






















-  Tutorial Completo: PyWebView e Integração com Google Sheets
 -  Objetivo
 -  Índice
 - 1. O que é PyWebView
 -  Definição
 -  Características Principais
 -  Comparação com Outras Tecnologias
 - 2. Arquitetura do PyWebView
 -  Estrutura em Camadas
 -  Fluxo de Dados
 - 3. Instalação e Configuração
 -  Instalação Básica
 -  Dependências por Sistema Operacional
 - macOS
 - Linux (Ubuntu/Debian)
 - Windows
 - 4. Conceitos Fundamentais
 -  Criação de Janelas
 - Parâmetros Principais
 -  Configurações Globais
 - 5. Criando sua Primeira Janela
 -  Exemplo Básico
 -  Com HTML Local
 - 6. Integração com Flask
 -  Servidor Flask Básico
 -  Servidor em Thread Separada
 -  Verificação de Servidor Pronto
 - 7. Comunicação Python ↔ JavaScript
 -  Expondo Funções Python para JavaScript
 - Backend Python
 - Frontend JavaScript
 -  Eventos e State
 - 8. Integração com Google Sheets
 -  Autenticação OAuth2
 -  Operações CRUD
 - Leitura de Dados
 - Escrita de Dados

- Adicionar Dados
- Deletar Linhas
-  Integração Completa PyWebView + Flask + Google Sheets
- 9. Ciclo de Vida da Aplicação
 -  Eventos Disponíveis
 -  Fluxo Completo
- 10. Boas Práticas
 -  Estrutura de Projeto Recomendada
 -  Segurança
 -  Performance
 -  Modo Debug vs Produção
- 11. Troubleshooting
 -  Problemas Comuns
 - Porta já em uso
 - Google Sheets Error 403
 - PyWebView não abre janela (Linux)
 - SSL Certificate Error
-  Recursos Adicionais
 - Links Úteis
 - Exemplos Avançados
-  Conclusão
 - Próximos Passos

Tutorial Completo: PyWebView e Integração com Google Sheets

Objetivo

Este tutorial detalha o funcionamento da biblioteca **PyWebView** e como integrá-la com **Google Sheets** para criar aplicações desktop usando tecnologias web (React, Flask, etc.).

Índice

1. O que é PyWebView
 2. Arquitetura do PyWebView
 3. Instalação e Configuração
 4. Conceitos Fundamentais
 5. Criando sua Primeira Janela
 6. Integração com Flask
 7. Comunicação Python ↔ JavaScript
 8. Integração com Google Sheets
 9. Ciclo de Vida da Aplicação
 10. Boas Práticas
 11. Troubleshooting
-

1. O que é PyWebView



Definição

PyWebView é uma biblioteca Python leve que permite criar aplicações desktop usando tecnologias web (HTML, CSS, JavaScript) com uma janela nativa do sistema operacional.



Características Principais

- **Multiplataforma:** Funciona em Windows, macOS e Linux
- **Leve:** Usa os componentes nativos do sistema (WebKit, Chromium, etc.)
- **Integração Python:** Permite chamar funções Python do JavaScript
- **Sem dependências pesadas:** Não precisa empacotar um navegador inteiro (como Electron)
- **Servidor HTTP integrado:** Suporta aplicações web completas



