# Remote Executor v16-gfx — Documentação

Um micro‑"laboratório remoto" em NiceGUI 2.11 para editar, executar e observar scripts Python com stdout/stderr em tempo real, geração de artefactos (incluindo gráficos e animações) e um painel de debug que conta a história toda. Compacto, resiliente e pragmático.

## Visão geral

Remote Executor é uma aplicação NiceGUI que expõe uma página única (/executor) com:

* um editor CodeMirror para escrever/colar Python,
* botões para Executar/Parar,
* streaming do output em tempo real,
* lista de ficheiros gerados + botões de download,
* um painel DEBUG com registo detalhado dos eventos.

É pensada para ambientes headless (sem ecrã) e inclui um runner que cuida de Matplotlib: plt.show() guarda PNG, plt.savefig(...) vai para o OUTPUT\_DIR, e animation.Animation.save(...) redireciona .gif/.mp4 para o sítio certo. No fecho de cada execução, um coletor varre o /tmp e copia artefactos gráficos para o destino.

## Principais funcionalidades

* Sincronização robusta do editor: no clique em Executar, o browser envia o código por POST /\_\_code\_\_. O servidor só arranca quando o texto chega.
* Streaming fiável dos print(): por defeito corre sob PTY (line‑buffering). Se o PTY falhar, faz fallback para pipes.
* Geração de gráficos/animações:
  + MPLBACKEND=Agg automático;
  + monkeypatch de plt.show, plt.savefig e animation.Animation.save para guardar no OUTPUT\_DIR quando o caminho é relativo;
  + coleta pós‑execução dos artefactos novos (extensões: .png, .jpg, .jpeg, .svg, .pdf, .html, .gif, .mp4, .webm, .avi, .mov).
* Multi‑sessão por cliente: estado isolado por ui.context.client.id (processo, timers, buffers, PTY, etc.).
* Segurança simples por token (EXECUTOR\_TOKEN) via query string.
* Shell opcional: caixa para comandos básicos (pip install ..., etc.), limitada por timeout.
* Limites sensatos: timeouts de execução, truncagem do log a ~200 KB, limpeza de temporários.

## Arquitetura (em 10 linhas)

Browser (CodeMirror) ──► POST /\_\_code\_\_ ──► CODE\_STORE[cid]

◄─ polling curto ── servidor arranca runner

runner.py ──► importa matplotlib (Agg), monkeypatch show/save/save\_anim

└─► runpy.run\_path(user\_script.py)

stdout/stderr ──► PTY (os.read + select) ──► textarea (streaming)

/tmp/remote-executor ──► coletor ► OUTPUT\_DIR (artefactos)

UI: /executor (editor + debug + lista + downloads)

API extra: GET /download/{ficheiro}

## Requisitos & Instalação

* Python ≥ 3.10
* NiceGUI 2.11
* Dependências para gráficos/animações:
  + matplotlib (Agg backend)
  + pillow (para PillowWriter em .gif)
  + (opcional) ffmpeg no PATH para .mp4/.webm via writers do Matplotlib

Instalação típica:

pip install nicegui matplotlib pillow

# (opcional) sudo apt-get install ffmpeg

## Configuração

Constantes no topo do ficheiro:

* OUTPUT\_DIR: pasta onde ficam os artefactos (downloads)
* TMP\_DIR: onde o script é gravado e executado
* PORT: porta HTTP (ex.: 2020)
* ALLOW\_SHELL: ativa/desativa a aba Shell
* PY\_TIMEOUT, SH\_TIMEOUT: tempo máximo em segundos
* STREAM\_INTERVAL: frequência de flush do streaming
* MAX\_OUTPUT\_CHARS: limite do buffer de saída

Variável de ambiente (opcional):

* EXECUTOR\_TOKEN: se definido, o acesso exige ?token=...

## Execução

export EXECUTOR\_TOKEN="segredo" # opcional

python3 remote\_executor.py

# abre: http://localhost:2020/executor

# ou: http://<ip>:2020/executor?token=segredo (se usares token)

## Interface /executor

### Editor

* CodeMirror com syntax highlight;
* botão Tema editor alterna Dracula/Eclipse.

