

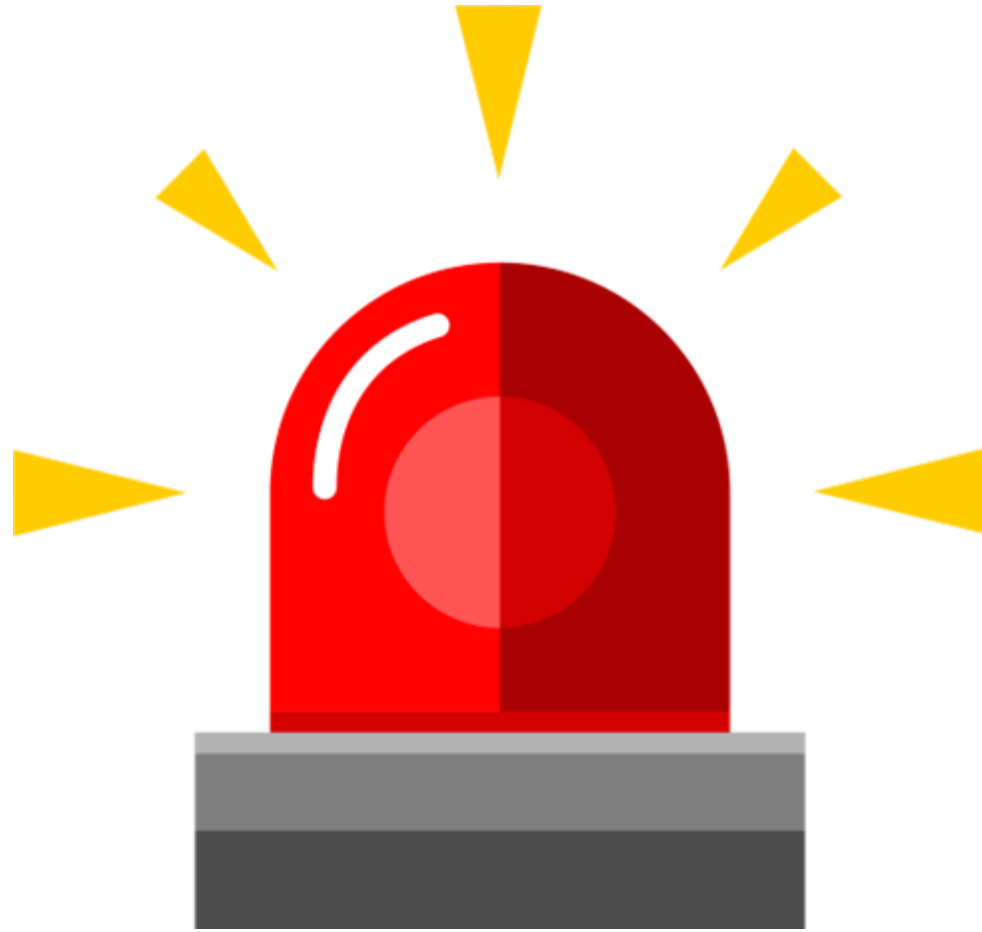
Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración



Práctica 4 Automatización de pruebas





Antes de continuar, hay que actualizar nuestro proyecto con el fork de la asignatura

1º) Añadir repositorio original como upstream

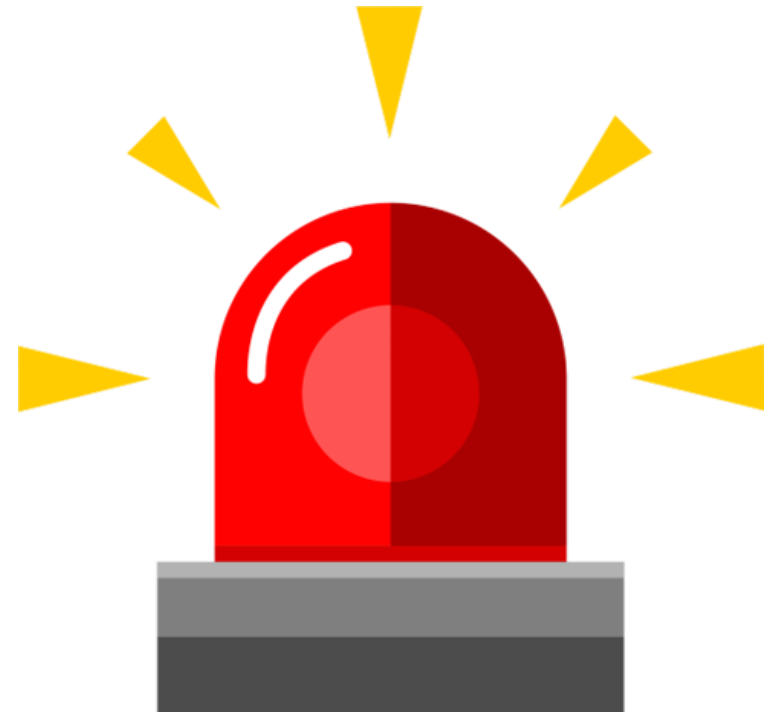
```
git remote add upstream https://github.com/EGCETSII/uvlhub
```