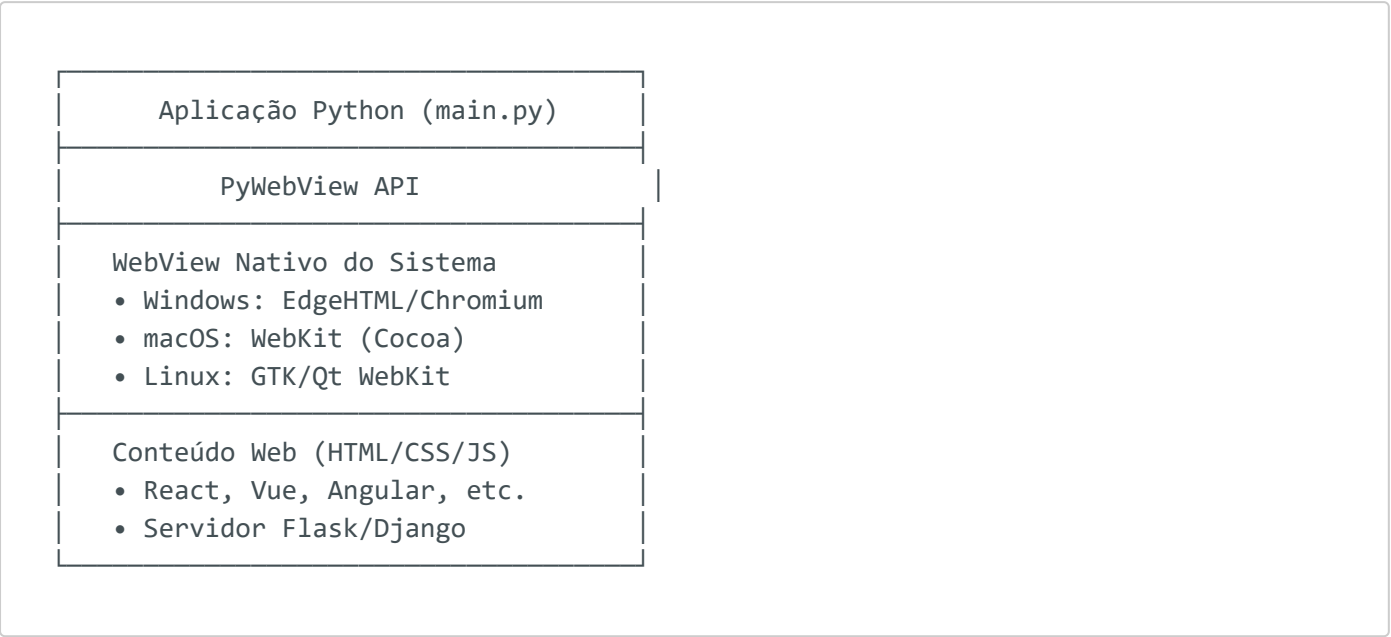
Comparação com Outras Tecnologias

Característica	PyWebView	Electron	PyQt/Tkinter
Tamanho do executável	Pequeno (~5-10 MB)	Grande (~100+ MB)	Médio (~20-50 MB)

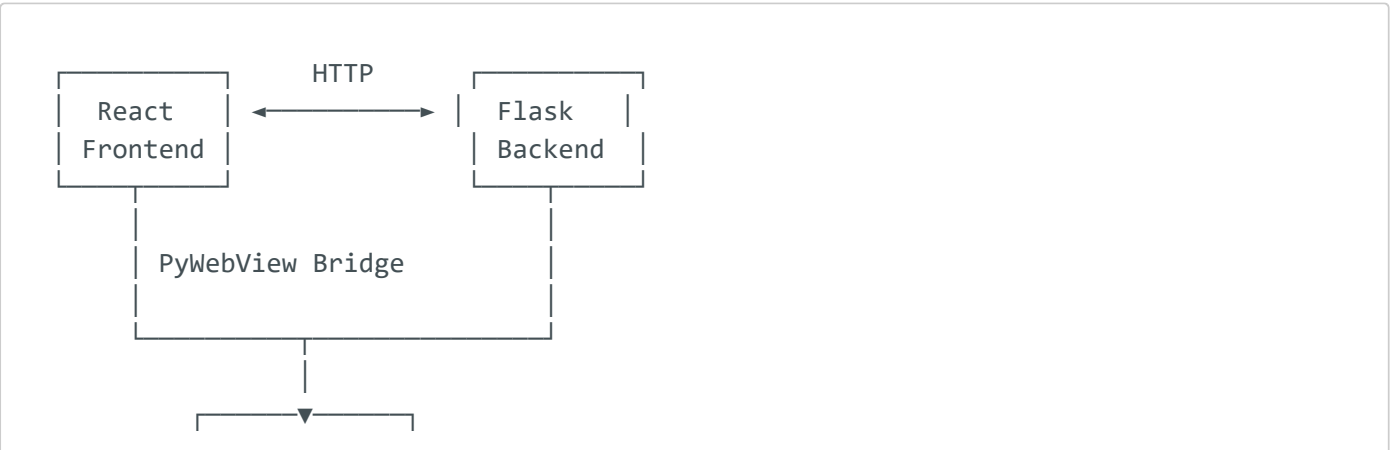
Característica	PyWebView	Electron	PyQt/Tkinter
Tecnologias web	✓ HTML/CSS/JS	✓ HTML/CSS/JS	✗ Widgets nativos
Integração Python	✓ Direta	⚠ Via IPC	✓ Direta
Curva de aprendizado	Baixa	Média	Alta
Performance	Boa	Boa	Excelente

