PyGraphiX: Una Herramienta Web para Facilitar la Codificación y Creación de Interfaces Gráficas de Usuario Basadas en Python

Mateo Sebastián Espinosa Martínez[1]
mbespinosam@uce.edu.ec

Julio César Mendoza Tello[1]
jcmendoza@uce.edu.ec

Mauro Leonardo Rosas Lara[1]
mrosas@uce.edu.ec

[1]Universidad Central del Ecuador
Quito, Ecuador

DOI: https://doi.org/10.56124/encriptar.v8i16.007

Resumen

El desarrollo de aplicaciones gráficas basadas en Python presenta diversos desafíos, especialmente por la complejidad de su sintaxis y la falta de disponibilidad de herramientas intuitivas. Esto representa un obstáculo para programadores novatos y ralentiza el desarrollo de interfaces eficientes. En este contexto, surge la necesidad de desarrollar una herramienta web para crear interfaces gráficas de manera fácil, sin requerir conocimientos avanzados de programación. Con el fin de abordar esta problemática, el objetivo de esta investigación es desarrollar una herramienta web que facilite la codificación y creación de interfaces gráficas basadas en Python, denominada “PyGraphiX”. Para ello, se analizan los inconvenientes asociados al desarrollo gráfico utilizando la biblioteca estándar Tkinter de Python, ampliamente utilizada en proyectos educativos. Como solución, este estudio propone una herramienta basada en tecnologías web como HTML, CSS, y JavaScript, las cuales permiten construir interfaces dinámicas y flexibles. Luego, se describe un proceso de generación automática de código en tres etapas: análisis, desarrollo y pruebas funcionales. Adicionalmente, se presenta un método para convertir tamaños en píxeles, con el fin de optimizar la compatibilidad del diseño con el entorno Python. De esta manera, esta solución permite a los programadores obtener automáticamente scripts funcionales (basados en Python) a partir de la configuración personalizada de los elementos gráficos, lo que facilita la implementación en proyectos reales. En resumen, PyGraphiX permite diseñar interfaces de manera más intuitiva, superando las limitaciones de Tkinter. Finalmente, se presentan las conclusiones del estudio y se proponen futuras investigaciones.

Palabras clave: interfaz gráfica de usuario; software de código abierto; lenguaje de programación; software de aplicación; Python.

PyGraphiX: A Web Tool to Facilitate Coding and Creation of Python-based Graphical User Interfaces

ABSTRACT

Developing graphical applications based on Python presents several challenges, especially due to the complexity of its syntax and the lack of availability of intuitive tools. This represents an obstacle for novice programmers and slows down the development of efficient interfaces. In this context, the need arose to develop a web tool to create graphical interfaces easily, without requiring advanced programming knowledge. To address this problem, this research aims to develop a web tool called PyGraphiX, which facilitates the coding and creation of Python-based graphical interfaces. To this end, the drawbacks associated with graphical development using Python's standard Tkinter library, which is widely used in educational projects, are analyzed. As a solution, this study proposes a tool based on web technologies such as HTML, CSS, and JavaScript, which enable the creation of dynamic and flexible interfaces. Then, the automatic code generation process is described in three stages: analysis, development, and functional testing. Additionally, a method for converting sizes to pixels is presented to optimize design compatibility with the Python environment. This solution allows programmers to automatically generate functional (Python-based) scripts from custom graphical element configurations, making implementation in real-world projects easier. In summary, PyGraphiX allows for more intuitive interface design, overcoming the limitations of Tkinter. Finally, the conclusions of the study are presented, and future research is proposed.

Keywords: graphical user interface; open-source software; computer languages; application software; Python.

1. Introducción

Tkinter es la librería estándar basada en Python para crear interfaces gráficas de usuario (GUI). A través de esta librería, componentes gráficos son incorporados dentro de una interfaz. En este sentido, existe una gran variedad de componentes gráficos tales como botón, menú, barra de progreso, etiqueta, cuadro de texto y otros elementos interactivos. Sin embargo, el uso de Tkinter

presenta dos inconvenientes.

Primero, falta de una plataforma intuitiva para el diseño de la GUI. El diseño de interfaces gráficas de usuario basadas en Python ha sido históricamente un proceso laborioso y desafiante (Sherathiya et al., 2021). Tkinter, aunque poderoso, requiere un conocimiento profundo de Python y mucha codificación manual. Tkinter es un constructor gráfico basado en comandos y no dispone de una interfaz visual que agilite el diseño de la GUI. Es decir, cada propiedad del componente gráfico (tales como estilo y tipografías) debe ser configurado a través de línea de comando. Tkinter, al ser una biblioteca relativamente simple, carece de las capacidades avanzadas de personalización. Esto puede resultar en interfaces menos atractivas y funcionales, especialmente en aplicaciones que requieren un diseño moderno y sofisticado. Además, el rendimiento de Tkinter puede no ser adecuado para aplicaciones más complejas o que manejan grandes volúmenes de datos, lo que puede llevar a una experiencia de usuario lenta y poco receptiva (Charatan & Kans, 2022). Consecuentemente, el diseño de la GUI resulta laborioso, lento, desalentador, propenso a errores, poco intuitivo y frustrante para desarrolladores con poca experiencia.

Segundo, falta de una plataforma integrada de soluciones. Tkinter no dispone de un editor visual para arrastrar y soltar elementos sobre una GUI. Esto limita la creatividad para integrar una interfaz con otras soluciones informáticas. El desarrollo de interfaces gráficas de usuario requiere una variedad de herramientas y recursos que aborden las diferentes necesidades del proceso de diseño (Katricheva et al., 2020). La falta de una plataforma integrada que combine estas herramientas puede limitar significativamente la eficacia y la productividad de los desarrolladores (Antonova et al., 2019). Además, la falta de soporte para dispositivos móviles limita el desarrollo de aplicaciones para teléfonos inteligentes. Esto obliga a los desarrolladores a buscar soluciones alternativas o complementar Tkinter con otras tecnologías,

lo que puede aumentar la complejidad del desarrollo y el mantenimiento del software. Consecuentemente, la integración con otras tecnologías es complicada. A menudo se requiere conocimientos avanzados y puede ser propensa a errores, lo que aumenta el tiempo y esfuerzo necesarios para desarrollar y mantener aplicaciones (Bronshteyn, 2013).

