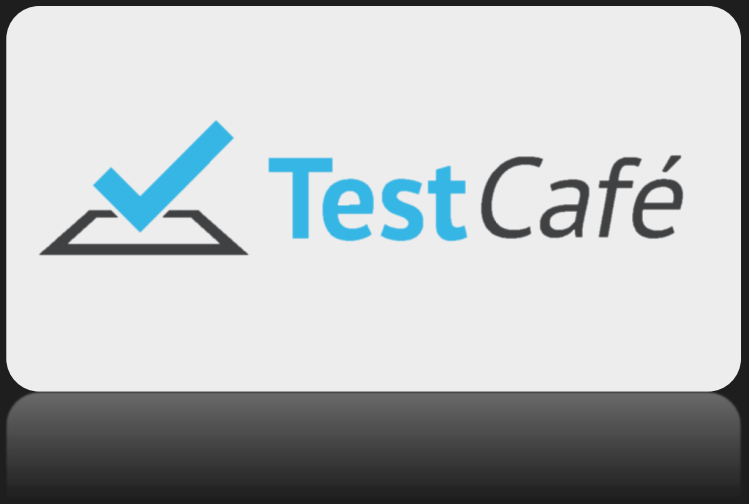
Taller de Automatización y testing web con TestCafe



Por: Luis Manuel Arellano Muñoz

Agenda del Taller

1

Introducción y Contexto

Fundamentos del testing E2E

2

Herramientas de Automatización

Comparativa y TestCafe

3

Instalación y Configuración

Setup del entorno

4

Práctica Guiada

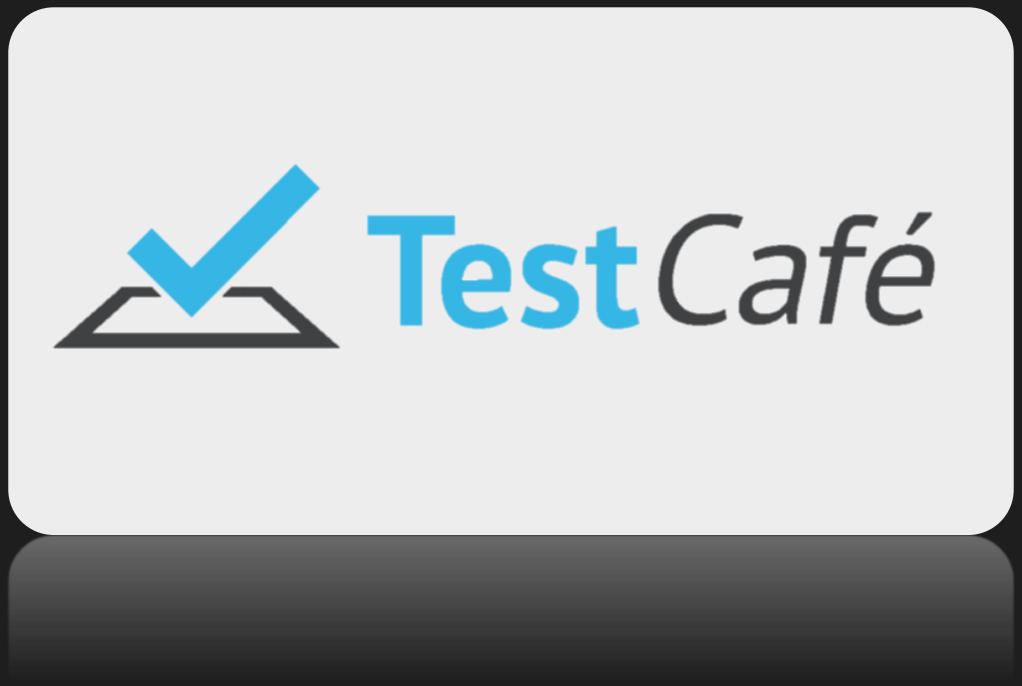
Hands-on con TestCafe

5

Consideraciones Finales

Mejores prácticas

Demostración



1.1 ¿Qué es el Testing de Software y por qué es importante?

- Proceso sistemático para verificar que un software funciona correctamente.

