Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2024

IPCART-STUDIO: CREACIÓN DE ARTE DIGITAL EN PÍXELES ; INTEGRACION A SISTEMA WEB

202200239 – Pedro Luis Avila

Resumen

IPCArt-Studio deja de ser una plataforma de escritorio diseñada para la gestión y creación de arte digital en píxeles y evoluciona para ser una página web moderna y más intuitiva . El presente ensayo describe el desarrollo de un sistema informático enfocado en la gestión y procesamiento de datos dispersos mediante una arquitectura modular. Este sistema utiliza Python como lenguaje principal y está diseñado para manejar usuarios, procesar imágenes y optimizar recursos mediante estructuras avanzadas como matrices dispersas. La implementación asegura escalabilidad, mantenibilidad y eficiencia

Palabras clave

pixel art, programación orientada a objetos, tipos abstractos de datos, Graphviz, plataformas interactivas.

Abstract

IPCArt-Studio ceases to be a desktop platform designed for the management and creation of digital pixel art and evolves to be a modern and more intuitive website. This essay describes the development of a computer system focused on the management and processing of sparse data through a modular architecture. This system uses Python as the main language and is designed to manage users, process images, and optimize resources using advanced structures such as sparse matrices. The implementation ensures scalability, maintainability, and efficiency

.

Keywords

pixel art, object-oriented programming, abstract data types, Graphviz, interactive platforms.

Introducción

El arte digital ha experimentado un auge significativo en las últimas décadas, destacándose el pixel art como una forma de expresión popular entre artistas y entusiastas. Sin embargo, gestionar y crear obras de arte en este formato requiere herramientas que optimicen recursos y fomenten la creatividad. IPCArt-Studio surge como una respuesta a esta necesidad, ofreciendo una plataforma que combina funcionalidades de gestión y edición con un enfoque en la eficiencia y colaboración.

En el ámbito del desarrollo de software, la organización de los componentes del sistema juega un papel crucial para garantizar la mantenibilidad, escalabilidad y eficiencia. Este ensayo analiza la estructura de un sistema de backend desarrollado en Python, detallando sus componentes principales y discutiendo cómo estos interactúan para cumplir con los requerimientos de la aplicación propuesta en el proyecto.

Desarrollo del tema

Objetivos

- Diseñar un sistema que administre usuarios y datos de manera eficiente.
- Implementar una estructura modular que separe responsabilidades en controladores, modelos y bases de datos.
- Optimizar el manejo de información mediante el uso de matrices dispersas.
- Proveer una arquitectura escalable y fácil de mantener.

Análisis de la Estructura del Sistema

La estructura del sistema presentada cuenta con cuatro directorios principales: `controllers`, `database`, `models` y `entradas`. A continuación, se describe el propósito de cada uno:

1. **Controllers:** Este directorio contiene la lógica de control del sistema, encargada de procesar las solicitudes y coordinar las operaciones entre los diferentes componentes. En este caso, se incluyen los archivos `imagenController.py` y `usuarioController.py`. Estos controladores manejan funcionalidades relacionadas con imágenes y usuarios, respectivamente.
2. **Database:** Este directorio está destinado a manejar la conexión y las operaciones relacionadas con la base de datos, tales como consultas, inserciones y actualizaciones.
3. **Models:** Este directorio contiene la representación de los objetos y estructuras de datos fundamentales del sistema. Por ejemplo, los archivos `Imagen.py`, `Pixel.py` y `Usuario.py` representan clases que modelan las entidades correspondientes. Además, la subcarpeta `matrizDispersa` implementa estructuras de datos avanzadas, posiblemente para optimizar el manejo de imágenes o datos dispersos.
4. **Entradas:** Este directorio podría almacenar datos de entrada, como archivos de configuración o recursos necesarios para el funcionamiento del sistema.

Finalmente, el archivo `mian.py` es el punto de entrada principal de la aplicación, donde se inicializan y coordinan los componentes previamente descritos.

Diagrama de Clases

El diagrama de clases permite visualizar las relaciones entre las diferentes clases y entender cómo interactúan entre sí. En este caso, las clases principales serían:

- **Imagen:** Representa los atributos y métodos relacionados con las imágenes.
- **Pixel:** Modela las propiedades de un pixel individual dentro de una imagen.

- **Usuario:** Define los datos y métodos asociados a los usuarios del sistema.
- **MatrizDispersa:** Una estructura de datos avanzada para almacenar información de forma eficiente.

Resultados Esperados

El sistema implementado debe permitir:

- Administrar usuarios y datos de manera eficiente.
- Procesar imágenes optimizando recursos computacionales.
- Almacenar y acceder a información dispersa mediante matrices especializadas.
- Facilitar futuras modificaciones gracias a su arquitectura modular.

