

Laboratorio de seguridad: guía y retos

Este material tiene cuatro fallos deliberados. Utiliza datos ficticios y escucha únicamente en 127.0.0.1. No lo publiques en una VM, un servicio cloud o una red compartida. Solo se revisa esta copia local.

Abrir y arrancar

1. Instala o utiliza el Python 3 del aula. En una terminal ejecuta `python --version` (en algunos Windows, `py --version`).
2. Extrae el ZIP y abre esa carpeta en VS Code. En Terminal → Nueva terminal comprueba que ves `app.py` y `test_app.py` con `dir` (Windows) o `ls`.
3. Ejecuta `python app.py`. Visita `http://127.0.0.1:8090`. La respuesta JSON muestra las rutas. Deja la terminal abierta. Ctrl+C detiene el servidor.
4. Abre otra terminal en la misma carpeta para ejecutar `python -m unittest -v`. Al principio hay 7 pruebas: 3 pasan y 4 fallan deliberadamente. No cambies las expectativas para que queden verdes: corrige la aplicación.

Identidad simplificada

La cabecera `X-Demo-User` representa una identidad YA validada por un sistema de autenticación que no forma parte del laboratorio. Aquí se puede escribir `ana`, `bruno` o `admin` para probar reglas. Poder escribir esa cabecera no es autenticarse: nunca se utilizaría así en producción. No se evalúa construir un login ni JWT.

En Windows utiliza `curl.exe`; en Linux/macOS, `curl`:

```
curl.exe -i -H "X-Demo-User: ana" http://127.0.0.1:8090/orders/1
curl.exe -i -H "X-Demo-User: ana" http://127.0.0.1:8090/orders/2
curl.exe -i http://127.0.0.1:8090/config
curl.exe -i http://127.0.0.1:8090/demo-error
```

La primera consulta es válida. La segunda debería responder 403 porque el pedido pertenece a Bruno, pero la versión inicial responde 200: ese es el primer fallo. Tras editar `app.py`, detén el servidor y arráncalo de nuevo antes de comprobar HTTP; las pruebas importan la versión nueva en cada ejecución.

Matriz esperada

Acción	Anónimo	Cliente	Admin
Consultar un pedido	No, 401	Solo el propio; ajeno 403	Sí
Buscar productos	Sí	Sí	Sí
Consultar configuración pública	Solo nombre del sitio	Solo nombre del sitio	Solo nombre del sitio
Recibir error público	Mensaje genérico	Mensaje genérico	Mensaje genérico

Evidencias

Una tabla por actividad: función, fallo, prueba inicial, corrección, prueba posterior y límite. Al terminar, las correcciones deben funcionar y la tabla debe permitir explicar qué se ha comprobado. No se necesita el CRUD de Servidor, GitHub Actions ni un servidor externo.

Pistas graduadas

1. Pedido ajeno: caso guiado

En `get_order`, después de comprobar que el pedido existe y antes de devolver 200, compara `user` con `order["owner"]`. Si no coincide y tampoco es `admin`, devuelve 403 y un mensaje de acceso denegado. No elimines las comprobaciones de identidad ni de existencia. Ejecuta: `python -m unittest test_app.ReglasDeLaboratorio.test_pedido ajeno_rechazado -v` y luego toda la suite. Deben conservarse pedido propio y acceso `admin`.

2. Búsqueda: separar código y dato

La consulta actual concatena el texto que recibe. SQLite admite un marcador `?` y una tupla de parámetros: `db.execute("SELECT name FROM products WHERE name = ?", (query,))`. Sustituye la construcción y ejecución de la consulta, no la aserción de la prueba. La coma de la tupla es necesaria. Una entrada que parece SQL debe buscarse como texto literal.

3. Configuración pública: elegir campos

`public_config` no debería devolver toda `CONFIG`. Construye un diccionario nuevo con únicamente `site`. La constante `demo_secret` es un marcador ficticio para observar una exposición: no es una credencial. En un sistema real, además habría que gestionar los secretos fuera del repositorio y rotar cualquier credencial expuesta. Ocultar una respuesta no resuelve por sí solo esos otros aspectos.

4. Error público: limitar detalle

`error_response` debe devolver estado 500 y el cuerpo `{"error": "Error interno"}`. Un sistema real necesitaría diagnóstico interno adecuado y sin información sensible, separado de su respuesta pública. El laboratorio solo comprueba la respuesta, no implementa esa infraestructura.

Fichas de discusión

- Contraseñas: guardar texto plano o un hash rápido sin un mecanismo específico de almacenamiento de contraseñas no es una solución adecuada. Elige una biblioteca mantenida y parámetros apropiados; en esta actividad no se construye un login casero.
- Secretos: borrar una clave del archivo actual no la elimina de versiones anteriores ni invalida copias. Identifica exposición, revocación/rotación y configuración del entorno.
- Dependencias: este laboratorio usa biblioteca estándar; no hace falta instalar paquetes. En una aplicación con paquetes, el inventario, sus versiones y avisos son el punto de partida. Un aviso debe investigarse según uso y alcance; no equivale por sí solo a una explotación confirmada.
- Logs: diagnosticar no requiere copiar contraseñas, tokens ni datos personales completos. Explica quién accede al registro y qué información necesita.

Siete pruebas verdes demuestran estos casos, no seguridad absoluta. Las cuatro correcciones se reparten entre las sesiones 2, 3 y 4 y se revisan en la 5.