NiceGUI: interfaz web para tu código Python

[Juan Carlos Sedano Salas] · Pythonistas GDL · Abril 24, 2026

Tienes código Python que hace algo útil...

¿cómo lo integras en una app?

Opción 1: GUI de escritorio

tkinter · **PyQt** · **wxPython** — potentes, pero...

- ❌ Curva de aprendizaje alta
- ❌ Distribución complicada (instaladores por SO)
- ❌ Estética desactualizada

Opción 2: construir una app web

Flask / FastAPI para el backend + HTML + CSS + JavaScript para el frontend

```
Python ↔ HTML/CSS/JS ↔ Navegador
```

- Aprender HTML para estructurar
- Aprender CSS para estilos
- Aprender JavaScript para interactividad
- Conectar frontend con backend

> Mucho trabajo solo para mostrar un botón.

Opción 3: Streamlit y Gradio

Fáciles de usar, ideales para prototipos rápidos.

	Streamlit	Gradio
Enfoque	Datos y dashboards	Demos de ML/IA
Instalación	`pip install streamlit`	`pip install gradio`
Popular en	Ciencia de datos	Hugging Face

Pero tienen sus limitaciones...

Flexible como una app web real,

simple como Streamlit y Gradio

todo en Python.

¿Qué es NiceGUI?

Framework de UI en Python que se muestra en el navegador (o en una ventana nativa).

- `pip install nicegui`
- Todo el código en Python — sin HTML, CSS ni JS
- Recarga automática al guardar
- Disponible en PyPI / conda-forge / Docker

Instalación y Hello World

```
from nicegui import ui

ui.label('¡Hola NiceGUI!')
ui.button('Presióname', on_click=lambda: ui.notify('¡Hola!'))

ui.run()
```

Guardan en `main.py`, ejecutan `python main.py`, abren `localhost:8080`.

Sin HTML. Sin CSS. Sin JavaScript.

Elementos básicos I

Elemento	Uso
`ui.label('texto')`	Texto estático o dinámico
`ui.input('placeholder')`	Caja de texto de una línea
`ui.textarea('placeholder')`	Texto de varias líneas

Cada elemento devuelve un objeto Python:

- `.value` — leer o escribir el valor actual
- `.set_value()`, `.set_visibility()`, etc.

Elementos básicos II

Elemento	Descripción
`ui.checkbox('Acepto')`	Casilla de verificación
`ui.switch('Modo oscuro')`	Toggle on/off
`ui.slider(min=0, max=100)`	Control deslizante
`ui.select(['a', 'b'])`	Menú desplegable

Todos aceptan `on_change` para reaccionar a cambios. Todo en Python.

Layout: organizar la pantalla

```
with ui.row():
    ui.button('Uno')
    ui.button('Dos')

with ui.card():
    ui.label('Título')
    ui.input('Escribe aquí')

with ui.grid(columns=3):
    for i in range(6):
        ui.label(f'Item {i}')
```

`ui.row()` · `ui.column()` · `ui.card()` · `ui.grid()`

Estilos con Tailwind CSS

NiceGUI incluye Tailwind CSS. Agrega clases con `.classes()`:

```
ui.label('Texto importante').classes('text-2xl font-bold text-blue-500')
ui.button('Acción').classes('mt-4 w-full')
```

Las clases son descriptivas:

- `text-2xl` → texto grande
- `font-bold` → negrita
- `text-blue-500` → azul

Sin escribir CSS manual.

Eventos y callbacks

```
def al_hacer_click():
    ui.notify('¡Funcionó!')

ui.button('Haz clic', on_click=al_hacer_click)
```

> El servidor maneja la lógica, el navegador muestra el resultado — vía **WebSocket**.

Data binding

```
class Config:
    nombre = ''

config = Config()

ui.input('Nombre').bind_value(config, 'nombre')
ui.label().bind_text_from(config, 'nombre')
```

objeto Python ↔️ elemento UI

Sincronización automática y bidireccional. Sin `addEventListener`. Sin polling.

`@ui.refreshable`

```python
@ui.refreshable
def lista_de_items():
    for item in items:
        ui.label(item)

lista_de_items()

def agregar():
    items.append('Nuevo item')
    lista_de_items.refresh()

ui.button('Agregar', on_click=agregar)
```

Reconstruye solo esa sección — **sin recargar toda la página**.

Páginas múltiples

```
@ui.page('/')
def index():
    ui.label('Inicio')
    ui.link('Ir al dashboard', '/dashboard')

@ui.page('/dashboard')
def dashboard():
    ui.label('Dashboard')

ui.run()
```

Rutas HTTP reales. Cada visita crea un contexto de cliente **independiente**.

Persistencia: storage

Storage	Alcance	Persistencia
`app.storage.user`	Por usuario	Persiste al recargar
`app.storage.general`	Todos los usuarios	Indefinida
`app.storage.tab`	Por pestaña	Se pierde al cerrar

```
app.storage.user['operador'] = nombre_input.value
```

Diccionarios que se persisten automáticamente en disco.

Modo nativo: ventana de escritorio

```
ui.run(native=True)  # ← solo cambia esto
```

La misma app abre en una ventana de escritorio usando **pywebview**.
El código Python es exactamente el mismo.

```
pip install pywebview
```

Lo que hay debajo

Capa	Tecnología
Tu código	Python
Servidor	FastAPI + Starlette + Uvicorn
Comunicación	WebSockets (tiempo real)
Frontend	Vue 3 + Quasar + Tailwind CSS

Tú escribes Python; NiceGUI traduce todo al frontend automáticamente.

Característica	NiceGUI	Streamlit	Gradio
Licencia	MIT, gratis	Apache 2.0, gratis[1]	Apache 2.0, gratis
Manejo de estado	Python puro	Re-ejecuta el script	Controlado por lib
Flexibilidad UI	Alta	Media	Baja–Media
Curva aprendizaje	Media	Baja	Muy baja
Ideal para	Apps, IoT, escritorio	Dashboards de datos	Demos ML/IA

[1] *Community Cloud gratis solo para apps públicas*

¿Cuándo SÍ usar NiceGUI?

- ✅ Herramientas internas de empresa o equipo
- ✅ Dashboards de monitoreo y control
- ✅ Apps de robótica, IoT, laboratorio
- ✅ Prototipos rápidos de aplicaciones completas
- ✅ Apps de escritorio con Python puro

¿Cuándo NO usar NiceGUI?

- ❌ Sitios públicos de alto tráfico o SEO crítico
- ❌ Cuando el equipo ya domina React/Vue/Angular
- ❌ Demos simples de modelos ML (Gradio es más directo)
- ❌ Aplicaciones que requieren control total del HTML/DOM

Cosas a tomar en cuenta

- Cada cliente tiene estado propio en el servidor → planear memoria con muchos usuarios
- `reload=True` solo para desarrollo, desactivar en producción
- Para escalar: Docker + nginx como proxy inverso
- La comunidad es activa: GitHub, Discord, ejemplos oficiales

¿Preguntas?

🌐 nicegui.io

💻 github.com/zauberzeug/nicegui

📦 `pip install nicegui`