // goto, waits, helpers: a mano 😊  
}
```

```
// En KATA:  
export class LoginPage extends UiBase {  
    // hereda navegación, waits, forms,  
    // config y faker — GRATIS 🎁  
    // tú solo escribes lo específico de login  
}
```



La cadena extends que practicaste con User → Admin, pagando su dividendo más grande: **escribes solo lo que es único de tu página.**

L3 + el concepto estrella: ATC

tests/components/pages/login.page.ts — capa L3

```
export class LoginPage extends UiBase {  
  
  /** @atc TC-101 — login con credenciales válidas */  
  async loginAs(user: { email: string; password: string }) {  
    await this.page.goto(this.config.webUrl + "/login");  
    await this.page.getByLabel("Email").fill(user.email);  
    await this.page.getByLabel("Password").fill(user.password);  
    await this.page.getByRole("button", { name: /sign in/i }).click();  
  }  
}
```

ATC

Acceptance Test Case: un método que ejecuta un **mini-flujo completo y atómico**. Tu `login()` de siempre — con identidad y reglas.

TC-101

Cada ATC lleva el ID de su test case del TMS — **trazabilidad:** del código al test documentado y de vuelta.

Objeto en vez de 2 params

Nota la firma: `loginAs(user)` recibe un objeto — regla de la casa para 3+ datos.

Las reglas del **ATC** — pocas y sagradas

REGLA	QUÉ SIGNIFICA	POR QUÉ
Atómico	Un ATC ejecuta UN mini-flujo completo y NUNCA llama a otro ATC	Si A llama a B y B cambia, A se rompe en cadena — el dominó que mata frameworks
Cadenas → Steps	¿Necesitas login + crear proyecto + invitar? Eso es un Steps module que orquesta ATCs	Composición explícita en un solo lugar, no acoplamiento oculto
Locators inline	Los locators viven DENTRO del ATC; se extraen solo si 2+ ATCs los usan	Leer un ATC = verlo completo, sin saltar entre archivos
Máx 2 params sueltos	3 o más datos → un objeto: <code>loginAs({ email, password, role })</code>	<code>fn(a, b, c, d)</code> = ¿cuál era el tercero? El objeto se autodocumenta

★ QUIZ 1 · La regla sagrada

Tu ATC `createProject()` necesita que el usuario esté logueado. ¿Puede llamar internamente a `loginAs()`?

A Sí — es reutilización, justo lo que aprendimos

B No — un ATC nunca llama a otro ATC; la cadena login→crear vive en un Steps module (o el login va en el fixture/hook)

C Sí, pero solo si están en la misma clase

💡 La trampa de la opción A: reutilización ES buena, pero el acoplamiento oculto no. Si `loginAs` cambia y 14 ATCs lo llamaban por dentro, tienes 14 roturas invisibles. KATA exige que la composición sea EXPLÍCITA: un Steps module que orquesta, o la precondition en el fixture. Mismo beneficio, cero dominó.

L4 · TestFixture: tu fixtures.ts con esteroides

```
// Tu spec en KATA:
import { test, expect } from "@fixtures/test.fixture";

test("crear proyecto", async ({ ui }) => {
  await ui.loginPage.loginAs(ui.ctx.users.admin);
  await ui.projectsPage.createProject({ name: "Demo" });
  await expect(ui.page.getByText("Demo")).toBeVisible();
});
```

`ui` llega con TODAS las páginas instanciadas + contexto + datos. Cero

`new`, cero setup — como tu fixture `loginPage`, multiplicado.

La regla de selección

Test solo de API	(<code>{ api }</code>) — sin browser, vuela
Test solo de UI	(<code>{ ui }</code>) — páginas listas
Híbrido (API prepara, UI verifica)	(<code>{ test }</code>) — ambos mundos

Por qué importa

Pedir `{ api }` en un test de API **no levanta navegador** — tu suite de 200 tests de API corre en segundos. El fixture correcto = la velocidad correcta.

★ QUIZ 2 • Fixtures

Escribes 15 tests que solo validan respuestas de la API. ¿Qué fixture pides?

A ({ test }) — el completo, por si acaso

B ({ ui }) — todo test necesita navegador

C ({ api }) — sin navegador: 15 tests en segundos

💡 El "por si acaso" de la opción A cuesta caro: levantar browser para tests que jamás lo tocan multiplica la duración de la suite. La regla: pide el fixture MÍNIMO que tu test necesita. La pirámide de Playwright Pro, aplicada a nivel de fixture.

kata-manifest.json: el registro civil