Con estas consideraciones, el objetivo de esta investigación es desarrollar una herramienta web para crear interfaces gráficas de usuario basadas en Python. Para ello, una herramienta web (llamada PyGraphiX) fue desarrollada para proporcionar una interfaz accesible y fácil de utilizar. Consecuentemente, las secciones del documento son las siguientes. Sección 2 explica métodos, requerimientos funcionales, hardware y software utilizados en esta investigación. Sección 3 detalla la fase de desarrollo y los resultados obtenidos a partir de las pruebas funcionales realizadas a la aplicación. Consecuentemente, discutimos y comparamos nuestros resultados con previas investigaciones. Finalmente, la sección 4 describe conclusiones y futuras investigaciones.

2. Materiales y Métodos

Materiales utilizados tales como software y hardware utilizados son descritos. Además, se explica el método y los requerimientos funcionales para el desarrollo de la herramienta web.

2.1 Materiales: hardware y software utilizados

Un computador con las siguientes características fue utilizado: 8 GB de RAM, 2 GB de tarjeta gráfica, procesador Intel Core i7 de octava generación, y sistema operativo Windows 10. En este contexto, el siguiente software fue utilizado, a saber.

- Lenguaje de marcado de hipertexto (HTML, *HyperText Markup Language*). Es la codificación estándar que permite la visualización de

documentos en un navegador web. Este lenguaje es complementado con hojas de estilo en cascada y JavaScript para crear páginas web interactivas y visualmente atractivas. Para ello, HTML utiliza marcadores escritos, conocidos como etiquetas, las cuales están encerrados entre paréntesis angulares (por ejemplo, <html>). Estas etiquetas definen la apariencia y el orden interno de una página web, así como los scripts o rutinas que operan dentro de ellas. De esta manera, HTML define la ubicación de los recursos para su representación, ordenamiento y definición (Kawamura & Yamamoto, 2021). Las aplicaciones más comunes de este lenguaje son: páginas web, aplicaciones web, correo electrónico y juegos.

- JavaScript. Es un lenguaje de programación utilizado para crear aplicaciones web dinámicas e interactivas a través de la codificación de eventos de usuarios, manipulación de datos y animación de elementos gráficos. JavaScript es ejecutado en el navegador; en este contexto, puede interactuar con el modelo de objetos del documento (DOM, *Document Object Model*) para modificar la estructura, estilo y contenidos de la página web. Esto permite a los desarrolladores crear experiencias de usuario ricas y reactivas. Además, JavaScript es utilizado para crear sitios web dinámicos y aplicaciones de una sola página (Freeman, 2022). A través de frameworks como React Native, aplicaciones móviles nativas y videojuegos pueden ser desarrollados e implementados para navegadores web. Incluso, Node.js utiliza JavaScript para desarrollar e implementar servicios backend.
- Hojas de estilo en cascada (CSS, *Cascading Style Sheets*). Es un lenguaje de composición de estilos para estructurar y diseñar documentos. De esta manera, CSS establece los límites entre el contenido y el diseño visual. En este sentido, éste proporciona un diseño estandarizado para características tales como: color, fuente, espacios del contenido, división en múltiples columnas y otras características gráficas. Para ello, CSS define y aplica reglas de estilo a los elementos HTML, a través de propiedades y valores (Attardi,

2020). De esta manera, CSS combinado con HTML y JavaScript son utilizados para crear diseños web atractivos y responsivos.

• Python. Es un lenguaje versátil que se adapta a una variedad de paradigmas de programación, tales como: orientación de objetos, imperativo y funcional. Frameworks como Django y Flask soportan Python para facilitar la construcción de sitios web robustos y aplicaciones tanto de escritorio como servidor. En esta línea, bibliotecas basadas en Python (tales como TensorFlow y Scikit-learn) favorecen el desarrollo de modelos de aprendizaje de máquina (Charatan et al., 2019).

• Tkinter. Es la biblioteca estándar basada en Python para la creación de interfaces gráficas de usuario. Es un envoltorio alrededor del kit de herramientas Tk, el cual es una extensión del lenguaje de programación Tcl. Esta biblioteca permite a los desarrolladores crear interfaces gráficas utilizando widgets como botones, menús, barras de progreso, etiquetas, cuadros de texto, entre otros. Tkinter brinda soporte a utilidades del sistema, a saber: administración de archivos, edición de texto y monitoreo del sistema. Consecuentemente, scripts son codificados para opciones de visualización, automatización de tareas repetitivas y administración de sistemas (Moruzzi, 2020).

2.2 Método

Esta investigación es conducida a través de las tres fases consecutivas del modelo en cascada: análisis, desarrollo y pruebas (Thesing et al., 2021).

• Con respecto al análisis. Para cumplir el objetivo de esta investigación, requerimientos funcionales fueron identificados.

• Con respecto al desarrollo. Utilizando JavaScript, hoja de estilo en cascada y Python, se diseña y codifica los componentes gráficos para el componente principal.

• Con respecto a las pruebas. Los requerimientos funcionales son validados acorde al objetivo de esta investigación.

2.3 Análisis: requerimientos funcionales

Tres requerimientos funcionales fueron identificados, a saber: (a) creación de un formulario principal que incluya componentes gráficos, (b) creación de una hoja de estilos en cascada, y (c) conversión de elementos para generar automáticamente codificaciones.

