LAB PRÁCTICO · QA ENGINEERING

Dojo Lab I 

Hoy no hay simuladores. Hoy tu código controla un **navegador real** contra una **app real**.
De cero a 5 tests verdes en tu máquina.

`7 misiones 🎯` `codegen` `UI mode` `trace viewer` `dojo.upexgalaxy.com`

REGLAS DEL JUEGO

Así funciona un lab de misiones

🎯

7 misiones

Cada una con objetivo, comandos copiables (click = copiar 📋) y **criterio de éxito verificable** — sabrás exactamente cuándo la lograste.

✅

Marca tu progreso

Checkbox en cada misión. Se guarda en tu navegador: cierra y vuelve cuando quieras, tu avance sigue ahí.

💡

Pistas colapsables

Atáscate primero, pide pista después. El atasco es parte del entrenamiento — en el trabajo real no hay slide con la respuesta.

⌨️

Laptop abierta

Esto NO es para mirar. Cada misión se hace en TU terminal y TU editor, en paralelo.

🐢

Ritmo propio

Nadie se queda atrás: el archivo es tuyo para siempre. Lo que no termines hoy, lo terminas en casa.

📚

Prerrequisitos

Coding Base ✓ y Coding Classes ✓ (o saber: variables, funciones, async/await, clases). Sin eso, primero esos decks.

EL CAMPO DE ENTRENAMIENTO

Conoce tu dojo 🏯

🌐 dojo.upexgalaxy.com

App real construida A PROPÓSITO para práctica de QA automation. Rompe lo que quieras — para eso existe.

`Login / Registro` `24 componentes UI` `Dashboard kanban` `API + docs`

🔑 Credenciales demo

email: **testuser@upex.dev**
password: **Test123!**

También puedes registrar tu propio usuario — eso será uno de tus tests.

🧭 **Regla de oro del lab:** los códigos de ejemplo en estas slides muestran la FORMA correcta. Los selectores exactos de cada elemento los descubres TÚ con codegen y el inspector — así funciona el trabajo real: nadie te regala los selectores.

🗺️ Tu plan de vuelo

M1-M2 → kit + proyecto
M3-M4 → grabar y ejecutar tu primer test
M5 → romper y diagnosticar
M6 → 4 specs a mano
M7 → 🏆 suite completa verde

MISIÓN 1

Instala el kit del oficio

☐ Completada

🎯 Tener **Node.js** (el motor que ejecuta JavaScript fuera del navegador) y **VS Code** (tu editor) funcionando.

1. Descarga e instala (versión LTS / estable):

`nodejs.org` → Node.js LTS · `code.visualstudio.com` → VS Code

2. Abre una terminal y verifica:

```
$ node --version                    📋 click = copiar
```

```
$ npm --version                     📋
```

```
$ node --version
v22.16.0   ← cualquier v18+ sirve
$ npm --version
10.9.2
```

💡 **Pista: "command not found"**

✅ **Criterio de éxito:** ambos comandos responden con un número de versión (Node v18 o superior).

MISIÓN 2

Crea tu proyecto Playwright

☐ Completada

🎯 Generar el proyecto completo con el instalador oficial y ver los tests de ejemplo **pasar en verde**.

```
$ npm init playwright@latest mi-dojo-lab
```

El asistente pregunta — responde así:

¿TypeScript o JavaScript?	**TypeScript** ✓
¿Carpeta de tests?	**tests** (default)
¿GitHub Actions workflow?	**false** (hoy no)
¿Instalar browsers?	**true** ✓ (tarda unos minutos)

Entra al proyecto y ejecuta los tests de ejemplo:

```
$ cd mi-dojo-lab
```

```
$ npx playwright test
```

```
Running 6 tests using 4 workers

  6 passed (8.3s)
```

💡 **Pista: ¿qué acaba de pasar?**

✅ **Criterio de éxito:** `npx playwright test` termina con **6 passed**.

RECONOCIMIENTO DEL TERRENO

Tour del proyecto: ya conoces todo esto

```
mi-dojo-lab/
├─ package.json          ← la "cédula" (Coding Base, slide 32)
├─ playwright.config.ts  ← browsers, baseURL, timeouts
├─ node_modules/         ← herramientas instaladas (no se toca)
├─ tests/
│  └─ example.spec.ts    ← ⭐ ábrelo en VS Code AHORA
└─ tests-examples/       ← demo más grande (de lectura)
```

tests/example.spec.ts

```ts
import { test, expect } from "@playwright/test";

test("has title", async ({ page }) => {
  await page.goto("https://playwright.dev/");
  await expect(page).toHaveTitle(/Playwright/);
});
```

Inventario de lo que reconoces: `import` ✓ `test()` ✓ `async ({ page })` ✓ `await` ✓ `expect` ✓ — **cero sorpresas**. Las dos sesiones anteriores fueron exactamente para este momento.

