

GUI LabVIEW WYSIWYG o Python?

Category: LabVIEW Talk

scritto da Nicola Bavarone | Aprile 12, 2026



Introduzione

Come team di **sviluppatori LabVIEW**, siamo abituati a lavorare con strumenti che privilegiano la rapidità, la visualità e l'efficienza:

l'interfaccia **WYSIWYG** di LabVIEW è un punto di forza, permette di costruire GUI desktop in tempi record e senza scrivere codice per il design.

Ma il mondo evolve: Python è sempre più usato anche in automazione e test, il cloud e le web app portano nuove esigenze, ed è naturale chiedersi se convenga esplorare Python e i suoi framework anche per le interfacce grafiche.

Qui entriamo da sviluppatori e raccontiamo le nostre esperienze dirette tra LabVIEW, Python/Kivy e Flask (web), evidenziando pro, contro e scenari applicativi reali.

Il valore unico di LabVIEW

- **Interfaccia WYSIWYG:** drag&drop, sviluppo visivo, nessun codice per il design.
- **Rapidità di prototipazione** su desktop, perfetta per test automatici, acquisizione dati e applicazioni industriali.
- **Stretta integrazione con hardware NI** e con I/O real time, code FIFO e multithreading senza pensieri.
- **Debugger visuale** e strumenti evoluti per chi lavora in laboratorio o produzione.

