TEST AUTOMATION ENGINEER – FORMACIÓN INTEGRAL

{desafío} latam_

CURSO:

TEST AUTOMATION ENGINEER – FORMACIÓN INTEGRAL

- Módulo 1: FUNDAMENTOS DEL TESTING DE SOFTWARE
- Módulo 2: CONTROL DE VERSIONES Y ENTORNOS DE DESARROLLO
- Módulo 3: FUNDAMENTOS DE PROGRAMACIÓN APLICADOS AL TESTING (JAVASCRIPT)
- Módulo 4: CARACTERISTICAS DE CYPRESS – AUTOMATIZACIÓN WEB I
- **Módulo 5: CARACTERISTICAS DE CYPRESS – AUTOMATIZACIÓN WEB II**
- Módulo 6: PRUEBAS DE APIS CON POSTMAN Y SUPERTEST

Te encuentras aquí

{desafío} latam_

CURSO:

TEST AUTOMATION ENGINEER – FORMACIÓN INTEGRAL

- Módulo 7: HERRAMIENTAS DE PLAYWRIGHT Y PRUEBAS CON MÚLTIPLES NAVEGADORES
- Módulo 8: DESARROLLO GUIADO EN EL COMPORTAMIENTO (BDD) CON CUCUMBER.JS
- Módulo 9: HERRAMIENTAS DE AUTOMATIZACIÓN MÓVIL CON APPIUM
- Módulo 10: HERRAMIENTAS DE INTEGRACIÓN DE PRUEBAS EN CI/CD
- Módulo 11: HERRAMIENTAS DE DOCKER, ENTORNOS VIRTUALIZADOS Y PRUEBAS EN LA NUBE
- HERRAMIENTAS DE AUTOMATIZACIÓN DE UN FLUJO COMPLETO WEB + API + CI/CD

{desafío} latam_

Módulo 5: CARACTERISTICAS DE CYPRESS – AUTOMATIZACIÓN WEB II.

{desafío} latam_

OBJETIVO ESPECÍFICO DEL MÓDULO

- CONOCER CARACTERISTICAS DE CYPRESS AUTOMATIZACIÓN WEB II, DE ACUERDO A LAS APLICACIONES WEB, MÓVILES Y APIS.

{desafío} latam_

¿Qué aspectos considera clave para escalar y mantener una suite de pruebas automatizadas a medida que una aplicación crece en complejidad y variedad de entornos?

HOOKS Y COMANDOS PERSONALIZADOS

- **Hooks (ganchos):**
- **before():** se ejecuta una vez antes de todos los tests
- **beforeEach():** antes de cada it()
- afterEach() y after() para limpieza
- Útiles para login, navegación inicial, limpieza de datos, etc.

- **Comandos personalizados:**
- Definidos en cypress/support/commands.js

- **Ejemplos:**

```
Cypress.Commands.add('login', (user, pass) => {
  cy.get('#user').type(user)
  cy.get('#pass').type(pass)
  cy.get('#login').click()
})
```

{desafío} latam_

- Permite reutilizar lógica y reducir duplicación de código.

- **Organización avanzada:**
- Dividir archivos de prueba por módulos, flujos o áreas (/login, /checkout, /admin)
- Mantener carpetas limpias, con nombres consistentes y documentación breve en cada archivo

PAGE OBJECT MODEL (POM)

- **¿Qué es?**
- Patrón de diseño que separa la lógica de interacción con la página web del test.
- Cada página (o componente) se representa como una clase JS.