Ahorra Costos

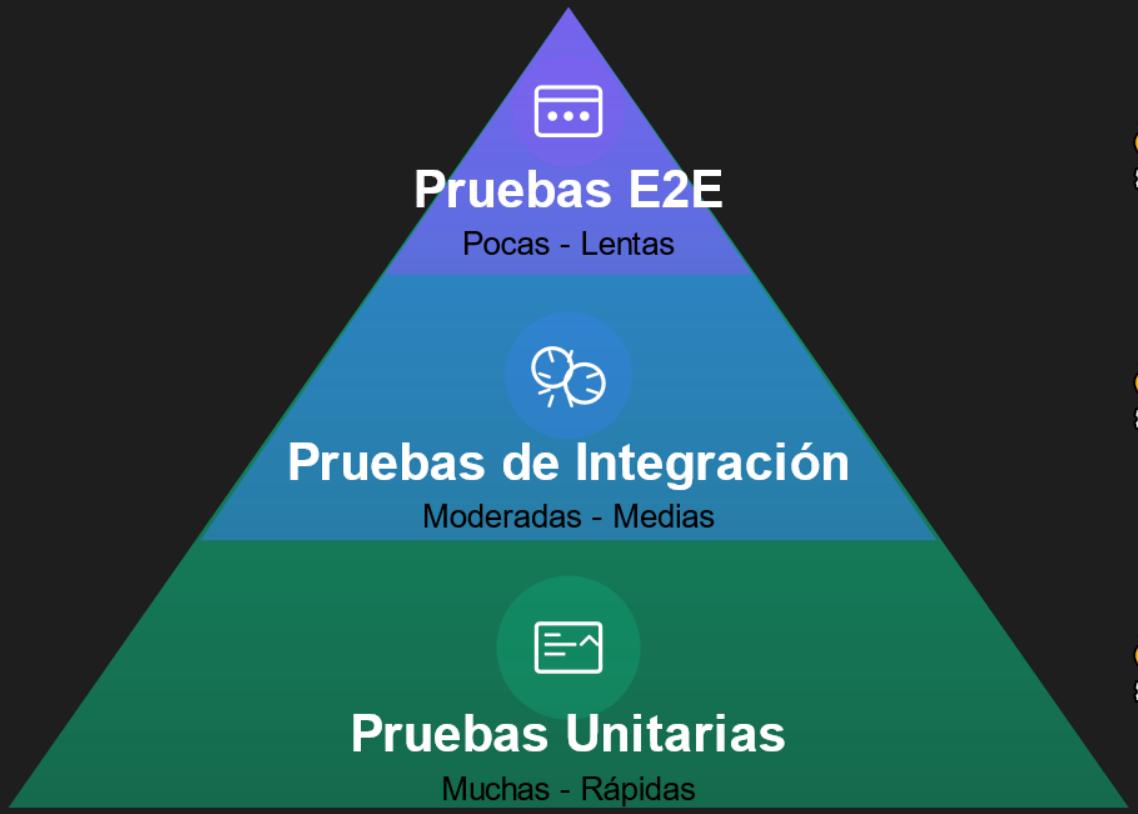
Garantiza Calidad

Protege la Reputación

Seguridad

Mejora Continua

1.2 Tipos de pruebas



Alto Costo
Alta Complejidad

Costo Medio
Complejidad Media

Bajo Costo
Baja Complejidad

1.3 Prueba manual y automatizada

Pruebas Manuales

Lentas
Requieren tiempo humano

Costosas
A largo plazo

Propensas a Error
Error humano posible

No Repetibles
Difíciles de escalar

UX/UI Testing
Mejor para experiencia

Pruebas Automatizadas

Rápidas
Ejecución en segundos

Económicas
A largo plazo

Consistentes
Sin error humano

Repetibles
Fáciles de escalar

Inversión Inicial
Requiere setup y código

1.4 Beneficios de la automatización E2E

Cobertura Completa

Prueba flujos completos
de inicio a fin

Detección Temprana

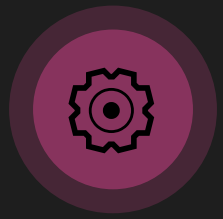
Encuentra bugs antes
de producción

Confianza en Deploy

Despliega con seguridad
y tranquilidad

Ahorro de Tiempo

Ejecución automática
en minutos



Integración CI/CD

Automático en cada
commit del código

2.1 Herramientas de automatización

Cypress

Framework moderno y rápido
con excelente experiencia
de desarrollo y debugging

Selenium

Estándar de la industria,
soporte multi-lenguaje y
todos los navegadores

Playwright

Moderno, rápido y confiable.
Auto-wait inteligente y
soporte multi-navegador

TestCafe

Sin WebDriver, setup simple.
Ejecución paralela gratuita
y auto-wait integrado

2.2 Ventajas de TestCafe

Zero Config
Sin WebDriver
ni plugins

Fácil Aprendizaje
JavaScript/TypeScript
sintaxis intuitiva

Auto-Wait
Espera automática
de elementos

Paralelización
Gratis y nativa
múltiples navegadores

Multi-Navegador
Chrome, Firefox, Safari
Edge, y más

Reportes Claros
Screenshots y videos
automáticos en fallos

CI/CD Ready
Integración fácil con
Jenkins. GitHub Actions

Selectores Smart
API potente para
selección de elementos

3.1 Pre-requisitos

Node.js

Versión 12 o superior
(Recomendado: LTS)

NPM / Yarn

Gestor de paquetes
(incluido con Node.js)

Navegador

Chrome, Firefox, Safari
o Edge instalado

Editor de Código

VS Code, WebStorm
o cualquier IDE

3.2 Instalación

Instalar Node.js y NPM

Descargar e instalar desde nodejs.org
Configurar variables de entorno PATH

Instalación Global (Opcional)

