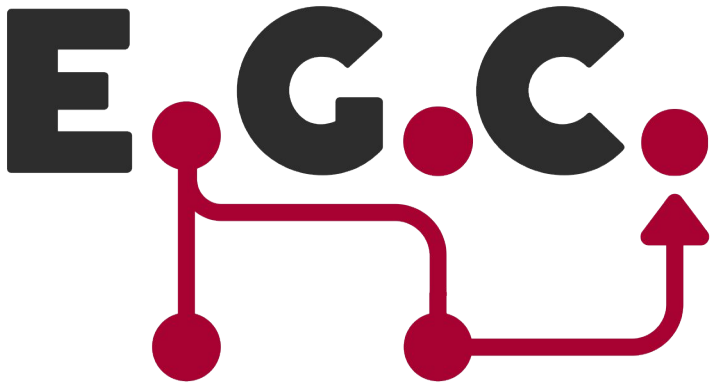


Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración



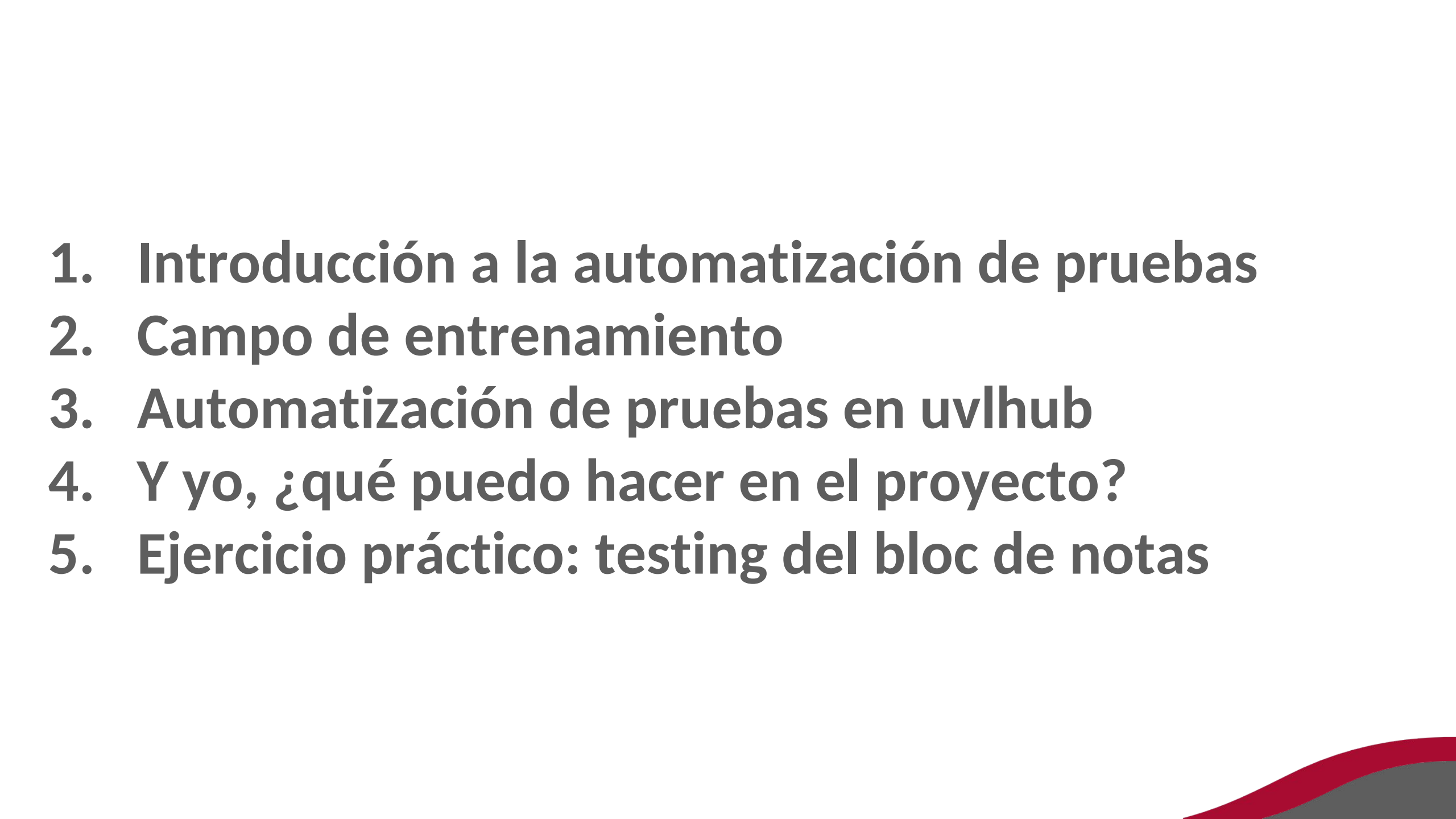


Escuela Técnica Superior de Ingeniería Informática

Práctica 3
Automatización de pruebas



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

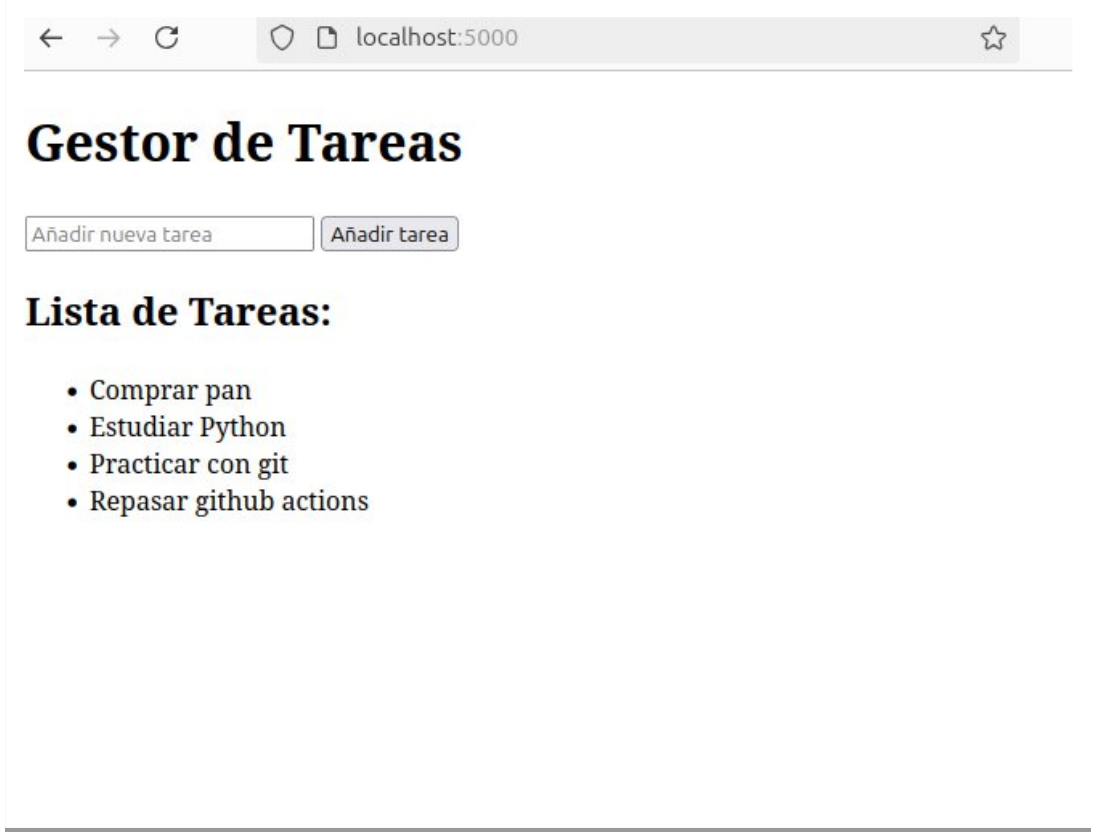


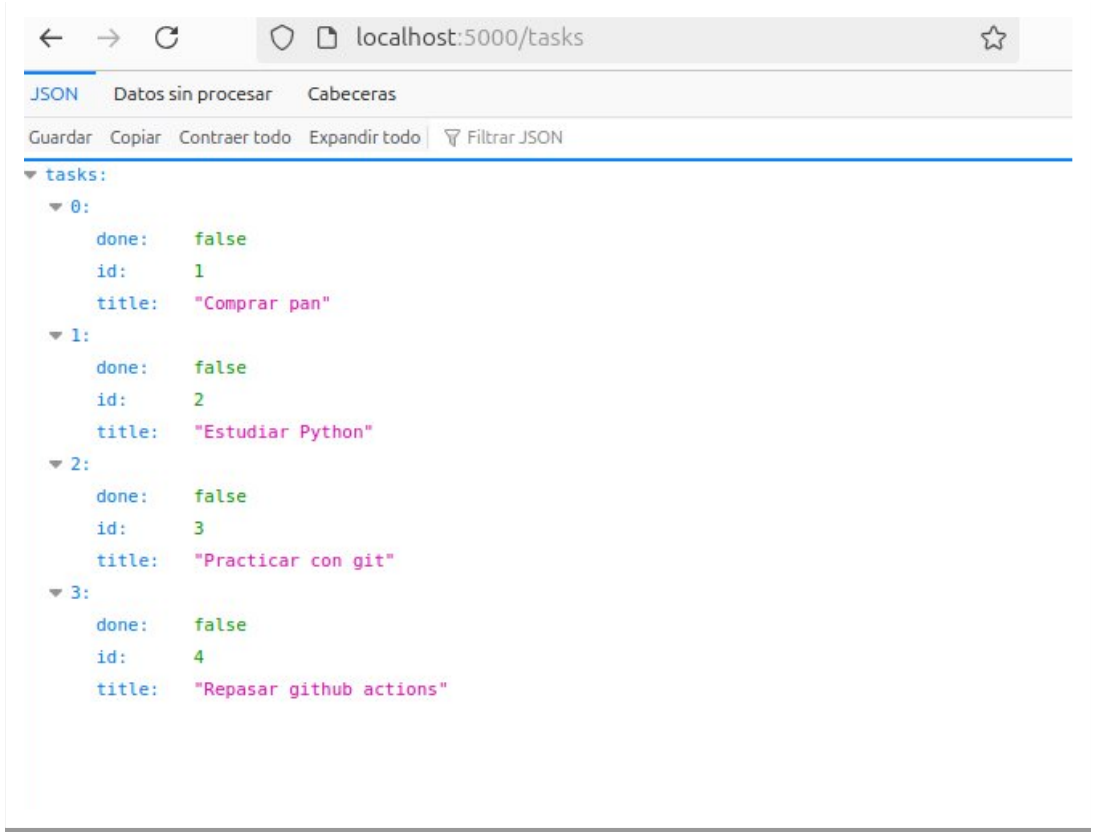
1. Las pruebas automatizadas **son más rápidas** que las manuales
2. Garantizan que se ejecuten **de la misma manera** siempre
3. Permiten probar más escenarios **de forma exhaustiva**
4. Detectan errores causados por cambios en el código (**regresión**)
5. **Reducen costos** al identificar errores **temprano**