MISIÓN 3

Graba tu primer test con codegen

☐ Completada

🎯 Usar la **grabadora oficial**: tú haces click en el dojo, Playwright escribe el código por ti.

```
$ npx playwright codegen https://dojo.upexgalaxy.com
```

Se abren DOS ventanas: el navegador y el inspector (donde aparece el código en vivo). Ahora, en el navegador:

1. Navega al **login**
2. Escribe email y password demo
3. Click en el botón de entrar
4. Mira el inspector: 👀 **el código se escribió solo**

5. Copia el código grabado a un archivo nuevo: `tests/login.spec.ts` — créalo en VS Code y pega adentro de un `test("login", async ({ page }) => { ... })` si codegen no lo envolvió ya.

💡 **Pista: el inspector no muestra código**

⚠️ **Codegen es un borrador, no un producto.** Graba TODO lo que haces (incluso clicks accidentales) y no agrega aserciones inteligentes. La próxima slide: limpiarlo.

✅ **Criterio de éxito:** tienes `tests/login.spec.ts` con el flujo de login grabado por codegen.

DEL BORRADOR AL TEST DIGNO

Limpia la grabación — y agrégale el expect

❌ LO QUE GRABA CODEGEN (BORRADOR)

```
await page.goto("https://dojo.upexgalaxy.com/");
await page.getByRole("link", { name: "Login" }).click();
await page.getByLabel("Email").click();   ← click inútil
await page.getByLabel("Email").fill("testuser@upex.dev");
await page.getByLabel("Password").click(); ← otro
await page.getByLabel("Password").fill("Test123!");
await page.getByRole("button", { name: "Sign in" }).click();
// ...y AQUÍ TERMINA. ¿Dónde está la verificación? 😱
```

✅ TU VERSIÓN LIMPIA (CON ASERCIÓN)

```
test("login exitoso en el dojo", async ({ page }) => {
  await page.goto("https://dojo.upexgalaxy.com/login");
  await page.getByLabel("Email").fill("testuser@upex.dev");
  await page.getByLabel("Password").fill("Test123!");
  await page.getByRole("button", { name: /sign in/i }).click();

  await expect(page).toHaveURL(/dashboard/);
});
```

Limpieza: clicks inútiles fuera, goto directo al login, y el **expect que codegen nunca escribe** — sin aserción no hay test, hay paseo.

MISIÓN 4

Córrelo con tus propios ojos

☐ Completada

🎯 Ejecutar tu login.spec.ts viendo el navegador moverse, y conocer el **UI mode** — tu centro de control.

1. Con navegador visible (headed):

```
$ npx playwright test login --headed
```

2. El modo que vas a AMAR — UI mode:

```
$ npx playwright test --ui
```

🕹️ En UI mode prueba esto

▶ Corre tu test desde la lista

⏱️ Click en cada paso de la línea de tiempo: **viaje en el tiempo** — ves la página ANTES y DESPUÉS de cada acción

🔍 Pestaña "Locator": pasa el mouse sobre la página y te dice el locator de cada elemento

💡 **Pista: "login" no corre nada**

✅ **Criterio de éxito:** viste el navegador hacer login solo, y en UI mode recorriste la línea de tiempo paso a paso.

MISIÓN 5

Rómpelo a propósito y diagnostica

☐ Completada

🎯 Provocar un fallo controlado, **leer el error sin pánico** y usar el trace viewer para ver la "caja negra" del accidente.

1. En tu login.spec.ts cambia la password a `"incorrecta"`

2. Corre con la caja negra activada:

```
$ npx playwright test login --trace on
```

3. Falla (esperado ✓). Abre el reporte y el trace:

```
$ npx playwright show-report
```

```
✗ login.spec.ts:3 › login exitoso

Error: expect(page).toHaveURL(/dashboard/)
Expected pattern: /dashboard/
Received string:  "…/login"  ← se quedó en login
  at login.spec.ts:9:24
```

En el reporte, click en el test rojo → **Trace**: línea de tiempo con screenshots de cada paso, red, consola. La caja negra completa del fallo.

💡 **Pista: leer el error como detective**

✅ **Criterio de éxito:** encontraste en el trace el screenshot del momento del fallo (la página que se quedó en login). Restaura la password buena y vuelve a verde antes de seguir.

★ QUIZ 1 · Leer errores

Tu test falla con: `TimeoutError: waiting for getByLabel("Email")` — ¿qué significa?

A La app está caída — avisar a infraestructura

B Playwright esperó y nunca encontró un elemento con label "Email" — locator equivocado o página distinta

C Hay que subir el timeout a 60 segundos y listo

💡 TimeoutError en un locator casi siempre = "busqué y no existe (con ese nombre, en esta página)". Primer reflejo: abrir UI mode o trace y MIRAR qué página había y cómo se llama el elemento realmente. Subir el timeout (opción C) solo hace que el error tarde más en aparecer — es esconder el problema bajo la alfombra.