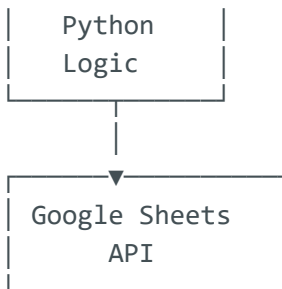
2. Arquitetura do PyWebView

Estrutura em Camadas



Fluxo de Dados





3. Instalação e Configuração



Instalação Básica

```
# Instalação simples
pip install pywebview

# Com suporte GTK (Linux)
pip install pywebview[gtk]

# Com suporte Qt (multiplataforma)
pip install pywebview[qt]

# Com suporte CEF (Chromium Embedded Framework)
pip install pywebview[cef]
```



Dependências por Sistema Operacional

macOS

```
# Não precisa de dependências adicionais
# Usa WebKit nativo via PyObjC
pip install pywebview
```

Linux (Ubuntu/Debian)

```
# GTK
sudo apt install python3-gi python3-gi-cairo gir1.2-gtk-3.0 gir1.2-webkit2-4.1
```

```
# Qt
sudo apt install python3-pyqt5 python3-pyqt5.qtwebengine
```

Windows

```
# Usa Edge WebView2 (Windows 10+)
# Ou MSHTML (Windows 7/8)
pip install pywebview
```

4. Conceitos Fundamentais



Criação de Janelas

```
import webview

# Janela básica
window = webview.create_window('Título', 'https://example.com')
webview.start()
```

Parâmetros Principais

```
webview.create_window(
    title='Minha App',          # Título da janela
    url='http://localhost:5000', # URL a carregar
    html='<h1>HTML direto</h1>', # Ou HTML direto

    # Dimensões
    width=800,                  # Largura
    height=600,                 # Altura
    x=100,                      # Posição X
    y=100,                      # Posição Y
    min_size=(400, 300),       # Tamanho mínimo

    # Comportamento
    resizable=True,             # Redimensionável
    fullscreen=False,          # Tela cheia
    minimized=False,           # Minimizada
    on_top=False,               # Sempre no topo

    # Aparência
    frameless=False,           # Sem bordas
```

```
easy_drag=True,          # Arrastar fácil
background_color='#FFFFFF', # Cor de fundo

# Funcionalidades
text_select=True,        # Seleção de texto
zoomable=False,          # Zoom
confirm_close=False,     # Confirmar fechamento

# API
js_api=None,              # Objeto Python exposto
)
```

Configurações Globais

```
import webview

# Configurações antes de start()
webview.settings = {
    'ALLOW_DOWNLOADS': False,
    'ALLOW_FILE_URLS': True,
    'OPEN_EXTERNAL_LINKS_IN_BROWSER': True,
    'OPEN_DEVTOOLS_IN_DEBUG': True,
}

webview.start()
```

5. Criando sua Primeira Janela

Exemplo Básico

```
import webview

def main():
    # Cria janela
    window = webview.create_window(
        'Minha Primeira App',
        'https://www.google.com'
    )

    # Inicia GUI
    webview.start()
```

```
if __name__ == '__main__':  
    main()
```