2º) Fetch del repositorio original


```
git fetch upstream
```

3º) Fusionar los cambios en tu fork

```
git checkout main  
git merge upstream/main
```




- 1. Introducción a la automatización de pruebas**
 - 2. Campo de entrenamiento**
 - 3. Automatización de pruebas en uvlhub**
 - 4. Y yo, ¿qué puedo hacer en el proyecto?**
 - 5. Ejercicio práctico: testing del bloc de notas**
- 

- 1. Introducción a la automatización de pruebas**
 2. Campo de entrenamiento
 3. Automatización de pruebas en uvlhub
 4. Y yo, ¿qué puedo hacer en el proyecto?
 5. Ejercicio práctico: testing del bloc de notas
- 

1. Introducción a la automatización de pruebas

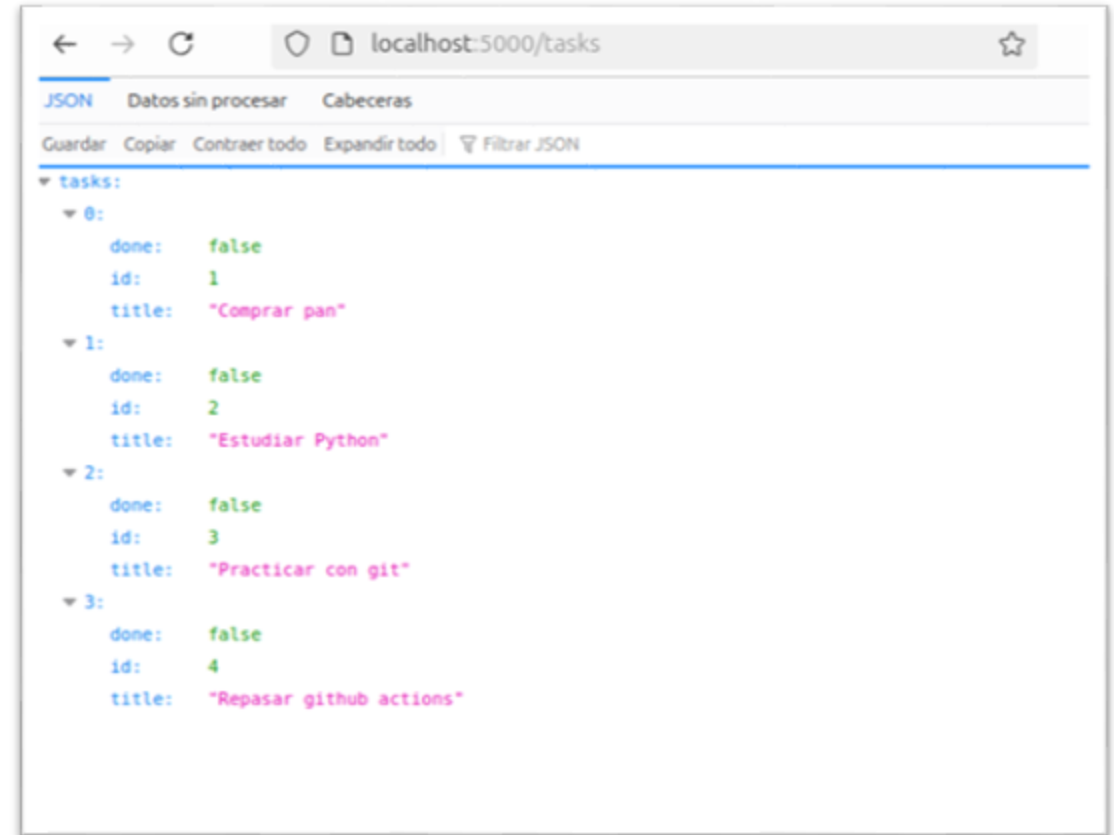
1. Las pruebas automatizadas **son más rápidas** que las manuales
2. Garantizan que se ejecuten **de la misma manera** siempre
3. Permiten probar más escenarios **de forma exhaustiva**
4. Detectan errores causados por cambios en el código (**regresión**)
5. **Reducen costos** al identificar errores **temprano**
6. Facilitan **iteraciones rápidas** en **CI/CD** con retroalimentación constante