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/
│   ├── __init__.py
│   ├── app.py
│   ├── models.py
│   ├── routes.py
│   └── templates/
│       └── tasks.html
│
├── tests/
│   ├── conftest.py
│   ├── test_unit.py
│   ├── test_integration.py
│   └── test_interface.py
│
└── locustfile.py
```

2. Campo de entrenamiento

Pruebas unitarias y de integración con pytest



pytest

```
collected 9 items

tests/test_integration.py::test_get_tasks_endpoint_returns_existing_tasks PASSED                [ 11%]
tests/test_integration.py::test_create_task_endpoint_returns_201_and_json PASSED                [ 22%]
tests/test_integration.py::test_create_task_without_title_returns_400_error PASSED              [ 33%]
tests/test_integration.py::test_add_task_html_redirects_and_renders_new_task PASSED             [ 44%]
tests/test_integration.py::test_create_then_retrieve_task_from_api PASSED                       [ 55%]
tests/test_unit.py::test_get_all_tasks_returns_list_of_dicts PASSED                             [ 66%]
tests/test_unit.py::test_create_task_adds_new_item_and_increments_length PASSED                 [ 77%]
tests/test_unit.py::test_create_task_increments_id_sequentially PASSED                          [ 88%]
tests/test_unit.py::test_create_task_raises_value_error_if_title_missing PASSED                 [100%]