Com HTML Local

```
import webview  
  
html_content = '''  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Minha App</title>  
    <style>  
        body {  
            font-family: Arial, sans-serif;  
            display: flex;  
            justify-content: center;  
            align-items: center;  
            height: 100vh;  
            margin: 0;  
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);  
            color: white;  
        }  
    </style>  
</head>  
<body>  
    <h1>Hello PyWebView!</h1>  
</body>  
</html>  
'''  
  
window = webview.create_window('App Local', html=html_content)  
webview.start()
```

6. Integração com Flask



Servidor Flask Básico

```
from flask import Flask, render_template  
import webview  
  
# Cria app Flask  
app = Flask(__name__)
```



```
@app.route('/')
def index():
    return '<h1>Flask + PyWebView</h1>'

# Passa o app Flask para PyWebView
window = webview.create_window('Flask App', app)
webview.start()
```



Servidor em Thread Separada

```
from flask import Flask
import webview
import threading

app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>Servidor em Thread</h1>'

def start_flask():
    app.run(host='127.0.0.1', port=5000, debug=False)

# Thread para Flask
flask_thread = threading.Thread(target=start_flask, daemon=True)
flask_thread.start()

# Aguarda servidor iniciar
import time
time.sleep(1)

# Cria janela PyWebView
window = webview.create_window('App', 'http://127.0.0.1:5000')
webview.start()
```



Verificação de Servidor Pronto

```
import requests
import time

def wait_for_server(url, timeout=10):
    start = time.time()
    while time.time() - start < timeout:
        try:
            response = requests.get(url, timeout=1)
            if response.status_code == 200:
                return True
```

```

        except:
            time.sleep(0.5)
        return False

# Inicia Flask em thread
flask_thread = threading.Thread(target=start_flask, daemon=True)
flask_thread.start()

# Aguarda servidor
if wait_for_server('http://127.0.0.1:5000'):
    window = webview.create_window('App', 'http://127.0.0.1:5000')
    webview.start()
else:
    print("Erro: Servidor não iniciou")

```

7. Comunicação Python ↔ JavaScript

Expondo Funções Python para JavaScript

Backend Python

```

import webview

class API:
    def get_data(self):
        """Função Python chamável do JavaScript"""
        return {'message': 'Hello from Python!', 'value': 42}

    def calculate(self, a, b):
        """Função com parâmetros"""
        return a + b

    def process_async(self, data):
        """Operação assíncrona"""
        import time
        time.sleep(2) # Simula processamento
        return f"Processed: {data}"

# Expõe API
api = API()
window = webview.create_window('API Demo', 'index.html', js_api=api)
webview.start()

```

Frontend JavaScript

```
// Chamando funções Python do JavaScript
async function callPython() {
    // Função simples
    const data = await window.pywebview.api.get_data();
    console.log(data); // {message: "Hello from Python!", value: 42}

    // Função com parâmetros
    const result = await window.pywebview.api.calculate(10, 20);
    console.log(result); // 30

    // Função assíncrona
    const processed = await window.pywebview.api.process_async('test');
    console.log(processed); // "Processed: test"
}

// Aguarda API estar pronta
window.addEventListener('pywebviewready', () => {
    console.log('PyWebView API ready!');
    callPython();
});
```



Eventos e State

```
import webview

class API:
    def __init__(self, window):
        self.window = window

    def notify_user(self, message):
        """Executa JavaScript da janela"""
        js_code = f"alert('{message}')"
        self.window.evaluate_js(js_code)

    def update_dom(self, element_id, content):
        """Atualiza elemento do DOM"""
        js_code = f"""
        document.getElementById('{element_id}').textContent = '{content}';
        """
        self.window.evaluate_js(js_code)

# Uso
api = API(window)
window = webview.create_window('Events', 'index.html', js_api=api)

def on_loaded():
    api.update_dom('title', 'Loaded!')

window.events.loaded += on_loaded
webview.start()
```

8. Integração com Google Sheets



Autenticação OAuth2

```
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.discovery import build
import pickle
import os

SCOPES = ['https://www.googleapis.com/auth/spreadsheets']

def authenticate():
    """Autentica com Google Sheets"""
    creds = None

    # Carrega token existente
    if os.path.exists('token.json'):
        with open('token.json', 'rb') as token:
            creds = pickle.load(token)

    # Se não há credenciais válidas
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

    # Salva token
    with open('token.json', 'wb') as token:
        pickle.dump(creds, token)

    return build('sheets', 'v4', credentials=creds)
```