1. Introducción a la automatización de pruebas
 - 2. Campo de entrenamiento**
 3. Automatización de pruebas en uvlhub
 4. Y yo, ¿qué puedo hacer en el proyecto?
 5. Ejercicio práctico: testing del bloc de notas
- 

2. Campo de entrenamiento

Mini app Flask para gestión de tareas



2. Campo de entrenamiento

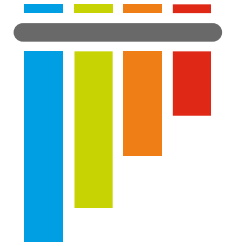
Mini app Flask para gestión de tareas

Estructura del proyecto

```
flask_testing_project/
├── app.py                # Archivo principal de la aplicación Flask
├── templates/           # Directorio que contiene la plantilla HTML
│   └── tasks.html       # Plantilla para mostrar y agregar tareas
├── tests/
│   ├── test_app.py     # Pruebas unitarias usando pytest
│   └── test_funcional.py # Pruebas funcionales con Selenium
└── locustfile.py       # Archivo para pruebas de carga con Locust
```

2. Campo de entrenamiento

Pruebas unitarias con pytest



pytest

```
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/jemole/Documentos/US/EGC/Practicas/testing_P4
collected 4 items

tests/test_app.py .... [100%]

===== 4 passed in 0.10s =====
```

2. Campo de entrenamiento

Pruebas de cobertura con pytest-cov

```
----- coverage: platform linux, python 3.12.3-final-0 -----
Name      Stmts  Miss  Cover
-----
app.py    26     8    69%
-----
TOTAL     26     8    69%
```