(a) Creación de un formulario principal. Se requiere de dos contenedores secundarios: base y propiedades. El contenedor base es utilizado para colocar los elementos gráficos dentro de él, proporcionando al usuario una vista preliminar del diseño final. El contenedor de propiedades es utilizado para ajustar y configurar las propiedades de los elementos o componentes gráficos. En esta línea, los componentes utilizados son: panel, etiqueta, menú de opciones, lienzo, texto, texto multilínea, lista, botón, casilla de verificación, y casilla de opción. A excepción del elemento menú de opciones, cada uno de estos elementos posee las siguientes propiedades: identificador, color de fondo, ancho, largo, posición, tamaño, tipo, color y estilo de fuente. El elemento menú no incluye las anteriores características porque su función es organizar las opciones y comandos de acceso rápido (atajos); para ello, el menú (de opciones) puede tener hasta ocho secciones con un número indefinido de subsecciones.

(b) Creación de hoja de estilo en cascada. Se requiere contenedores y componentes gráficos configurados a través de clases de plantillas. En este contexto, para los cuadros de texto y lista de opciones, se requiere: borde redondeado, ancho de 60%, altura de 30 pixeles, color negro para fuente, alineación central y vertical con un margen inferior de 3%. Con respecto a las etiquetas de los elementos, se requiere: color blanco para el texto, tamaño de fuente de 15 pixeles, ancho de 20%, margen izquierdo de 7%, margen inferior de 7%, margen inferior de 2%, fuente *Georgia, Times New Roman o Serif*. En esta misma línea, para los botones y cuadro de texto multilínea, se requiere: margen superior de 2.4%, margen izquierdo de 3%, ancho de 150 pixeles,

fuente de color negro, fondo de color marrón claro, tamaño de fuente de 12 pixeles, fuente Arial. Además, para las casillas de verificación y opción, se requiere: margen superior de 2.4%, margen izquierdo de 3%, fuente Arial de color negro con tamaño de 12 pixeles. Con estas especificaciones, una plantilla es necesaria para realizar a todos los elementos, los siguientes ajustes generales: margen superior e inferior de 5%, ancho de 100% y una altura automática (Elumalai, 2021). De esta manera, se garantiza la adaptación correcta de los elementos gráficos al espacio disponible en el formulario principal.

(c) Conversión de elementos. La generación automática de código requiere una conversión de pixeles a caracteres. Para ello, es necesario que el código sea ajustado a una plantilla específica para Tkinter. El resultado es un script (basado en Python), donde se puede visualizar el contenido creado.

3. Resultados y Discusión

A continuación, se explica los resultados basados en el desarrollo y pruebas funcionales de la aplicación. Luego, la discusión es abordada a partir de la comparación con previas investigaciones.

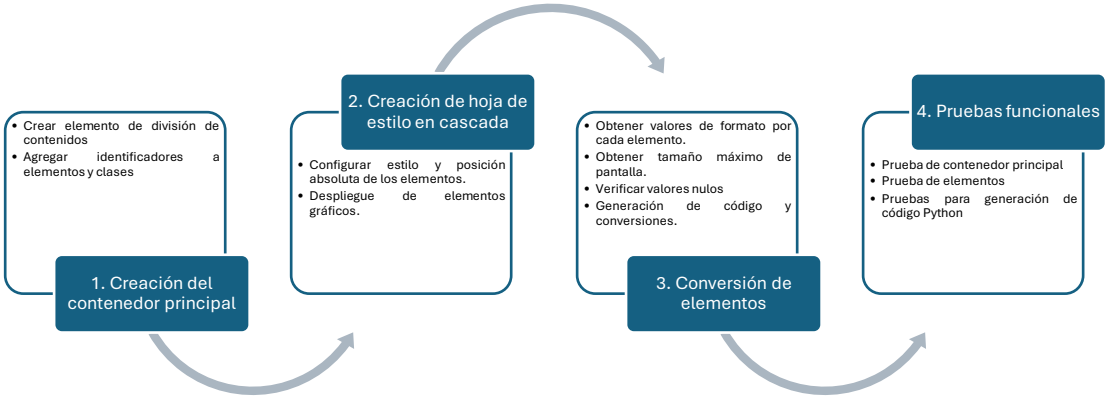
3.1 Desarrollo y Pruebas

En esta etapa, cuatro actividades secuenciales fueron identificados. Figura 1 muestra un resumen de las tareas llevadas a cabo durante este proceso de desarrollo de software.

Primero, creación del contenedor principal. HTML fue utilizado para estructurar la aplicación. Varios elementos HTML fueron colocados dentro del contenedor, entre ellos destacan <button>, <input>, y . En este sentido, un identificador (ID) es asignado a cada elemento para facilitar su manipulación y configuración. Figura 2 muestra la codificación inicial de esta

actividad.

Figura 1. Etapas para desarrollar el software.



Fuente: Los autores.

Figura 2. Creación del contenedor principal.

```
<div id="P1"> <div id="MenuDiv"> <h1 style="text-align: center;">Menú Desplegable </h1> <br>
<div id="header"></div></div><div id="interfaz"></div></div>
<div id="P2">
    <div id="HerramientasDV">
        <a><img id="PanelHerramientas" src="Imagenes/PanelOpciones.png"></a>
        <br>
        <button class="btn draw-border" id="VisualizarCP" type="button">Configuración del panel</button>
        <button class="btn draw-border" id="VisualizarCF" type="button">Lienzo</button>
        <button class="btn draw-border" id="VisualizarCT" type="button">Texto</button>
        <button class="btn draw-border" id="VisualizarCE" type="button">Entry</button>
        <button class="btn draw-border" id="VisualizarCEM" type="button">Entry MultiLinea</button>
        <button class="btn draw-border" id="VisualizarCCBX" type="button">Combobox</button>
        <button class="btn draw-border" id="VisualizarCB" type="button">Button</button>
        <button class="btn draw-border" id="VisualizarCBCR" type="button">Checkbox /  Radio buttons</button>
        <button class="btn draw-border" id="VisualizarMN" type="button">Menú</button>
        <button class="btn draw-border" type="button" id="ExtraerDatos">Extraer Datos</button>
    </div>
```

Fuente: Los autores.