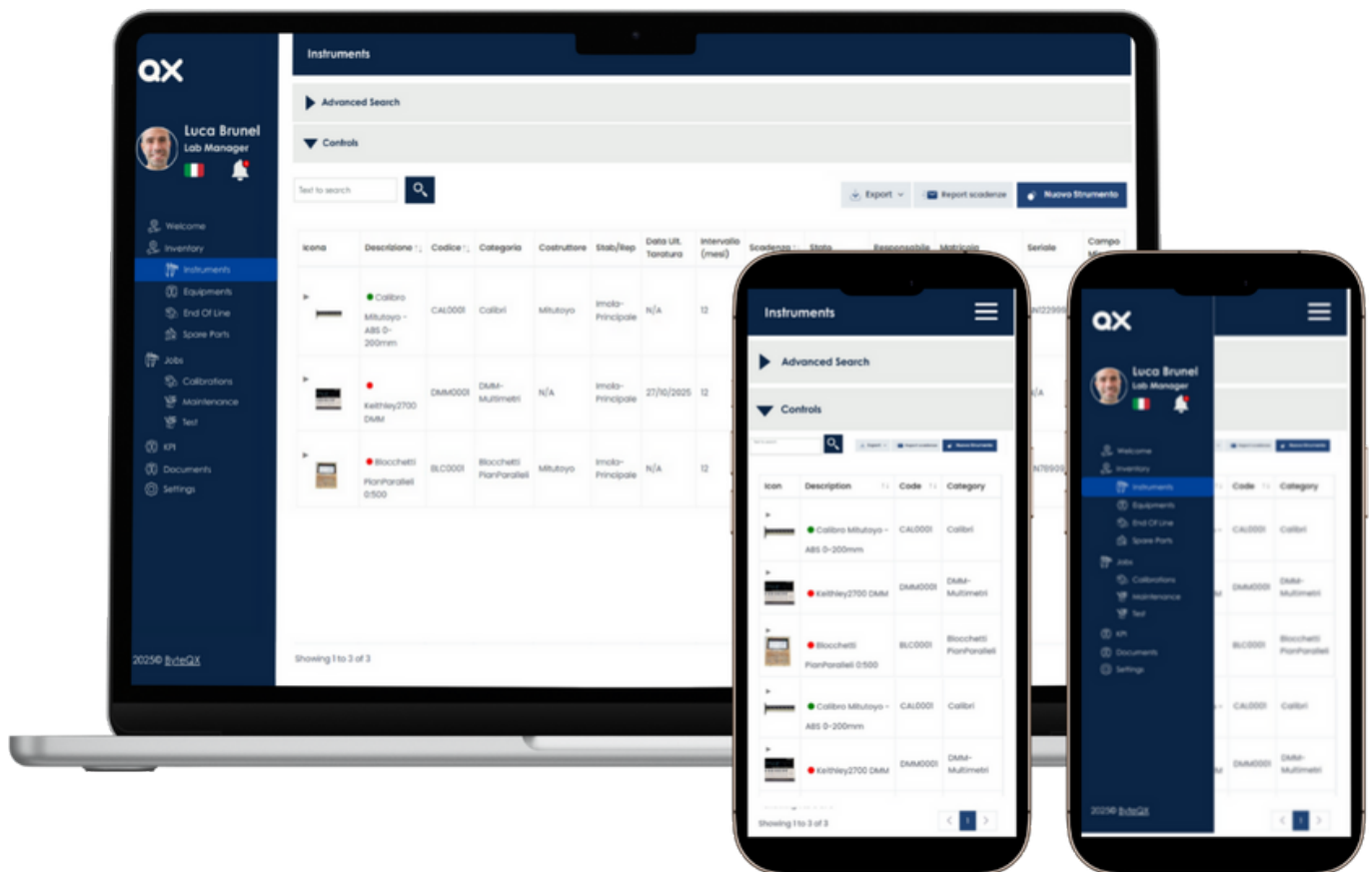


Perché rimane imbattibile:

- Dove conta il tempo di sviluppo, la robustezza, la facilità di modifica sul campo e la connessione con dispositivi industriali, LabVIEW è ancora il riferimento.

KIVY: Python per GUI desktop e anche mobile

- **Framework open source:** permette di creare App Python desktop e mobile (Windows/Linux/Mac/Android/iOS) con look moderno e responsive.
- **Tutto è codice:** la GUI si definisce via script Python e file .kv, senza editor visuale.
- **Multipiattaforma reale:** lo stesso progetto funziona su PC e tablet/smartphone.
- **Ideale per:** applicazioni che escono dal mondo Windows, necessità di touch UI, utilizzo su device portatili aziendali.



Esempio codice:

```
pythonfrom kivy.app import App
from kivy.uix.button import Button
```

```
class TestApp(App):
    def build(self):
        return Button(text='Hello Kivy!')
TestApp().run()
```

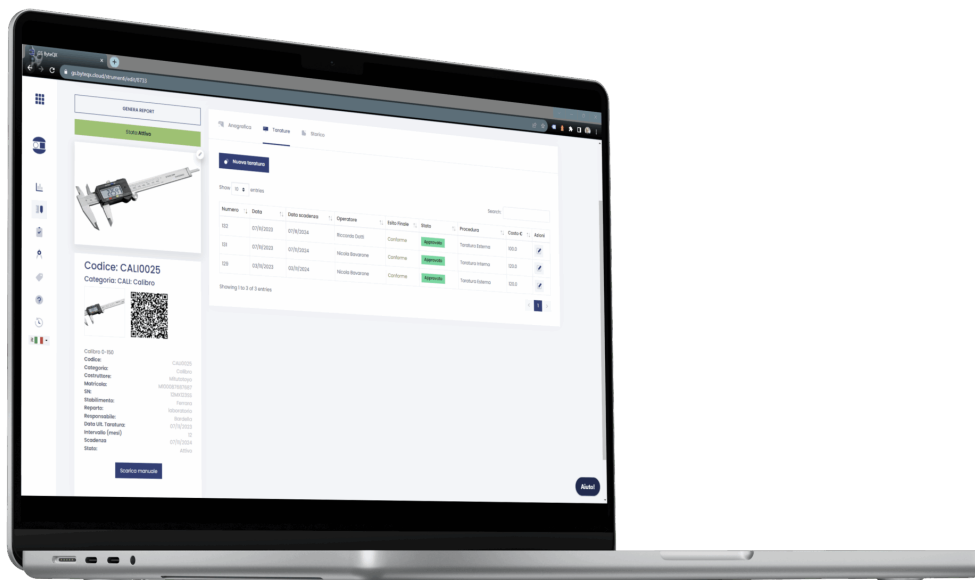
Con quattro righe, hai una GUI funzionante e portatile.

Pro e contro:

- (+) Portabilità, UX moderna, Python nativo, ottima gestione async
- (-) Tutto va scriptato, meno componenti pronti rispetto a LabVIEW.
- Competenze richieste diverse: si parte da concetti di GUI totalmente differenti rispetto a quello LabVIEW.