- **Ventajas:**
- Código más limpio y reutilizable
- Cambios en la UI solo afectan un archivo
- Mayor mantenibilidad

{desafío} latam_

- **Ejemplo básico:**

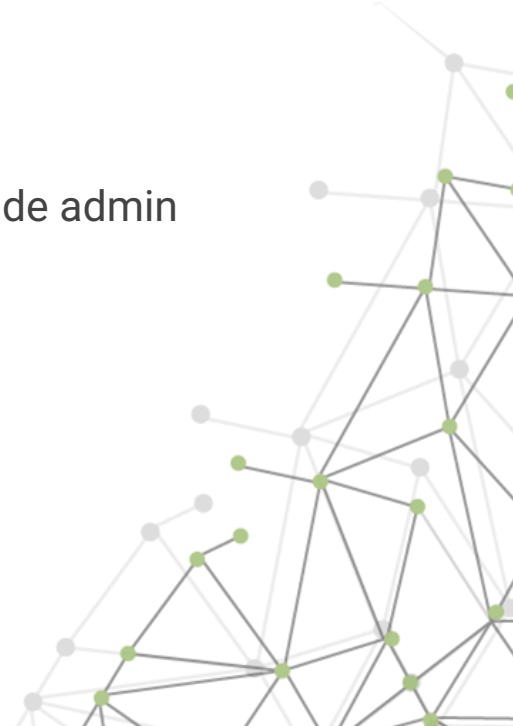
```
class LoginPage {
  visit() {
    cy.visit('/login')
  }
  fillUsername(name) {
    cy.get('#username').type(name)
  }
  fillPassword(pass) {
    cy.get('#password').type(pass)
  }
  submit() {
    cy.get('#login-button').click()
  }
}
export default LoginPage
```

{desafío} latam_

AUTOMATIZACIÓN DE FLUJOS END-TO-END COMPLEJOS

- **E2E (End-to-End):**
- Pruebas que simulan el comportamiento del usuario desde el inicio al final de un proceso

- **Ejemplo de flujo complejo:**
- Login → añadir productos → checkout → validación en dashboard de admin



- **Recomendaciones:**
- Dividir el flujo en pasos reutilizables
- Usar comandos personalizados o clases POM
- Validar datos en cada paso intermedio para asegurar consistencia
- Incluir evidencias en cada etapa (cy.screenshot())

- Las pruebas E2E validan la integración de varios sistemas y son críticas para asegurar que todo el flujo del usuario final funciona correctamente.

ENTORNOS MÚLTIPLES Y PRUEBAS CONDICIONALES

- **Manejo de entornos (dev, QA, staging, producción):**
- **Configuración en cypress.config.js o variables de entorno:**

```
baseUrl: process.env.BASE_URL || 'http://localh
```

- O usando .env y librerías como dotenv
- **Pruebas condicionales:**
- **Ejecutar pruebas solo en ciertos entornos o bajo ciertas condiciones:**

```
if (Cypress.env('ENV') === 'QA') {
  it('debe ejecutar solo en QA', () => {...})
}
```

- Útil para evitar pruebas destructivas en ambientes sensibles

- El control de entornos permite flexibilidad y seguridad al automatizar pruebas en proyectos reales.

INTERCEPTAR Y SIMULAR RESPUESTAS API

- **Interceptar llamadas HTTP:**
- Se usa cy.intercept() para observar, modificar o simular una respuesta

- **Ejemplo:**

```
cy.intercept('GET', '/api/usuarios', { fixture:
```

{desafío} latam_

- **Mockeo (simulación):**
- Útil cuando el backend no está listo o cuando se desea forzar un resultado específico (éxito, error, etc.)

- **Ventajas:**
- Control total del flujo
- Aislamiento de fallos
- Pruebas más rápidas y deterministas

- La interceptación y el mockeo permiten que las pruebas sean más confiables incluso en etapas tempranas del desarrollo.

Recuerde, debe desarrollar los ejercicios prácticos del Módulo…

{desafío} latam_

¿Cómo impacta el uso de patrones como POM, hooks personalizados y mockeo de APIs en la eficiencia, claridad y estabilidad de sus pruebas automatizadas en escenarios complejos y variados?

{desafío} latam_

Éxito en la evaluación parcial y en la Prueba Final...

{desafío}
latam_

Academia de
talentos digitales