Segundo, creación de hoja de estilo (CSS). Utilizando el ID para cada elemento, diseños individuales y clases fueron definidas. En este sentido, tres configuraciones fueron utilizadas, a saber. (a) Estilo, el cual permite incluir color, tamaño, tipo de fuente y ancho. (b) Posición absoluta, para permitir el

posicionamiento exacto de los elementos, de tal manera que se adapten automáticamente de forma porcentual al tamaño de la pantalla. (c) Despliegue, a través de su valor <none> para ocultar elementos en momentos específicos. Estos elementos fueron fundamentales para establecer el diseño visual tanto de la plataforma como de los componentes generados para Tkinter. Esto optimiza el código generado y facilita el proceso de diseño. En este sentido, la herramienta permite ajustar fácilmente propiedades visuales y funcionales de los elementos, como el color, tipo de letra, ancho y largo, permitiendo un mayor grado de personalización. Figura 3 muestra un extracto de la plantilla CSS utilizada.

Figura 3. Codificación de plantillas utilizando CSS

```
css
Copiar código
.Entry {
    border-radius: 5px;
    width: 60%;
    height: 30px;
    color: black;
    text-align: center;
    align-items: center;
    margin-bottom: 3%; }
```

```
css
Copiar código
.Combobox {
    border-radius: 5px;
    width: 60%;
    height: 30px;
    color: black;
    text-align: center;
    align-items: center;
    margin-bottom: 3%; }
```

```
css
Copiar código
.Input {
    margin-top: 2.4%;
    margin-left: 3%;
    width: 150px;
    color: #000000;
    background-color:
#C0A275;
    font-size: 12px;
    font-family: Arial;
    position: absolute;
    display: inline-block; }
```

```
css
Copiar código
.CheckButton {
    display: inline-block;
    margin-top: 2.4%;
    margin-left: 3%;
    color: #000000;
    background-color:
#C0A275;
    font-size: 12px;
    font-family: Arial;
    position: absolute; }
```

Fuente: Los autores.

Tercero, conversión de cada elemento HTML. Cuatro tareas secuenciales fueron ejecutadas para realizar este proceso de conversión. (a) Obtención de valores de formato, tales como ancho, alto, color, ID y posición del elemento. (b) Obtención del tamaño máximo de pantalla. Ajustes automáticos son configurados para diversos tamaños de pantalla, manteniendo valores proporcionales tanto para el alto como para el ancho. (c) Verificación de valores nulos. En el caso de existir valores nulos para un campo o elemento, el usuario es notificado para que complete los valores faltantes.

(d) Generación de código para Tkinter. Esto asegura que el diseño creado en la plataforma HTML se convierta correctamente en una interfaz funcional basada en Tkinter. En este contexto, debido a que Python no utiliza pixeles sino caracteres (como unidad de medida), es necesario realizar esta conversión de tamaño. Para realizar esta conversión, se considera que 28 pixeles equivalen a 3 caracteres de tamaño de fuente igual a 12. Una vez obtenido el valor en pixeles, se procede a realizar un ajuste para otros tamaños de pixeles diferentes a 28, tal como se indica en la ecuación (1). Sin embargo, el tamaño de los caracteres cambia de acuerdo con el tamaño de la letra. En este sentido, es necesario realizar un segundo ajuste, la cual representa la fórmula utilizada para convertir de pixeles a número de caracteres, tal como se indica en la ecuación (2). Esto simplifica aún más el proceso de diseño y asegura una alineación correcta de los elementos. Esta funcionalidad adicional permite a los desarrolladores centrarse en el aspecto visual de la interfaz sin preocuparse por los detalles técnicos de la conversión de unidades. Figura 4 muestra la codificación utilizada para esta conversión.

$$\text{Ajuste} = \frac{\text{Tamaño en pixeles } \times 3}{28} \qquad (1)$$

$$\text{Conversión} = \frac{12 \times \text{Ajuste}}{\text{Tamaño de la letra}} \qquad (2)$$

Cuarto, pruebas. Estas fueron llevadas secuencialmente para satisfacer el requerimiento funcional de generar codificaciones basadas en Python. Con esta consideración, tres pruebas funcionales son esenciales: (i) prueba de formulario principal, (ii) prueba de elementos basados en CSS, y (iii) prueba para generación de código.

(i) Prueba de formulario principal. Una vez iniciado el programa, éste mostrará dos marcos secundarios: un panel (donde se colocarán los elementos gráficos), y propiedades (donde se visualizarán cada una de las

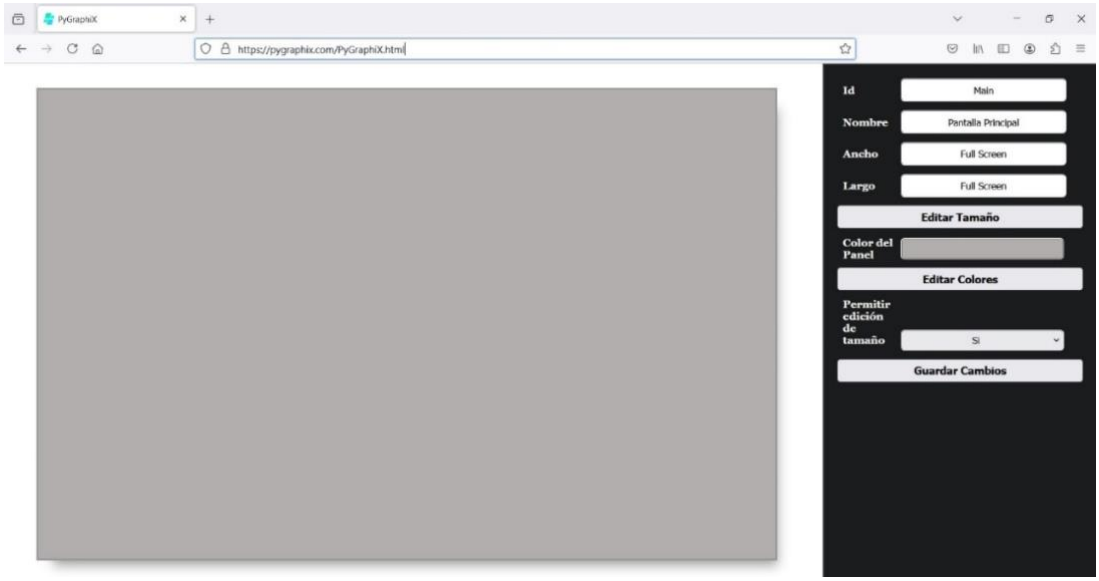
características gráficas, tales como identificador, nombre, color y tamaño). Figura 5 muestra el formulario principal con las propiedades descritas.