```
// kata-manifest.json (extracto)
{
  "components": {
    "LoginPage": {
      "file": "tests/components/pages/login.page.ts",
      "atcs": ["TC-101", "TC-102"]
    },
    "AuthApi": {
      "file": "tests/components/api/auth.api.ts",
      "atcs": ["TC-201"]
    }
  }
}
```

Resuelve el Dolor 3

"¿Ya existe un ATC de login?" — el manifest responde en 2 segundos.
Se consulta SIEMPRE antes de crear un Component o ATC nuevo.
Anti-duplicación.

Se genera, no se edita

`bun run kata:manifest` lo regenera leyendo el código. Nunca a mano.

Guardián automático

Manifest desactualizado = **el commit se bloquea** (hook de pre-commit). El registro nunca miente porque no puede quedarse atrás.

El repo profesional, a vuelo de pájaro

bunkai-qa-engineering/

- └─ **tests/components/** ← L2 + L3: bases, Pages, Apis, Steps
- └─ **tests/e2e/** · **tests/integration/** ← los specs
- └─ **api/schemas/** ← tipos TS generados del OpenAPI real
- └─ **kata-manifest.json** ← el registro civil
- └─ **.context/** ← mapas de negocio + plan maestro de testing
- └─ **.claude/skills/** ← los flujos de trabajo de la IA
- └─ **package.json** ← scripts con bun (= npm, otro motor)

api/schemas/

¿Recuerdas /api/docs del dojo? Aquí el contrato OpenAPI se convierte en **tipos TypeScript**: tu test de API sabe la forma exacta de cada respuesta. Typo en un campo = error antes de ejecutar.

.context/

Mapas del negocio bajo prueba: features, datos, APIs, plan maestro. La "memoria" del proyecto — tuya y de la IA.

bun

`bun install` = npm install · `bun run test` = npm run test. Mismo concepto, motor más rápido.

Plot twist: trabajarás con una IA de copiloto 🤖

🔧 Es un repo "agentic"

Trae flujos guiados por IA: `/sprint-testing` (QA manual de tickets), `/test-automation` (escribir tests KATA), `/regression-testing` (suites en CI). La IA conoce las reglas KATA y consulta el manifest.

📐 Plan → Code → Review

Regla de la casa: **plan de implementación ANTES de escribir código**. La IA propone el plan, tú lo apruebas, luego se escribe. Nada de código a ciegas.

🧠 ¿Entonces para qué aprendí todo esto?

Porque tu rol NO es tipear código — es **tener criterio**:

- ✅ **Leer** lo que la IA propone y detectar lo que está mal
- ✅ **Juzgar**: ¿este ATC es atómico? ¿el fixture es el mínimo? ¿el locator sobrevivirá?
- ✅ **Decidir** qué se prueba, qué se automatiza y qué NO

Sin las 6 sesiones, la IA te dicta. Con ellas, **tú diriges**.

★ QUIZ 3 · Criterio agentic

La IA te propone un ATC nuevo: `registerUser(name, email, password, role)`.
¿Qué marcas en el review?

A Nada — compila y los tests pasan, se aprueba

B 4 parámetros sueltos viola la regla de la casa: 3+ datos van en un objeto → `registerUser({ name, email, password, role })`

C Que use menos parámetros, quitando el role

💡 "Compila y pasa" no es el estándar — **cumple las reglas de la casa** lo es. La firma con objeto se autodocumenta y no se rompe si mañana se agrega un campo. Esto ES el trabajo agentic: la IA produce, tu criterio garantiza. (La opción C "arregla" rompiendo funcionalidad — cuidado con los arreglos que amputan.)

MISIÓN B1

Pisa el territorio

☐ Completada

 Clonar el repo profesional, instalar dependencias y verificar que el mapa de esta sesión coincide con el territorio.

1. Clona (la URL te la da tu lead):

```
$ git clone <URL-del-repo> && cd bunkai-qa-engineering
```

2. El ritual del clon (¡quiz 1 de Dev Craft!), versión bun:

```
$ bun install
```

3. Abre y explora:

```
$ code .
```

Encuentra y abre estos 4 lugares:

`tests/components/` ¿ves bases y pages?

`kata-manifest.json` ¿cuántos components hay?

`api/schemas/` ¿reconoces tipos TS?

`package.json` ¿qué scripts ofrece?

 **Pista:** bun no está instalado

 **Criterio de éxito:** repo clonado, `bun install` sin errores, y encontraste los 4 lugares del mapa.

MISIÓN B2

Caza del ATC

☐ Completada

 Usar el manifest como lo usarás siempre: encontrar un ATC existente y leer su código fuente.

1. Abre `kata-manifest.json`
2. Elige un Component que te llame la atención
3. Salta a su archivo (el manifest te da la ruta)
4. Lee UN ATC completo, línea por línea
5. Verifica las reglas: ¿es atómico? ¿locators inline? ¿firma con objeto si 3+ datos?

 **Pista:** hay algo que no reconozco



Meta honesta: deberías reconocer el **80%+** de cada ATC (clase, this, async/await, locators, params). El 20% restante es lo que el trabajo diario te dará.



Criterio de éxito: puedes explicarle a un compañero qué hace ese ATC, línea por línea, y qué regla KATA cumple cada decisión.

El spec, descifrado por completo

☐ Completada

 Tomar un spec real del repo y mapear CADA elemento a la sesión de la serie donde lo aprendiste.

1. Abre cualquier spec de `tests/e2e/`
2. Por cada elemento, anota de qué deck salió:

<code>import { test } from "@fixtures/..."</code>	→ ¿deck?
---	----------

<code>async ({ ui }) =></code>	→ ¿deck?
-----------------------------------	----------

<code>ui.loginPage.loginAs({...})</code>	→ ¿deck?
--	----------

<code>await expect(...).toBeVisible()</code>	→ ¿deck?
--	----------



Si completas la tabla sin ayuda, estás graduada/o: no queda UNA sola línea de un spec profesional que no puedas explicar. El framework dejó de ser una caja negra.



Respuestas (sin trampa — primero intenta)



Criterio de éxito: tabla completa — cada línea del spec trazada a su origen. Eso es dominio, no memorización.

★ QUIZ 4 • El integrador final de la serie

Te asignan automatizar "el usuario puede archivar un proyecto". ¿Tu PRIMER movimiento?

A Abrir VS Code y escribir el test — para eso entrenamos

B Grabar el flujo con codegen y limpiar

C Consultar `kata-manifest.json`: ¿ya existe un `ProjectsPage`? ¿un ATC parecido? — construir sobre lo que hay

💡 En un framework de equipo, el primer movimiento NUNCA es escribir — es **consultar el registro**. Quizás `ProjectsPage` ya existe y solo agregas un ATC; quizás alguien ya automatizó `archive`. Codegen (B) sigue siendo útil DESPUÉS, para descubrir locators. El orden profesional: manifest → plan → código. Duplicar trabajo es el bug más caro de los equipos.

Todo lo que aprendiste ya vive en KATA

LO APRENDISTE COMO...	EN KATA SE LLAMA...	DECK
Clase con métodos (<code>LoginPage</code>)	Component de capa L3	Coding Classes
Método <code>login()</code> del POM	ATC con ID de trazabilidad	Classes + Pro
<code>extends + super</code>	La columna vertebral: L1 → L2 → L3	Coding Classes
Tu <code>fixtures.ts</code> con (<code>{ loginPage }</code>)	TestFixture L4: (<code>{ ui }</code>), (<code>{ api }</code>)	Playwright Pro
Helpers que repetías en cada POM	UiBase / ApiBase (L2) — heredados	Pro + hoy
Credenciales hardcodeadas 🤪	TestContext (L1) — config por ambiente	hoy
<code>Date.now()</code> para emails únicos	Faker en TestContext	Dojo Lab
Tu rama <code>test/*</code> + PR	El flujo de entrega del repo (idéntico)	Dev Craft

Tu primera semana en el oficio

1

Onboarding del repo

Pide el tour guiado: `/agentic-qa-onboard` en el repo te explica el flujo completo de QA (con visuales como estos).

2

Primer ticket acompañado

Un ticket real con `/test-automation`: la IA propone el plan, TÚ lo revisas con todo tu criterio nuevo, y juntos escriben el ATC.

3

Tu primer PR al framework

Rama `test/*`, commits semánticos, PR con review — el flujo de Dev Craft, ahora en el repo del equipo.



Sigue practicando

El dojo no se va: nuevos specs, nuevos POMs, nuevos retos. Los 6 decks quedan contigo para repasar.



Revisa a otros

El músculo de criterio crece revisando PRs ajenos. Ofrecete de reviewer desde la semana 1.



Recuerda

KATA = forma ensayada hasta que sale sola. La primera semana se siente torpe. La cuarta, natural. La octava... lo enseñas tú.

FIN DE LA SERIE

De "Ana" a Quality Engineer

— / 3 • — / 4



Coding Base



Coding Classes



Dojo Lab I



Playwright Pro



Dev Craft



KATA Bridge

Un valor → una variable → una función → una clase → un Page Object → fixtures → git
→ **un framework profesional que ahora sabes leer, juzgar y dirigir.**

Bienvenida/o al oficio. Nos vemos en el repo. 🤖