Instala TestCafe globalmente en el sistema

```
npm install -g testcafe
```

Instalación Local (Recomendado)

Instala TestCafe como dependencia del proyecto

```
npm install --save-dev testcafe
```

Verificar Instalación

Confirma que TestCafe está instalado correctamente

```
testcafe --version
```

3.3 Comandos clave

Navegador

OBLIGATORIO

Especifica el navegador donde ejecutar las pruebas

```
testcafe chrome test.js
```

Archivo de Prueba

OBLIGATORIO

Ruta al archivo o directorio con las pruebas

```
testcafe chrome tests/
```

Screenshots (-s)

OPCIONAL

Captura screenshots automáticamente en fallos

```
testcafe chrome test.js -s takeOnFails=true
```

Concurrencia (-c)

OPCIONAL

Ejecuta pruebas en paralelo (número de instancias)

```
testcafe chrome test.js -c 3
```

Velocidad (--speed)

OPCIONAL

Reduce velocidad (1=rápido, 0.01=lento)

```
testcafe chrome test.js --speed 0.5
```

Reportes (-r)

OPCIONAL

Genera reportes en formato específico

```
testcafe chrome test.js -r json:report.json
```

Modo Headless

OPCIONAL

Ejecuta sin interfaz gráfica (ideal para CI/CD)

```
testcafe chrome:headless test.js
```

4.1 Anatomía de un test

```
import { Selector } from 'testcafe';

fixture  `Prueba Básica con TestCafe`
  .page   `https://ejemplo.com`;

test  ('Completar formulario',
     async  t => {

  const  usernameInput =        Selector   ('#username');
  const  passwordInput =        Selector   ('#password');
  const  submitButton =      Selector    ('button');

  await   t
    .typeText   (usernameInput, 'usuario')
    .typeText   (passwordInput, 'pass123')
    .click  (submitButton)

    .expect  ( Selector   ('.welcome-message')
      .exists).ok()
    .expect  ( Selector   ('.welcome-message')
      .innerText).contains('Bienvenido');

});
```

1. Import
Importa funciones de TestCafe

2. Fixture
Agrupa tests y define URL inicial

3. Test
Define caso de prueba específico

4. Selectors
Identifica elementos del DOM

5. Actions
Simula interacciones del usuario

6. Assertions
Verifica resultados esperados

4.2 Selectores

1. Selector por ID (#)

El más específico - Recomendado

```
Selector('#username')
```

HTML

```
<input
id="username"       >
```

2. Selector por Clase (.)

Para elementos con clases CSS

```
Selector('.btn-primary')
```

HTML

```
<button
class="btn-primary"       >
```

3. Selector por Atributo ([...])

Cualquier atributo HTML

```
Selector('[data-testid="login"]')
```

HTML

```
<button
data-testid="login"         >
```

4. Selector por Etiqueta

Nombre del elemento HTML

```
Selector('button')
```

HTML

```
< button   >
Click me
```

5. Métodos de TestCafe para Selectores

```
Selector('button')    .withText('Login')
```

```
Selector('.item')     .nth(2)
```

```
Selector('#child')     .parent()
```

```
Selector('.parent')       .child('div')
```

Mejor Práctica: Usa data-testid para selectores estables

4.2 Practica guiada: Hola Mundo

Instrucciones:

1. Usa el archivo llamado `hola-mundo.test.js` que se encuentra en:

   ```
   https://github.com/luis16k/testcafe-scripts.git
   ```

2. Copia el código de ejemplo proporcionado

3. Ejecuta la prueba con el comando:

   ```
   testcafe chrome hola-mundo.test.js --speed 0.5
   ```

4. Observa cómo TestCafe abre el navegador

5. Completa la prueba

6. Ejecuta de nuevo

Tip:

Experimenta y anímate a cometer errores

4.3 Practica guiada: Formularios

Instrucciones:

1. Usa el archivo llamado `login.test.js` que se encuentra en:

   ```
   https://github.com/luis16k/testcafe-scripts.git
   ```

2. Copia el código de ejemplo proporcionado

3. Ejecuta la prueba con el comando:

   ```
   testcafe chrome login.test.js --speed 0.5
   ```

4. Observa cómo TestCafe abre el navegador

5. Completa la prueba, haciendo el **LOGOUT**

6. Ejecuta de nuevo

Pista:
Busca el menú hamburguesa y el botón de logout

5.1 Consideraciones finales

💪 Palabras de Motivación

El testing automatizado no es solo una habilidad técnica, es un **superpoder** que te permite construir software con confianza. Como futuros ingenieros, dominar esto les dará una **ventaja competitiva enorme** en el mercado laboral.

🎯 Recordatorio

No se trata de encontrar todos los bugs,
se trata de **prevenir** que los bugs lleguen a los usuarios.
Ustedes son los **guardianes** de la calidad del software.

❓ ¿Preguntas?