Coverage report: 69%

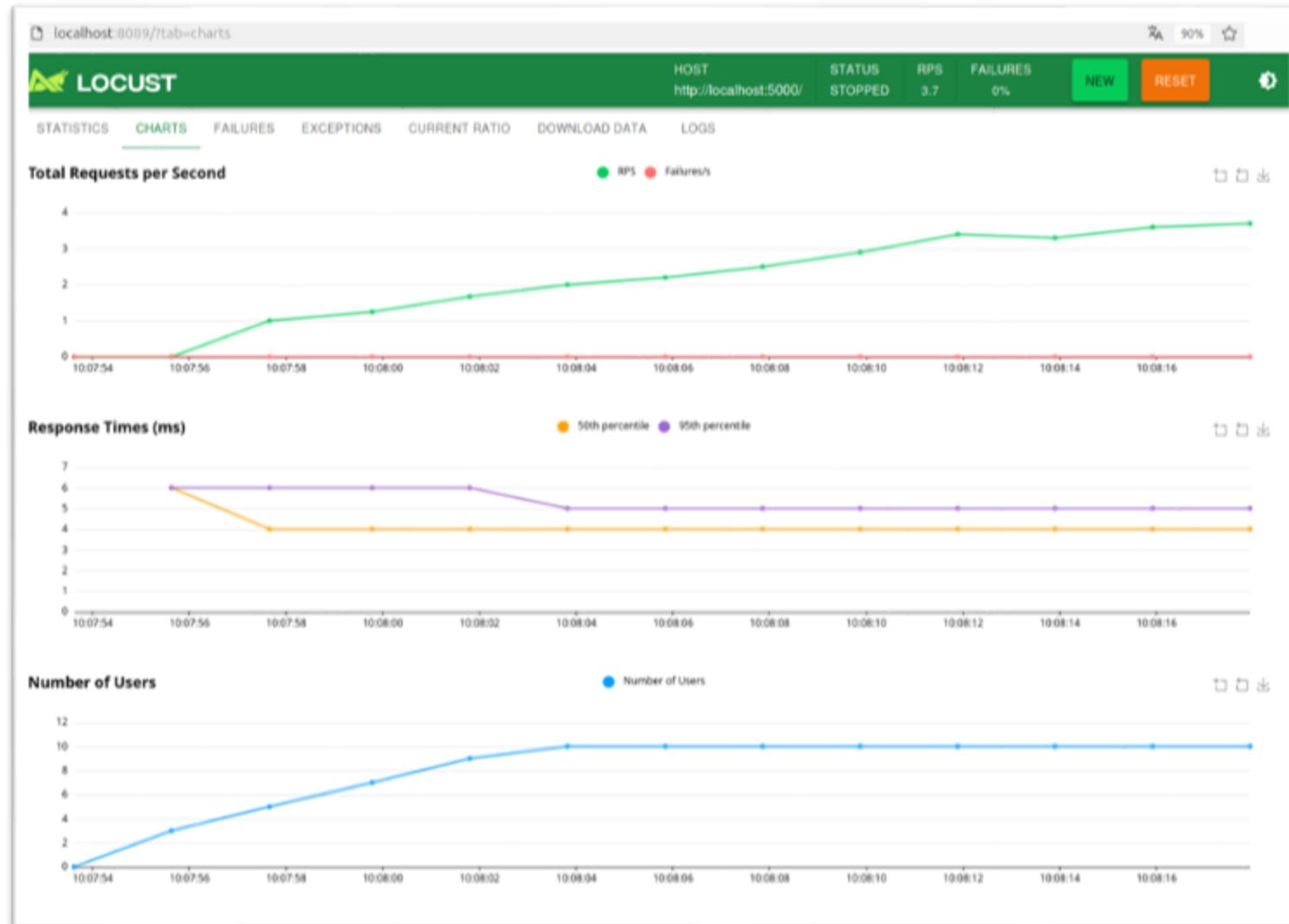
coverage.py v7.6.3, created at 2024-10-14 13:53 +0200

File ▲	function	statements	missing	excluded	coverage
app.py	task_list	1	1	0	0%
app.py	get_tasks	1	0	0	100%
app.py	add_task_html	6	6	0	0%
app.py	create_task	5	0	0	100%
app.py	(no function)	13	1	0	92%
Total		26	8	0	69%

coverage.py v7.6.3, created at 2024-10-14 13:53 +0200

2. Campo de entrenamiento

Pruebas de carga con Locust



2. Campo de entrenamiento

Pruebas de interfaz con Selenium



```
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/jemole/Documentos/US/EGC/Practicas/testing_P4
plugins: cov-5.0.0
collected 1 item


tests/test_funcional.py Iniciando el navegador Chromium...
Abriendo la aplicación web en localhost:5000...
Verificando que el título de la página es correcto...
Buscando el campo de entrada de nueva tarea...
Escribiendo 'Tarea de Selenium' en el campo de entrada...
Verificando que 'Tarea de Selenium' aparece en la lista de tareas...
Cerrando el navegador Chromium...

===== 1 passed in 1.51s =====
```

2. Campo de entrenamiento



https://1984.lsi.us.es/wiki-egc/index.php/Tutorial_Campo_de_entrenamiento

1. Introducción a la automatización de pruebas
 2. Campo de entrenamiento
 3. **Automatización de pruebas en uvlhub**
 4. Y yo, ¿qué puedo hacer en el proyecto?
 5. Ejercicio práctico: testing del bloc de notas
- 

3. Automatización de pruebas en uvlhub



docs.uvlhub.io

Test coverage

The `rosemary coverage` command facilitates running code coverage analysis for your Flask project using `pytest-cov`. This command simplifies the process of assessing test coverage.