Figura 4. Código basado en JavaScript para realizar conversiones.

```
var largo = document.getElementById("ConfPanelLargo").value;
var ancho = document.getElementById("ConfPanelAncho").value;
var color = document.getElementById("ConfPanelColor").value;
var id = document.getElementById("ConfPanelId").value;
var nombre = document.getElementById("ConfPanelNombre").value;
var CTamaño = document.getElementById("ConfPanelCombo").value;

largo = largo.replace("Full Screen",LargoMaximo);
ancho = ancho.replace("Full Screen",AnchoMaximo);

if(largo != "" && ancho != "" && color != "" && id != "" && nombre != ""){

     var ConversionAncho = Math.round((screen.width*ancho)/AnchoMaximo);
     var ConversionLargo = Math.round((screen.height*largo)/LargoMaximo);

     var Texto = "from tkinter import *\n"+
                 "from tkinter import ttk\n"+
                 "class Ventana():\n"+
                 "    def __init__(self):\n"+
                 "        "+id+"= Tk()\n"+
                 "        "+id+".title('"+nombre+"')\n"+
                 "        "+id+".geometry('"+ConversionAncho+"x"+ConversionLargo+"')\n"+
                 "        "+id+".config(bg='"+color+"')\n"+id+".mainloop()\n"+"Ventana()";
}
```

Fuente: Los autores.

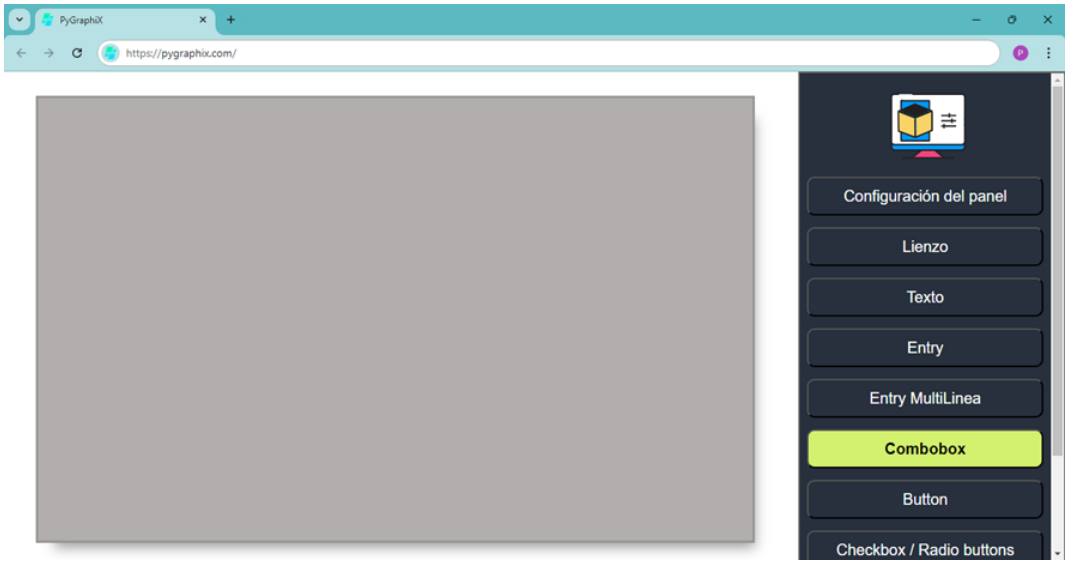
Figura 5. Formulario principal.



Fuente: Los autores

(ii) Prueba de elementos gráficos. Una vez configurado y guardado los cambios del formulario principal, un menú de opciones es visualizado para seleccionar los elementos gráficos del contenedor. En este sentido, la figura 6 muestra las siguientes opciones: configuración del panel, lienzo, texto, texto multilínea, lista desplegable, casillas de verificación, botón, menú, entre otros. Además, un segundo menú de opciones es activado acorde al elemento gráfico seleccionado. Figura 7 muestra una breve representación del contenedor de propiedades para un menú de opciones.

Figura 6. Elementos gráficos que soporta nuestra aplicación.