============================================================== 9 passed in 0.05s ===============================================================
```

2. Campo de entrenamiento

Pruebas de cobertura con pytest-cov

```
Name                 Stmts   Miss  Cover
----------------------------------------
app/__init__.py          1      0   100%
app/app.py               6      0   100%
app/models.py            9      0   100%
app/routes.py           26      2    92%
----------------------------------------
TOTAL                   42      2    95%
```

Coverage report: 95%

Files | Functions | Classes

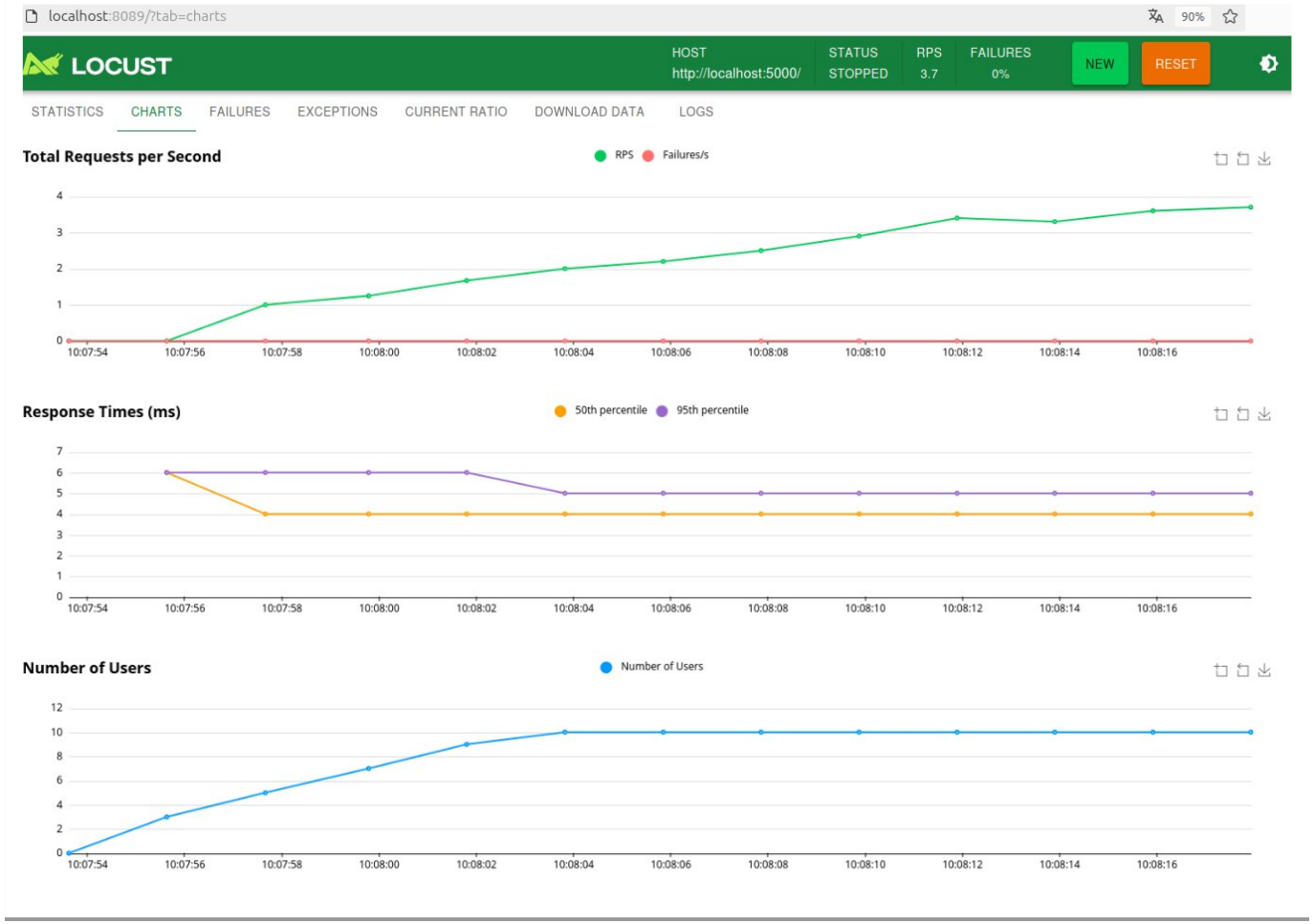
coverage.py v7.10.7, created at 2025-10-07 08:50 +0000