TABLE OF CONTENTS

- 1 Test coverage of all modules
- 2 Test coverage of a specific module
- 3 Command Options
 - o `--html`

Test coverage of all modules

To run coverage analysis for all modules within the `app/modules` directory and generate an HTML report, use:

```
rosemary coverage
```

Test coverage of a specific module

If you wish to run coverage analysis for a specific module, include the module name:

```
rosemary coverage <module_name>
```

Rosemary CLI / Testing / Unit tests

Unit tests

TABLE OF CONTENTS

- 1 Testing all modules
- 2 Testing a specific module
- 3 Testing with an expression

Testing all modules

To run tests across all modules in the project, you can use the following command:

```
rosemary test
```

This command will execute all tests found within the `app/modules` directory of the project.

Testing a specific module

If you're focusing on a particular module and want to run tests only for that module, you can specify the module name as an argument:

```
rosemary test <module_name>
```

Rosemary CLI / Testing / Load tests

Load tests

TABLE OF CONTENTS

- 1 Introduction to Locust
 - o Key Features:
 - o Ramp-Up in Locust
- 2 Run all load tests
- 3 Run load tests from specific module
- 4 Stop Locust
- 5 Official documentation

Introduction to Locust

Locust is an open-source load testing tool that allows you to define user behavior with Python code and swarm your system with millions of simultaneous users. It's useful for testing the performance of web applications and identifying potential bottlenecks.

Key Features:


- Scalability: Capable of simulating millions of users.
- Flexibility: User behavior can be defined with simple Python code.
- Real-time Monitoring: Provides real-time statistics and metrics during the test.

Rosemary CLI / Testing / GUI tests

GUI tests

TABLE OF CONTENTS

- 1 Introduction to Selenium
- 2 Interface testing in local environment
- 3 Interface testing in Docker and Vagrant environment
 - o Activate the virtual environment
 - o Install dependencies
 - o Run test
- 4 Selenium IDE
 - o Installation
 - o Recording a test
 - o Playback the recorded test
 - o Exporting the test script
 - o Using the test in the project
- 5 Using WSL2 (Windows Subsystem for Linux) and WebDriver
 - o Install Google Chrome version 114
 - o Install ChromeDriver
 - o Unzip and make ChromeDriver executable
 - o Move the ChromeDriver executable to the WSL2 path

1. Introducción a la automatización de pruebas
 2. Campo de entrenamiento
 3. Automatización de pruebas en uvlhub
 4. **Y yo, ¿qué puedo hacer en el proyecto?**
 5. Ejercicio práctico: testing del bloc de notas
- 

4. Y yo, ¿qué puedo hacer en el proyecto?




¡Diseña los tests de tus propias funcionalidades!

Diseña tests unitarios y de integración

Diseña tests de carga con Locust (o similar)

Diseña tests de vista con Selenium (o similar)

1. **Introducción a la automatización de pruebas**
 2. **Campo de entrenamiento**
 3. **Automatización de pruebas en uvlhub**
 4. **Y yo, ¿qué puedo hacer en el proyecto?**
 5. **Ejercicio práctico: testing del bloc de notas**
- 

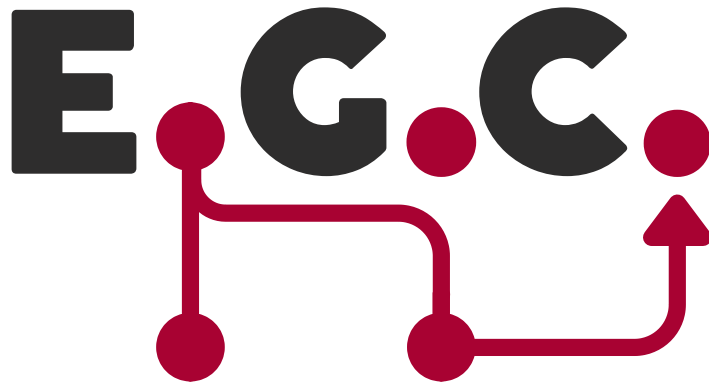
5. Ejercicio práctico: testing del bloc de notas



Ejercicio 1: Realizar testing unitario y de integración

Ejercicio 2: Realizar testing de carga con Locust

Ejercicio 3: Realizar testing de Interfaz con Selenium



Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración



Escuela Técnica Superior de
Ingeniería Informática

¡Gracias!

*“Si pds l3r 3st0, tns n bu3n ftro
en prb4s s0ftwr.”*

- Anónimo