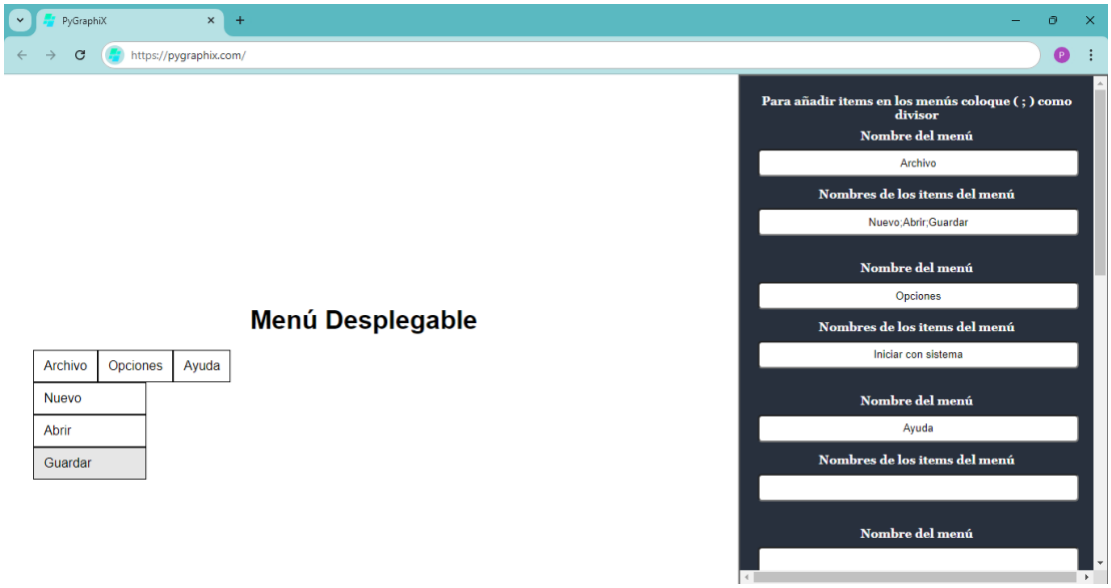


Fuente: Los autores.

(iii) Prueba de generación de código. Una vez concluido el diseño gráfico de la interfaz, se procede a generar y extraer la codificación basada en Python. En este contexto, este código puede ser copiado y ejecutado dentro de un editor de Python (tales como Anaconda y Pycharm). De esta manera, se facilita la codificación e implementación de interfaces gráficas basadas en Python.

Además, al facilitar la exportación del diseño a código Python, ayuda a mejorar el rendimiento de las aplicaciones desarrolladas con Tkinter. Aunque Tkinter sigue siendo limitado en cuanto a soporte para dispositivos móviles, PyGraphiX hace que el desarrollo de aplicaciones de escritorio sea mucho más accesible y eficiente. Como ejemplo, la figura 8 muestra el diseño gráfico finalizado utilizando PyGraphiX. En esta línea, la figura 9 muestra la codificación generada automáticamente

Figura 7. Configuración de las propiedades de un menú de opciones.

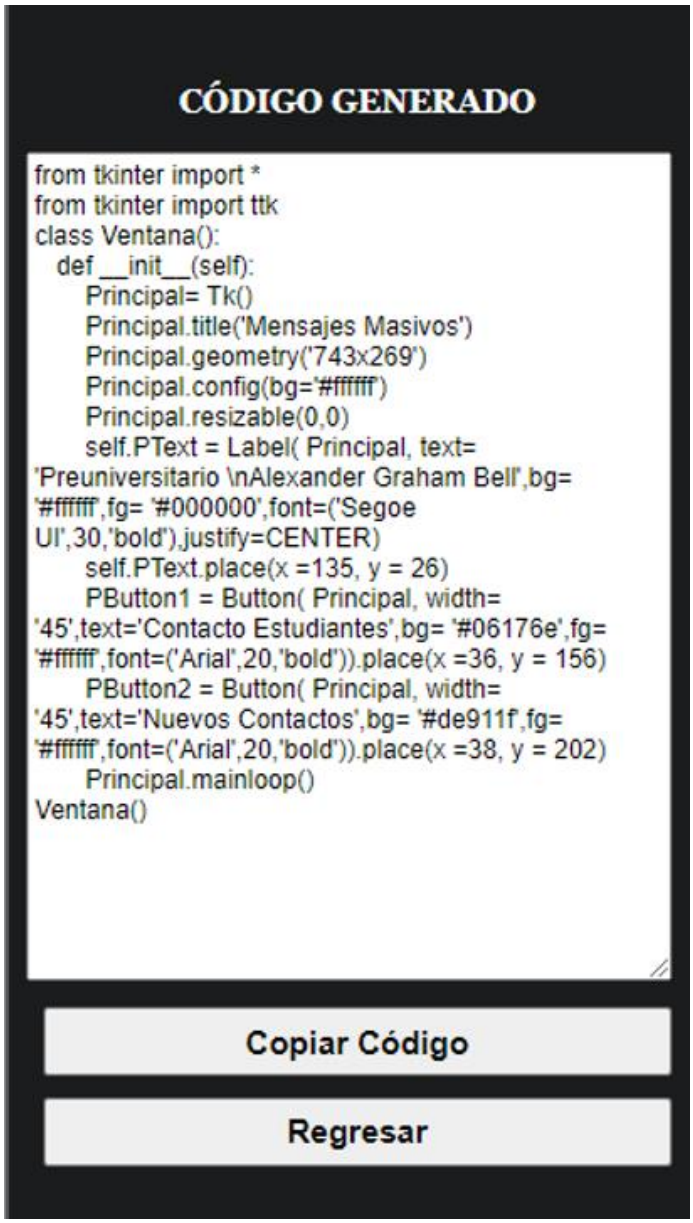


Fuente: Los autores.

Figura 8. Interfaz gráfica diseñada como ejemplo utilizando PyGraphiX.

Fuente: Los autores

Figura 9. Codificación basada en Python generada por PyGraphiX.



Fuente: Los autores.

3.2 Discusión

Previas investigaciones han desarrollado aplicativos para interfaces gráficas. En este contexto, Qt Designer es una herramienta de diseño visual

integrada dentro del marco de desarrollo Qt y puede utilizarse con PyQt y PySide para generar código C++ y Python (Willman, 2022). De igual manera, wxPython es un binding de la biblioteca gráfica wxWidgets para Python y cuenta con una variedad de widgets. Esta herramienta genera interfaces gráficas similares a las nativas del sistema operativo (Dong Gao et al., 2022). En esta línea, Kivy Designer es una herramienta visual para crear aplicaciones Kivy, la cual es una biblioteca de Python para aplicaciones gráficas multitáctiles (Gladstone, 2022). Todas estas herramientas soportan aplicaciones multiplataforma para varios sistemas operativos (Windows, macOS, iOS, y Linux); sin embargo, solo permiten el desarrollo de aplicaciones de escritorio y no dispone de una versión web, lo cual evidencia dificultades de accesibilidad y flexibilidad para los programadores que buscan soluciones colaborativas más flexibles. A diferencia de otras plataformas, nuestra plataforma ofrece cuatro ventajas únicas.