Flask/Django + Bootstrap: il mondo web entra nei test e misura

- **Flask/Django** sono framework Python per creare webapp: puoi realizzare GUI accessibili da browser ovunque, anche localmente.
- **Front-end/Back-end separati:** il front-end (HTML/CSS/JS/Bootstrap/React) può essere sviluppato in parallelo e su misura, il backend Python o LabVIEW può gestire la logica e i dati.
- **Web = Accessibilità + Scalabilità:** si lavora su qualsiasi device, update immediati, interfacce sempre attuali.



Esempio codice Flask:

```
python from flask import Flask, render_template
```

```
app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')
```

Poi in index.html usi i temi Bootstrap per UI responsive.

Vantaggi operativi:

- Team multidisciplinari (Pythonisti, frontend developer) possono lavorare in parallelo.
- Facile portare la supervisione su mobile o device remoti.
- Scambio dati con LabVIEW facile via REST API, Websocket o TCP.

Il design avanzato con i framework web: Bootstrap, React, Vue.js e oltre

Negli ultimi anni, la **capacità di design delle interfacce utente** ha fatto un salto di qualità grazie alla diffusione di framework frontend come **Bootstrap, React, Vue.js** (ma anche Angular e tanti altri).

Queste tecnologie permettono di creare applicazioni con:

- **Grafica moderna e avanzata:** animazioni, layout dinamici, palette cromatiche evolute e micro-interazioni che migliorano la user experience.
- **Interfacce responsive:** perfette su desktop, tablet e smartphone, adattandosi automaticamente a schermi e periferiche diverse.
- **Temi applicativi pronti:** esistono moltissimi template, dashboard professionali e componenti riutilizzabili che accelerano lo sviluppo.
- **Personalizzazione totale:** ogni dettaglio dell'aspetto può essere raffinato da designer e sviluppatori, ottenendo GUI di livello enterprise.
- **Adozione crescente anche per installazioni locali:** oggi molte interfacce web moderne non sono più solo "in cloud", ma vengono distribuite e utilizzate anche come applicazioni locali interne all'azienda o all'impianto (grazie a server locali o persino con packaging tramite Electron ecc.).

Questa evoluzione rende ormai la **separazione Frontend/Backend** non solo vantaggiosa dal punto di vista architettuale e di team, ma anche estremamente potente dal punto di vista del risultato estetico e funzionale, aprendo la porta a user experience di altissimo livello.

Sfide:

- Servono competenze web (HTML/CSS/JS), deployment più articolato e attenzione a sicurezza/autenticazione.

Confronto tra approcci

Soluzione	Pro	Contro	Ideale per
LabVIEW	Drag&Drop, rapidità, hardware NI	Grafica limitata, non web	Test/automazione industriale, team NI
Kivy	Multipiattaforma, Python puro, UX touch	Tutto codice, meno diffusa	Dashboard mobile/desktop, Pythonisti
Flask+Web	Web, separazione netta, teamwork	Competenze web, deployment	Supervisione cloud/local, team multidisciplinari

Esperienza e consigli pratici

- **LabVIEW è il metodo più rapido per il nostro team:** WYSIWYG e strumenti evoluti rendono insuperabile il lavoro su desktop per test e automazione.
- **Kivy** ha senso se vuoi piena portabilità, funzionalità touch e magari portare la GUI su device non Windows. Richiede cambio di mentalità: si programma tutto!
- **Flask/Django/Web** è la scelta inevitabile se vuoi scalabilità, accesso remoto e collaborazione fra diversi ruoli. Perfetto in cloud, ma anche per dashboard locali su browser aziendale.

Come ottimizzare il lavoro:

La vera evoluzione è la **separazione front-end/back-end**:

- LabVIEW/Python governano il processo e I/O
- Kivy, Flask, React, Bootstrap si occupano della GUI
- Dati scambiati via TCP/IP, REST API, Websocket
- Team multidisciplinari collaborano e ottengono risultati più modulari, scalabili e aggiornabili nel tempo

Vuoi vedere esempi concreti?

Contattaci e organizza il tuo training LabVIEW e Python con uno dei nostri Istruttori.