Operações CRUD

Leitura de Dados

```
def read_sheet(service, spreadsheet_id, range_name):
    """Lê dados da planilha"""
    result = service.spreadsheets().values().get(
```

```

        spreadsheetId=spreadsheet_id,
        range=range_name
    ).execute()

    values = result.get('values', [])
    return values

# Exemplo
service = authenticate()
data = read_sheet(service, 'SPREADSHEET_ID', 'Sheet1!A1:C10')
for row in data:
    print(row)

```

Escrita de Dados

```

def write_sheet(service, spreadsheet_id, range_name, values):
    """Escreve dados na planilha"""
    body = {'values': values}

    result = service.spreadsheets().values().update(
        spreadsheetId=spreadsheet_id,
        range=range_name,
        valueInputOption='RAW',
        body=body
    ).execute()

    return result

# Exemplo
service = authenticate()
data = [
    ['Nome', 'Idade', 'Email'],
    ['João', '30', 'joao@email.com'],
    ['Maria', '25', 'maria@email.com']
]
write_sheet(service, 'SPREADSHEET_ID', 'Sheet1!A1:C3', data)

```

Adicionar Dados

```

def append_sheet(service, spreadsheet_id, range_name, values):
    """Adiciona dados ao final da planilha"""
    body = {'values': values}

    result = service.spreadsheets().values().append(
        spreadsheetId=spreadsheet_id,
        range=range_name,
        valueInputOption='RAW',
        insertDataOption='INSERT_ROWS',
        body=body
    ).execute()

```

```

    return result

# Exemplo
service = authenticate()
new_data = [['Carlos', '35', 'carlos@email.com']]
append_sheet(service, 'SPREADSHEET_ID', 'Sheet1', new_data)

```

Deletar Linhas

```

def delete_row(service, spreadsheet_id, sheet_id, row_index):
    """Remove uma linha da planilha"""
    request_body = {
        'requests': [{
            'deleteDimension': {
                'range': {
                    'sheetId': sheet_id,
                    'dimension': 'ROWS',
                    'startIndex': row_index - 1,
                    'endIndex': row_index
                }
            }
        }]
    }

    result = service.spreadsheets().batchUpdate(
        spreadsheetId=spreadsheet_id,
        body=request_body
    ).execute()

    return result

```

Integração Completa PyWebView + Flask + Google Sheets

```

from flask import Flask, jsonify, request
import webview
import threading

app = Flask(__name__)
service = authenticate()
SPREADSHEET_ID = 'your_spreadsheet_id'

@app.route('/api/users', methods=['GET'])
def get_users():
    """Lista usuários"""
    data = read_sheet(service, SPREADSHEET_ID, 'Users!A2:C')
    users = [{'name': row[0], 'email': row[1], 'age': row[2]}]

```

```

        for row in data]
    return jsonify(users)

@app.route('/api/users', methods=['POST'])
def create_user():
    """Cria usuário"""
    data = request.json
    values = [[data['name'], data['email'], data['age']]]
    append_sheet(service, SPREADSHEET_ID, 'Users', values)
    return jsonify({'success': True})

# Inicia Flask em thread
def start_flask():
    app.run(host='127.0.0.1', port=5000, debug=False)

flask_thread = threading.Thread(target=start_flask, daemon=True)
flask_thread.start()

# Aguarda e inicia PyWebView
time.sleep(1)
window = webview.create_window('Google Sheets App', 'http://127.0.0.1:5000')
webview.start()

```

9. Ciclo de Vida da Aplicação

Eventos Disponíveis

```

import webview

window = webview.create_window('App', 'http://localhost:5000')

# Evento: Janela carregada
def on_loaded():
    print('Janela carregada!')
window.events.loaded += on_loaded

# Evento: Janela sendo fechada
def on_closing():
    print('Fechando...')
    return True # False cancela o fechamento
window.events.closing += on_closing

# Evento: Janela minimizada
def on_minimized():
    print('Minimizada')
window.events.minimized += on_minimized

# Evento: Janela restaurada
def on_restored():