Primero, independencia del sistema operativo. Esta independencia es crucial en el desarrollo de aplicaciones, ya que permite que el software funcione de manera uniforme en diversas plataformas. Muchas herramientas presentan limitaciones en compatibilidad, lo que complica el proceso de desarrollo y aumenta los costos de pruebas y ajustes. PyGraphiX aborda este problema proporcionando una solución verdaderamente independiente del sistema operativo para el diseño de interfaces gráficas con Tkinter. Al ser una herramienta web en línea, no está limitada por los requerimientos de los sistemas operativos. Los usuarios pueden seleccionar los elementos para configurar su posición y otras propiedades detalladamente. Los desarrolladores pueden crear y personalizar las GUI en un entorno, el cual funciona de manera consistente en Windows, macOS y Linux. La herramienta exporta el diseño a código Python, asegurando que las aplicaciones se ejecuten uniformemente en cualquier sistema operativo. Esta independencia

simplifica el desarrollo, reduce el tiempo y los recursos necesarios para pruebas y ajustes, y garantiza una experiencia de usuario coherente en todas las plataformas. Además, facilita la colaboración entre desarrolladores que utilizan diferentes entornos de trabajo, permitiendo compartir y ajustar diseños sin problemas de compatibilidad.

Segundo, configuración personalizada. La capacidad de personalización es crucial en el desarrollo de interfaces gráficas (GUI) para adaptarlas a las necesidades específicas de una aplicación y las preferencias de los usuarios. Sin una configuración robusta, las interfaces pueden parecer genéricas y limitadas. PyGraphiX comparte la capacidad de personalización de otras herramientas, permitiendo a los programadores ajustar cada componente según sus preferencias, incluyendo tamaño, color, ancho y largo, con verificaciones automáticas para garantizar la correcta configuración. Esto reduce errores y acelera el diseño, permitiendo agregar componentes fácilmente.

Tercero, conversión y validación automática. En el desarrollo de interfaces gráficas de usuario, la conversión y validación de parámetros es una tarea crítica que puede consumir mucho tiempo y ser propensa a errores si se realiza manualmente. La necesidad de convertir unidades (como píxeles a caracteres), y validar que los elementos estén correctamente configurados, pueden ralentizar el proceso de desarrollo y aumentar la complejidad. Las conversiones manuales no solo son tediosas, sino que también aumentan la probabilidad de errores de cálculo que pueden llevar a una interfaz desalineada y poco atractiva. En este sentido, PyGraphiX extrae y analiza las configuraciones de los elementos, ajustando automáticamente las dimensiones que superen el tamaño de la pantalla. Nuestra herramienta soluciona este problema ofreciendo capacidades de conversión y validación automática. Además, PyGraphiX convierte automáticamente las dimensiones de píxeles a caracteres utilizados por Tkinter, asegurando que los elementos

de la interfaz se ajusten y se alineen correctamente. La conversión automática permite a los desarrolladores centrarse más en la estética y en la funcionalidad de la interfaz, sin preocuparse por los detalles técnicos de las unidades de medida.

Además, PyGraphiX valida automáticamente la configuración de los elementos de la interfaz, asegurando que todos los parámetros sean coherentes y funcionales. Esta validación automática ayuda a detectar y corregir errores antes de la implementación, mejorando la calidad del diseño final y acelerando el desarrollo. La plataforma valida automáticamente que las propiedades de los elementos, como tamaños, colores y posiciones, sean válidas y compatibles con las especificaciones de Tkinter, lo que evita problemas de compatibilidad y errores de ejecución.

Cuarto, generación de código en tiempo real. La herramienta genera código Python en tiempo real, proporcionando una plantilla lista para usar, lo cual acelera el desarrollo y prueba de las aplicaciones. Además, nuestra plataforma ofrece una mayor facilidad para actualizaciones de contenido. La implementación de tecnologías web y la automatización del proceso de generación de código ha representado un avance significativo en el desarrollo de interfaces gráficas en Python, especialmente en el contexto de las limitaciones encontradas con Tkinter. Al adoptar tecnologías como HTML, CSS y JavaScript, se logra una mayor flexibilidad en el diseño y la personalización de las interfaces, permitiendo a los desarrolladores crear aplicaciones con un aspecto visual más atractivo y una experiencia de usuario más intuitiva.

4. Conclusiones

Esta investigación desarrolló PyGraphiX, una herramienta para generar interfaces gráficas basadas en Python. Para ello, cuatro fases fueron llevadas a cabo. Primero, creación del formulario principal, que representa el diseño de la estructura base de la herramienta. Segundo, creación de hoja de estilos,

que mejora la apariencia y usabilidad de la herramienta web. Tercero, conversión de elementos, que implementa mecanismos para transformar elementos gráficos en código funcional basado en Python. Cuarto, pruebas funcionales que aseguran la operación correcta de todos los componentes y el cumplimiento de los requisitos establecidos.

La herramienta propuesta representa una significativa contribución al ecosistema de desarrollo de aplicaciones basado en Python. En resumen, las capacidades de conversión y validación automática de PyGraphiX simplifican el desarrollo de GUI, reducen errores y mejoran la eficiencia. La conversión automática de unidades elimina la necesidad de cálculos manuales, mientras que la validación automática asegura que todos los elementos estén correctamente configurados y funcionales. Esto permite a los desarrolladores centrarse en aspectos más creativos y funcionales del diseño de interfaces, resultando en aplicaciones más robustas y estéticamente agradables. Además, la herramienta posee la capacidad de transformar el tradicional tedioso proceso de diseño de interfaces gráficas en una tarea más accesible y manejable. Al ofrecer una interfaz web intuitiva, los desarrolladores pueden interactuar visualmente con los componentes de Tkinter, configurar y observar los resultados en tiempo real. Esto no solo facilita la experimentación y el ajuste fino de las interfaces, sino que también reduce la barrera de entrada para los desarrolladores novatos.