File ▲	*function*	*statements*	*missing*	*excluded*	*coverage*
app/__init__.py	*(no function)*	1	0	0	100%
app/app.py	create_app	3	0	0	100%
app/app.py	*(no function)*	3	0	0	100%
app/models.py	get_all_tasks	1	0	0	100%
app/models.py	create_task	5	0	0	100%
app/models.py	*(no function)*	3	0	0	100%
app/routes.py	task_list	1	0	0	100%
app/routes.py	get_tasks	1	0	0	100%
app/routes.py	add_task_html	6	2	0	67%
app/routes.py	create_task_api	7	0	0	100%
app/routes.py	*(no function)*	11	0	0	100%
Total		**42**	**2**	**0**	**95%**

coverage.py v7.10.7, created at 2025-10-07 08:50 +0000

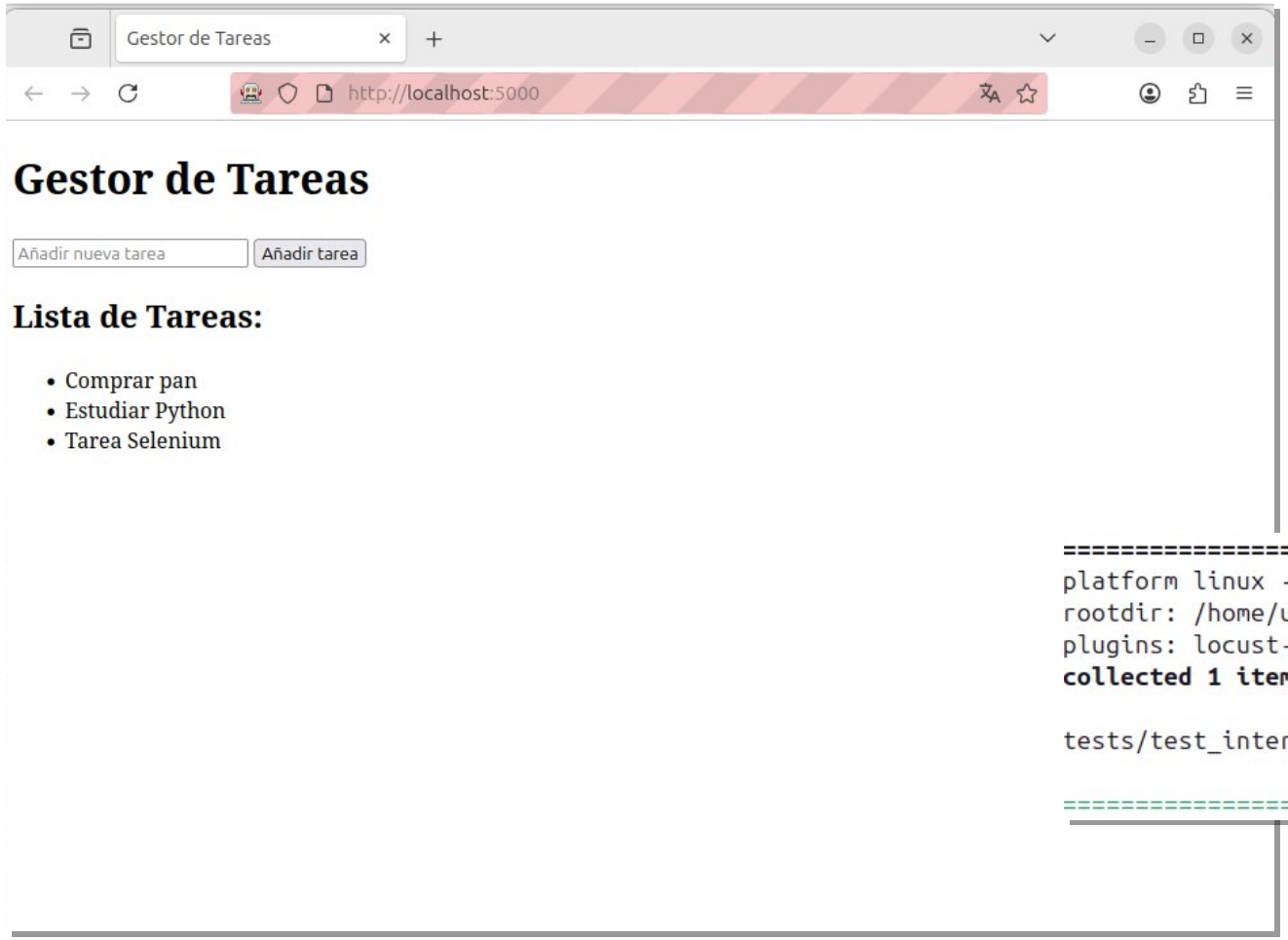
2. Campo de entrenamiento

Pruebas de carga con Locust



2. Campo de entrenamiento

Pruebas de interfaz con Selenium