```

```
print('Restaurada')
window.events.restored += on_restored

webview.start()
```



Fluxo Completo

```
import webview
import threading
import time

def initialize():
    """Código de inicialização"""
    print("Inicializando...")
    time.sleep(2)
    window.evaluate_js("console.log('Inicializado!')")

def main():
    global window

    # Cria janela
    window = webview.create_window('App', 'http://localhost:5000')

    # Configura eventos
    window.events.loaded += lambda: threading.Thread(
        target=initialize,
        daemon=True
    ).start()

    # Inicia
    webview.start(debug=True)

if __name__ == '__main__':
    main()
```

10. Boas Práticas



Estrutura de Projeto Recomendada

```
projeto/
├── backend/
│   ├── __init__.py
│   ├── app.py           # Flask app
│   └── api.py           # API endpoints
```



```
|   └─ sheets.py          # Google Sheets
└─ frontend/
    └─ src/
        └─ public/
            └─ build/      # Build React
└─ main.py                # Entry point
└─ config.py              # Configurações
└─ requirements.txt
```



Segurança

```
# Sempre valide entrada do usuário
def safe_api_call(user_input):
    # Validação
    if not isinstance(user_input, str):
        raise ValueError("Input inválido")

    # Sanitização
    clean_input = user_input.strip()

    # Uso seguro
    return process(clean_input)

# Use HTTPS em produção
webview.create_window('App', 'https://myapp.com')

# Desabilite DevTools em produção
webview.settings['OPEN_DEVTOOLS_IN_DEBUG'] = False
webview.start(debug=False)
```



Performance

```
# Use threads para operações longas
import threading

def long_operation():
    # Processamento pesado
    time.sleep(5)
    return "Resultado"

def async_call():
    thread = threading.Thread(target=long_operation, daemon=True)
    thread.start()

# Cache de dados
from functools import lru_cache
```

```
@lru_cache(maxsize=128)
def get_cached_data():
    return expensive_operation()
```



Modo Debug vs Produção

```
import os

DEBUG = os.getenv('DEBUG', 'False') == 'True'

webview.settings = {
    'OPEN_DEVTOOLS_IN_DEBUG': DEBUG,
}

webview.start(debug=DEBUG)
```

11. Troubleshooting



Problemas Comuns

Porta já em uso

```
import socket

def find_free_port():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(('', 0))
        return s.getsockname()[1]

port = find_free_port()
app.run(port=port)
```

Google Sheets Error 403

Solução:

1. Adicione usuários de teste no Google Cloud Console
2. Verifique se a API está ativada
3. Confirme as permissões do OAuth

PyWebView não abre janela (Linux)

```
# Instale dependências GTK
sudo apt install python3-gi gir1.2-webkit2-4.1

# Ou use Qt
pip install pywebview[qt]
```

SSL Certificate Error

```
# Temporário para desenvolvimento
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```



Recursos Adicionais

Links Úteis

- [Documentação Oficial PyWebView](#)
- [GitHub PyWebView](#)
- [Google Sheets API](#)
- [Flask Documentation](#)
- [React Documentation](#)

Exemplos Avançados

Veja mais exemplos em:

- [/docs/PROJETO_COMPLETO.md](#) - Visão geral do projeto
- [/docs/INSTRUcoes_USO.md](#) - Manual completo
- [/docs/GOOGLE_SHEETS_SETUP.md](#) - Setup detalhado



Conclusão

Este tutorial cobriu os conceitos essenciais do PyWebView e sua integração com Google Sheets. Com esses conhecimentos, você pode criar aplicações desktop completas usando tecnologias web que você já conhece!

Próximos Passos

1. Experimente criar sua própria janela
2. Integre com seu framework web favorito
3. Adicione funcionalidades do Google Sheets
4. Empacote sua aplicação para distribuição

Happy coding! 🚀