Con PyGraphiX, los usuarios pueden seleccionar y personalizar elementos de Tkinter de manera visual, ajustando propiedades como color, tipo de letra, ancho y largo de forma intuitiva. Además, la herramienta integra diversas funcionalidades en un solo entorno cohesivo, permitiendo a los usuarios acceder a una biblioteca de componentes predefinidos, personalizarlos según sus necesidades, y ajustar sus propiedades visuales y funcionales sin necesidad de cambiar entre múltiples herramientas. Además, PyGraphiX permite la exportación directa del diseño a código Python,

facilitando su implementación en proyectos reales. Esto acelera el ciclo de desarrollo y permite una mayor flexibilidad y creatividad en la creación de interfaces gráficas.

A partir de este trabajo, surgen diversas posibilidades de investigación y expansión. Mejoras de diseño basadas en las mejoras prácticas de UX/UI (Experiencia de usuario/Interfaz de usuario) a través del aprendizaje automático. Consecuentemente, es necesario ampliar el soporte multiplataforma para incluir otros marcos de desarrollos de interfaces gráficas, tales como PyQt y Kivy. En esta línea, la gestión de interfaces complejas y con muchos componentes gráficos demanda un estudio de la optimización del rendimiento de la herramienta a través de nuevas técnicas de diseño y medidas de seguridad. En resumen, esta herramienta no solo facilita el diseño de interfaces gráficas en Python, sino que también abre la puerta a futuras innovaciones y mejoras, contribuyendo significativamente al campo del desarrollo de software.

5. Referencias

Antonova, V. M., Zakhir, B. M., & Kuznetsov, N. A. (2019). **Modeling of Graphs with Different Types of Reachability in Python**. *Journal of Communications Technology and Electronics*, *64*(12), 1464–1472. https://doi.org/10.1134/S1064226919120015

Attardi, J. (2020). **Introduction to CSS**. *Modern CSS*, 1–289. https://doi.org/10.1007/978-1-4842-6294-8

Bronshteyn, I. E. (2013). **Study of defects in a program code in python**. *Programming and Computer Software*, *39*(6), 279–284. https://doi.org/10.1134/S0361768813060017

Charatan, Q., & Kans, A. (2022). **Python Case Study**. In *Programming in Two Semesters, Texts in Computer Science,* (pp. 255–290). https://doi.org/10.1007/978-3-031-01326-3

Charatan, Q., Kans, A., & Javafx, F. (2019). **Python Graphics with Tkinter**. In *Texts in Computer Science Java in Two Semesters* (p. 719). Springer. https://doi.org/10.1007/978-3-031-01326-3

Dong Gao, B., Wang, X., Ou, D., & Gao, H. (2022). **Research and Design of Virtual Driving Platform of Simulated Train Based on Python**. *Proceedings of the 5th International Conference on Electrical Engineering and Information Technologies for Rail Transportation (EITRT) 2021. EITRT 2021. Lecture Notes in Electrical Engineering*, *1*, 235–242. https://doi.org/10.1007/978-981-16-9909-2

Elumalai, A. (2021). **Play with Letters and Words**. In *Introduction to Python for Kids: Learn Python the Fun Way by Completing Activities and Solving Puzzles* (pp. 1–552). Apress. https://doi.org/10.1007/978-1-4842-6812-4

Freeman, A. (2022). **Using Methods and Interfaces**. In *Pro Go* (pp. 273–307). Apress. https://doi.org/doi.org/10.1007/978-1-4842-7355-5_11

Gladstone, A. (2022). **C++ Software Interoperability for Windows Programmers**. In *Building a Python Extension Module* (pp. 147–172). Apress. https://doi.org/10.1007/978-1-4842-7966-3_7

Katricheva, N., Yaskevich, A., Lisitsina, A., Zhordaniya, T., Kutuzov, A., & Kuzmenko, E. (2020). **Vec2graph: A python library for visualizing word embeddings as graphs**. In *Communications in Computer and Information Science* (Vol. 1086CCIS). Springer International Publishing. https://doi.org/10.1007/978-3-030-39575-9_20

Kawamura, K., & Yamamoto, A. (2021). **HTML-LSTM: Information Extraction from HTML Tables in Web Pages Using Tree-Structured LSTM**. In L. Soares, C., Torgo (Ed.), *Lecture Notes in Computer Science: Vol. 12986 LNAI* (pp. 29–43). Springer International

Publishing. https://doi.org/10.1007/978-3-030-88942-5_3

Moruzzi, G. (2020). **Tkinter Graphics**. In *Essential Python for the Physicist* (pp. 1–302). Springer. https://doi.org/10.1007/978-3-030-45027-4

Sherathiya, V. N., Schaid, M. D., Seiler, J. L., Lopez, G. C., & Lerner, T. N. (2021). **GuPPy, a Python toolbox for the analysis of fiber photometry data**. *Scientific Reports*, *11*(1), 1–9. https://doi.org/10.1038/s41598-021-03626-9

Thesing, T., Feldmann, C., & Burchardt, M. (2021). **Agile versus Waterfall Project Management: Decision model for selecting the appropriate approach to a project.** *Procedia Computer Science*, *181*, 746–756. https://doi.org/10.1016/j.procs.2021.01.227

Willman, J. M. (2022). **Creating GUIs with Qt Designer**. In *Beginning PyQt* (pp. 217–258). Apress. https://doi.org/10.1007/978-1-4842-7999-1_8