```
========================================= test session starts ==========================
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/uvlhub/test
plugins: locust-2.41.5, cov-7.0.0
collected 1 item

tests/test_interface.py .

========================================= 1 passed in 17.79s ===========================
```

2. Campo de entrenamiento





https://egc.us.es/cursos/egc-20262027/documento/automatizacion-de-pruebas-software-en-una-aplicacion-flask

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

Rosemary CLI / Testing / Running tests

Running tests

`rosemary test` runs the suite at the granularity of the testing pyram
with flags; the command translates them into a `pytest -m` expressi
part of `app/features/`.

TABLE OF CONTENTS

1 Running everything by default
2 Running one feature
3 Choosing levels
4 Filtering by expression
5 The –load flag
6 What the command actually runs

Running everything by default

```
rosemary test
```

With no arguments this runs the four infrastructure-light levels — `unit`, `repository`, `service` and `integration` — across every feature in `app/features/`. Browser and load tests are left out, because they need a Selenium Grid and a Locust run respectively.

The command echoes what it is about to do, so you can always see the resolved selection:

```
Running unit or repository or service or integration tests against /workspace/app/features...
```

The target carries the `$WORKING_DIR` prefix inside a container. Outside Docker the same line reads

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
  a --html

Test coverage of all modules

To run coverage analysis for all modules within the `app/modules` directory and generate an HT
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 / Load tests

Load tests

TABLE OF CONTENTS

1 Introduction to Locust
  a Key Features:
  b 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.

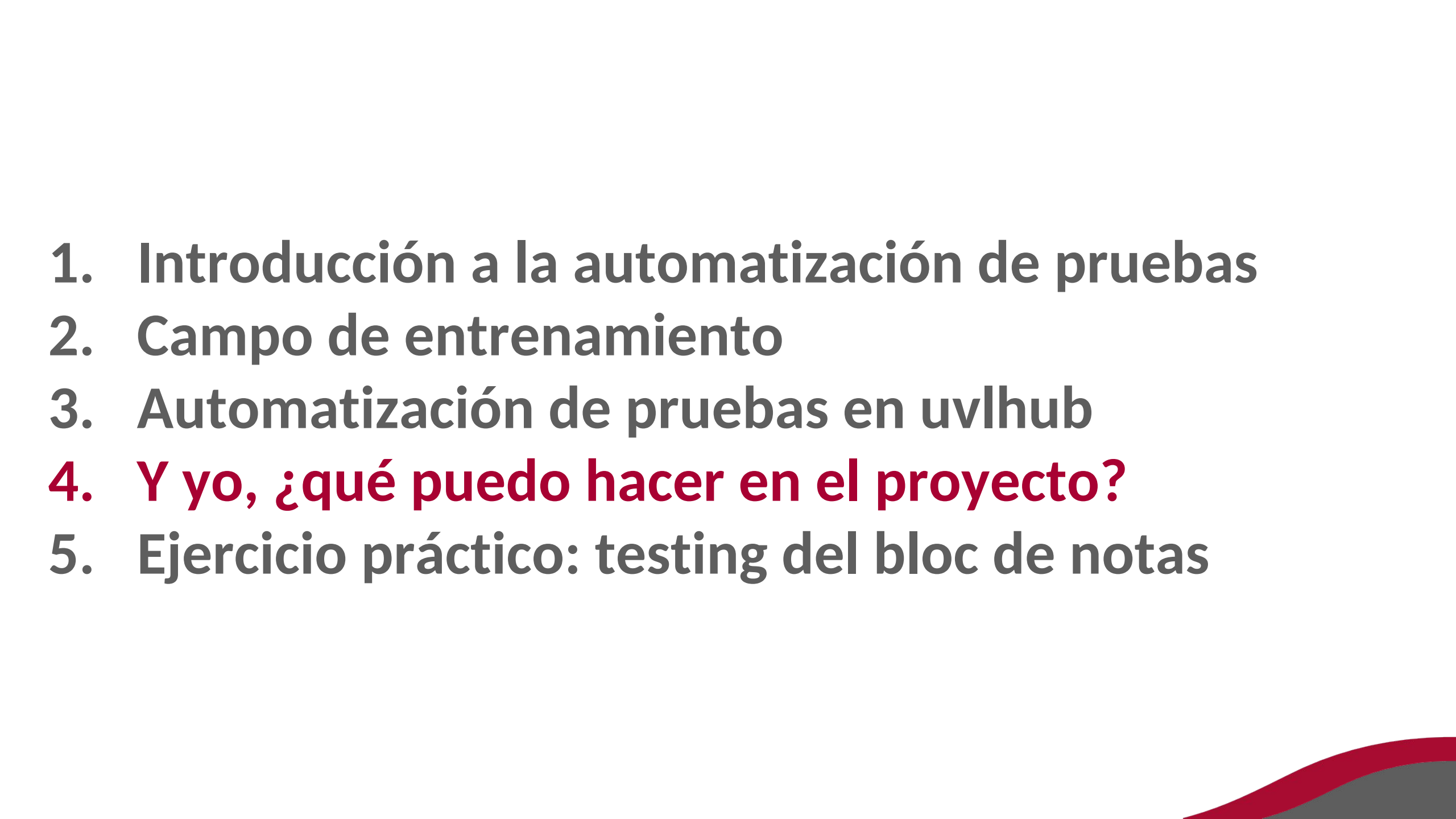
Rosemary CLI / Testing / GUI tests

GUI tests

TABLE OF CONTENTS

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

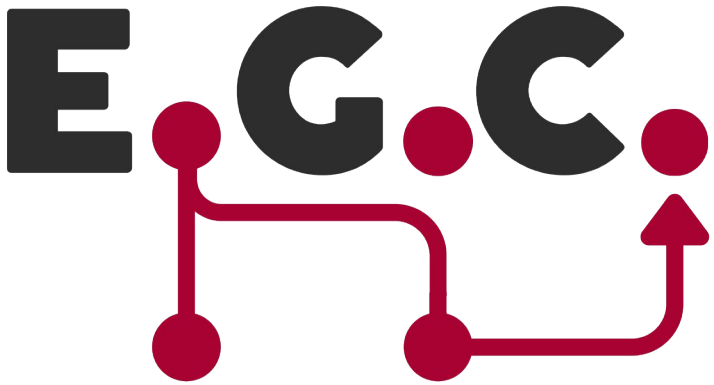
5. Ejercicio práctico: testing del bloc de notas



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

Universidad de Sevilla 1505

Escuela Técnica Superior de Ingeniería Informática

¡Gracias!

"Si pds l3r 3st0, tns n bu3n ftro en prb4s s0ftwr."

- Anónimo