MISIÓN 6

Escribe 4 specs a mano

☐ Completada

🎯 Sin codegen esta vez (úsalo solo para consultar locators): escribe tú el código. Un archivo por funcionalidad.

SPEC	PASOS (ACT)	ASERCIÓN MÍNIMA (ASSERT)
1 · **login-invalid.spec.ts**	Login con password mala	El mensaje de error es visible
2 · **register.spec.ts**	Registra un usuario nuevo (email único: usa `Date.now()`)	Llegas logueado o ves confirmación
3 · **navigation.spec.ts**	Desde el home, navega a la página de componentes	La URL y el título cambian
4 · **components.spec.ts**	Interactúa con un componente (checkbox, select...)	El estado del componente cambió

💡 **Pista: email único para el registro**

✅ **Criterio de éxito:** 4 archivos nuevos en tests/, cada uno verde individualmente: `npx playwright test <nombre>`.

LA FORMA DE TODO TEST DIGNO

Patrón AAA — tu brújula para la Misión 6

```
test("login inválido muestra error", async ({ page }) => {
  // 1. ARRANGE — prepara el escenario
  await page.goto("https://dojo.upexgalaxy.com/login");

  // 2. ACT — ejecuta la acción bajo prueba
  await page.getByLabel("Email").fill("testuser@upex.dev");
  await page.getByLabel("Password").fill("incorrecta");
  await page.getByRole("button", { name: /sign in/i }).click();

  // 3. ASSERT — verifica el resultado esperado
  await expect(page.getByText(/invalid|incorrect/i)).toBeVisible();
});
```

🎬 Arrange

Precondiciones de tu test case manual: "estando en la página de login..."

⚡ Act

Los pasos. UNA acción principal bajo prueba por test.

🔍 Assert

El resultado esperado. **Test sin expect no es test — es paseo.**

SABIDURÍA DESTILADA

Tips de supervivencia 🧰

⏳

Nunca uses sleep

Playwright **espera solo** a que los elementos aparezcan (auto-wait). `waitForTimeout(5000)` = test lento Y flaky. Si crees necesitarlo, falta un expect intermedio.

🎯

getByRole > selectores CSS

`getByRole("button", { name: "Sign in" })` sobrevive rediseños; `.btn-primary > div:nth-child(2)` muere mañana. Locators semánticos = mantenimiento barato.

1️⃣

Un test, un objetivo

Título describe UN comportamiento verificable. Si necesitas "y" en el título, divide en dos tests.

🔁

Corre en cada cambio

Escribe 3 líneas → corre. Feedback inmediato = errores chicos y fáciles. Escribir 50 líneas sin correr = arqueología de bugs.

MISIÓN 7 · BOSS 🏆

Suite completa en verde + reporte

☐ Completada

🎯 Tus 5 specs (login + los 4 de la M6) corriendo juntos como **suite**, y el reporte HTML como evidencia.

```
$ npx playwright test
```

```
$ npx playwright show-report
```

```
Running 15 tests using 4 workers
  5 specs × 3 browsers

  15 passed (31.2s)   🎉
```

📊 En el reporte HTML explora

- ✅ Lista de tests por browser (¡corrieron en 3!)
- ⏱️ Duración de cada test
- 📷 Click en cualquiera → pasos detallados

Este reporte es lo que un QA Engineer adjunta como **evidencia de ejecución**

💡 **Pista: uno falla solo en WebKit/Firefox**

✅ **Criterio de éxito:** reporte HTML mostrando tu suite completa en verde (15 passed = 5 specs × 3 browsers).

★ QUIZ 2 · El integrador

Tu suite pasó 15/15 hoy. Mañana corre de nuevo SIN cambios de código… y un test falla. ¿Primera hipótesis profesional?

A Playwright se corrompió — reinstalar todo

B Borrar ese test, total los otros 14 pasan

C Algo cambió fuera del código: la app, los datos, el timing — abrir el trace y comparar contra la corrida verde

💡 Test que falla sin cambios de código = señal de que algo SÍ cambió en otra parte: deploy de la app, datos de prueba consumidos (¿tu registro reusó un email?), timing de red. El trace te dice cuál. Si va y viene sin causa, se llama **flaky** — y diagnosticarlos es una habilidad estrella del QA Engineer (la entrenarás en Playwright Pro).

DEBRIEF

Misiones completadas

Tarea

Termina las misiones pendientes — tu progreso quedó guardado en este archivo.

Reto extra

Spec de logout + spec del kanban del dashboard (drag & drop: investígalo).

Próxima parada

Playwright Pro: hooks, fixtures, tu LoginPage (POM) conectada a tests reales, y API testing contra el dojo.