Informe técnico: Prueba de estrés con Locust

Proyecto: Sitio Web CucutocheViajero

Fecha: 01 de junio de 2025

Herramienta de prueba: Locust

Archivo analizado: `Locust_2025-06-01-18`

1) Introducción

Este informe detalla la ejecución de una prueba de estrés automatizada sobre el sitio web desarrollado con Flask, orientado a promover el turismo en municipios. La finalidad de esta prueba es identificar la capacidad de respuesta y robustez del servidor ante múltiples usuarios concurrentes accediendo simultáneamente a diversas rutas del sistema.

2) ¿Qué es Locust?

Locust es una herramienta de código abierto escrita en Python diseñada para realizar pruebas de carga y estrés sobre aplicaciones web y servicios HTTP. Permite definir el comportamiento simulado de usuarios mediante scripts (locustfile.py) y visualizar métricas en tiempo real a través de una interfaz web o consola.

3) Objetivo de la prueba

Evaluar el rendimiento del servidor Flask y su capacidad para manejar múltiples usuarios accediendo simultáneamente a:

- Página principal (/)
- Página de contacto (/contact)
- Página de departamentos (/departments)
- Departamentos específicos y municipios
- Envío de formularios de contacto (método POST)

4) Configuración de la prueba

- Herramienta: Locust (modo headless y gráfico)
- Archivo de configuración: locustfile.py
- Simulación de tareas: navegación y envío de datos
- Prueba ejecutada en entorno local (http://127.0.0.1:5000)
- Número de usuarios: variable durante la ejecución
- Tiempo de espera entre acciones: aleatorio entre 1 y 3 segundos
- Duración total: hasta la interrupción manual del test

5) Rutas incluidas en la simulación

Ruta	Tipo	Acción simulada
`/`	GET	Acceso a página principal
`/about`	GET	Acceso a información
`/contact`	GET	Formulario de contacto
`/contact`	POST	Envío de datos del formulario
`/departments`	GET	Listado de departamentos
`/departments/<dpto>`	GET	Acceso a departamento
`/departments/<dpto>/<municipio>`	GET	Acceso a municipio
`/municipality/<id>`	GET	Detalle de municipio
`/places/<id>`	GET	Detalle de lugar turístico

6) Resultados obtenidos (según el archivo HTML)

Los datos extraídos del reporte HTML indican lo siguiente:

a. Métricas globales

Métrica	Valor
Total de solicitudes	1862
Tasa de solicitudes	~260 por minuto
Fallos detectados	0 (cero errores)
Tiempo promedio respuesta	32.68 ms
Tiempo mínimo	2 ms
Tiempo máximo	274 ms

b. Desempeño por ruta (resumen)

Ruta	Tiempo promedio	Nº solicitudes	Errores
`/` (home)	~27 ms	210	0
`/about`	~28 ms	209	0
`/contact` (GET)	~31 ms	207	0
`/contact` (POST)	~32 ms	203	0
`/departments`	~31 ms	207	0
Rutas dinámicas	~35-45 ms	resto	0

7) Análisis de resultados

- El servidor respondió satisfactoriamente a todas las solicitudes, sin caídas ni errores (0.00% de fallos).
- El tiempo de respuesta promedio (32 ms) es muy bueno, indicando que el servidor maneja correctamente múltiples usuarios concurrentes.
- Las rutas dinámicas (como /places/1, /departments/X) mantuvieron una latencia aceptable incluso con solicitudes repetidas.
- La distribución de cargas fue uniforme, sin congestión en endpoints específicos.

8) Conclusión

La prueba de estrés demuestra que la aplicación Flask del proyecto "Cucutoche Viajero" se comporta de manera robusta y eficiente bajo una carga considerable. Todas las rutas críticas respondieron correctamente y los tiempos de respuesta fueron adecuados. No se identificaron cuellos de botella ni errores funcionales.

Esto valida que la arquitectura actual es apta para entornos productivos, bajo escenarios de carga moderada a alta.