### Controlo

* Executar Python: dispara POST /\_\_code\_\_ e arranque seguro.
* Parar Python: envia terminate() e fecha com grace period.

### Saída (streaming)

* Mostra stdout e stderr em tempo real; truncagem aos últimos ~200 KB.

### DEBUG

* Timeline de eventos: versão Python/SO, testes de escrita, Wrote script, Spawned with PTY, Timer ON, process exited, copied graphics: ..., etc.

### Ficheiros & Downloads

* Lista os conteúdos de OUTPUT\_DIR com botão Download.

### Shell (opcional)

* Executa um comando simples no OUTPUT\_DIR (ex.: pip install pillow), respeitando SH\_TIMEOUT.

## Como funciona por baixo do capot

### 1) Sincronização do código

No clique em Executar, o browser faz fetch('/\_\_code\_\_', {code, cid}). O servidor aguarda até o CODE\_STORE[cid] receber o texto (polling curto) e só então escreve o .py em TMP\_DIR.

### 2) Runner para gráficos

O runner:

* força MPLBACKEND=Agg (headless);
* redefine:
  + plt.show() → além de “mostrar”, guarda figure-<timestamp>.png em OUTPUT\_DIR;
  + plt.savefig(path) → se path for relativo, reescreve para OUTPUT\_DIR/path;
  + animation.Animation.save(path) → idem, redireciona para OUTPUT\_DIR/path.

### 3) Coleta pós‑execução

Ao terminar, copia de TMP\_DIR para OUTPUT\_DIR os artefactos criados durante a execução (.png, .jpg, .jpeg, .svg, .pdf, .html, .gif, .mp4, .webm, .avi, .mov). Evita conflitos acrescentando -copyN ao nome.

### 4) Streaming via PTY

Por defeito cria um PTY (pseudo‑terminal) para forçar line‑buffering. Se não for possível, usa pipes com bufsize=1. Um ui.timer faz select + os.read (PTY) ou drena queues (pipes) e atualiza o textarea.

### 5) Multi‑sessão

O estado de cada cliente (processo, timers, PTY, buffers) vive num dicionário indexado por ui.context.client.id. Assim, sessões distintas não se atropelam.

## Exemplos úteis

### A) Script simples com print() e PNG

# -\*- coding: utf-8 -\*-

import os, matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

print('INICIO')

plt.plot([1, 2, 3], [1, 4, 9])

plt.title('Demo')

plt.show() # guardado automaticamente como figure-<timestamp>.png

out = os.environ.get('EXECUTOR\_OUTPUT\_DIR', '.')

open(os.path.join(out, 'ok.txt'), 'w').write('feito\n')

print('FIM')

### B) Animação .gif (PillowWriter)

# -\*- coding: utf-8 -\*-

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.animation as animation

from matplotlib.animation import PillowWriter

x = np.linspace(-5, 5, 200)

y = np.linspace(-5, 5, 200)

X, Y = np.meshgrid(x, y)

def onda(t):

return np.sin(2 \* np.pi \* (X \* 0.2 - t \* 0.05)) \* np.exp(-0.05 \* (X\*\*2 + Y\*\*2))

fig, ax = plt.subplots(figsize=(6, 6))

im = ax.imshow(onda(0), extent=(-5, 5, -5, 5), origin='lower', cmap='viridis', animated=True)

def animate(t):

im.set\_array(onda(t))

return [im]

ani = animation.FuncAnimation(fig, animate, frames=100, interval=50, blit=True)

ani.save('tecido\_espaco\_tempo.gif', writer=PillowWriter(fps=20)) # vai para OUTPUT\_DIR

print('GIF gerado!')

Dica: se estiveres a gerar .mp4 e não surgir, verifica se ffmpeg está instalado e no PATH.

## Endpoints

* GET /executor → UI principal
* POST /\_\_code\_\_ → recebe o código do editor {cid, code} (JSON)
* GET /download/{filename} → entrega ficheiros do OUTPUT\_DIR

## Segurança

* Token simples: define EXECUTOR\_TOKEN e acede com ?token=<valor>.
* Boas práticas:
  + corre atrás de reverse proxy (https),
  + restringe IPs de origem (firewall),
  + mantém ALLOW\_SHELL=False se não precisares.

Roadmap: autenticação por sessão