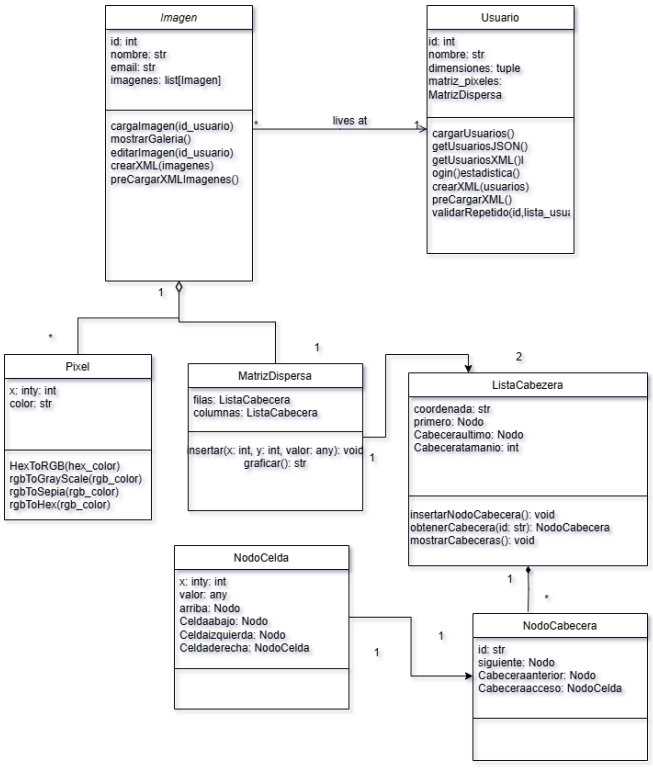

Figura 1. Diagrama de clases del funcionamiento del proyecto.

Fuente: elaboración propia, 2024.

Explicación:

Clases y Atributos

1. **Clase Imagen**
   - **Atributos:**
     - id (int): Identificador único de la imagen.
     - nombre (str): Nombre de la imagen.
     - dimensiones (tuple): Dimensiones (ancho, alto) de la imagen.
     - matriz_pixeles (MatrizDispersa): Referencia a una matriz dispersa que contiene los píxeles de la imagen.
   - **Relación:**
     - Asocia una matriz dispersa para gestionar la información de sus píxeles.
2. **Clase Pixel**
   - **Atributos:**
     - x (int): Coordenada X del píxel.
     - y (int): Coordenada Y del píxel.
     - color (str): Representación del color del píxel (por ejemplo, en formato hexadecimal).
   - **Relación:**
     - Cada píxel pertenece a una posición específica dentro de una imagen a través de la matriz dispersa.
3. **Clase Usuario**
   - **Atributos:**
     - id (int): Identificador único del usuario.

- nombre (str): Nombre del usuario.
- email (str): Dirección de correo electrónico del usuario.
- imagenes (list[Imagen]): Lista de imágenes asociadas al usuario.

- **Relación:**

Conclusiones

IPCArt-Studio representa una solución integral y eficiente para la gestión y creación de pixel art, destacándose por su aplicación de POO y TDA , servicios Api, gestión Backend y diseño Frontend.

La estructura del sistema refleja un enfoque modular y organizado, característico de buenas prácticas en el desarrollo de software. La correcta implementación y documentación de cada componente es esencial para garantizar su éxito. Además, el uso de patrones de diseño como la separación de responsabilidades (controladores, modelos y base de datos) mejora significativamente la calidad y mantenibilidad del código.

Referencias bibliográficas

1. Universidad de San Carlos de Guatemala. *Formato de Artículo Ensayo IPC2 2024*.
2. Universidad de San Carlos de Guatemala. *[IPC2] Enunciado Proyecto1 2024*.

Anexos

MÉTODO	DIRECCIÓN Y TIPO DE MÉTODO	BODY	RESPUESTA
Cargar Usuarios	`/usuarios/carga`, POST	Recibe un XML con los datos de los usuarios. Se valida cada usuario, si es correcto se añade al listado de usuarios.	Si todo está bien: `Usuarios cargados con éxito`, `status: 201`. Si hay errores: `Error al cargar los usuarios`, `status: 404`
Obtener Usuarios JSON	`/usuarios/json`, GET	Retorna un JSON con la lista de usuarios, incluyendo `id`, `nombre`, `correo`, `telefono`, `direccion` y `perfil`.	Si todo está bien: `usuarios`, `status: 200`. Si hay errores: `Error al obtener los usuarios`, `status: 404`
Obtener Usuarios XML	`/usuarios/xml`, GET	Retorna un XML con la lista de usuarios, incluyendo `id`, `pwd`, `nombre`, `correo`, `telefono`, `direccion` y `perfil`.	Si todo está bien: XML con la lista de usuarios, `status: 200`. Si hay errores: `Error al obtener los usuarios`, `status: 404`
Login	`/usuarios/login`, POST	Recibe un JSON con el `id` y `password` del usuario. Si las credenciales son correctas, se retorna un mensaje de bienvenida.	Si las credenciales son correctas: `message`: 'Bienvenido [nombre]', `status: 200`. Si no, `message`: 'Credenciales inválidas'
Estadísticas	`/usuarios/estadistica`, GET	Obtiene la cantidad de imágenes asociadas a cada usuario.	Si todo está bien: JSON con las estadísticas, `status: 200`. Si hay errores: `Error al obtener estadísticas`, `status: 404`
Cargar Imagen	`/imagenes/carga/:id_usuario`, POST	Recibe un XML con los datos de la imagen, valida los datos y la carga.	Si todo está bien: `Imagen cargada correctamente`, `status: 200`. Si hay errores: `Error al cargar la imagen`, `status: 404`
Mostrar Galería	`/imagenes/galeria`, GET	Retorna un JSON con la información de todas las imágenes, incluyendo `id`, `nombre`, `matriz`.	Si todo está bien: JSON con las imágenes, `status: 200`. Si hay errores: `Error al mostrar la galería`, `status: 404`
Editar Imagen	`/imagenes/editar/:id_usuario`, POST	Recibe un JSON con el `id` de la imagen y el filtro (1 para escala de grises, 2 para sepia). La imagen es editada y se retorna la nueva matriz.	Si todo está bien: `Imagen editada correctamente`, `status: 201`. Si hay errores: `Error al editar la imagen`, `status: 404`

Tabla 1. Elaboración propia 2024

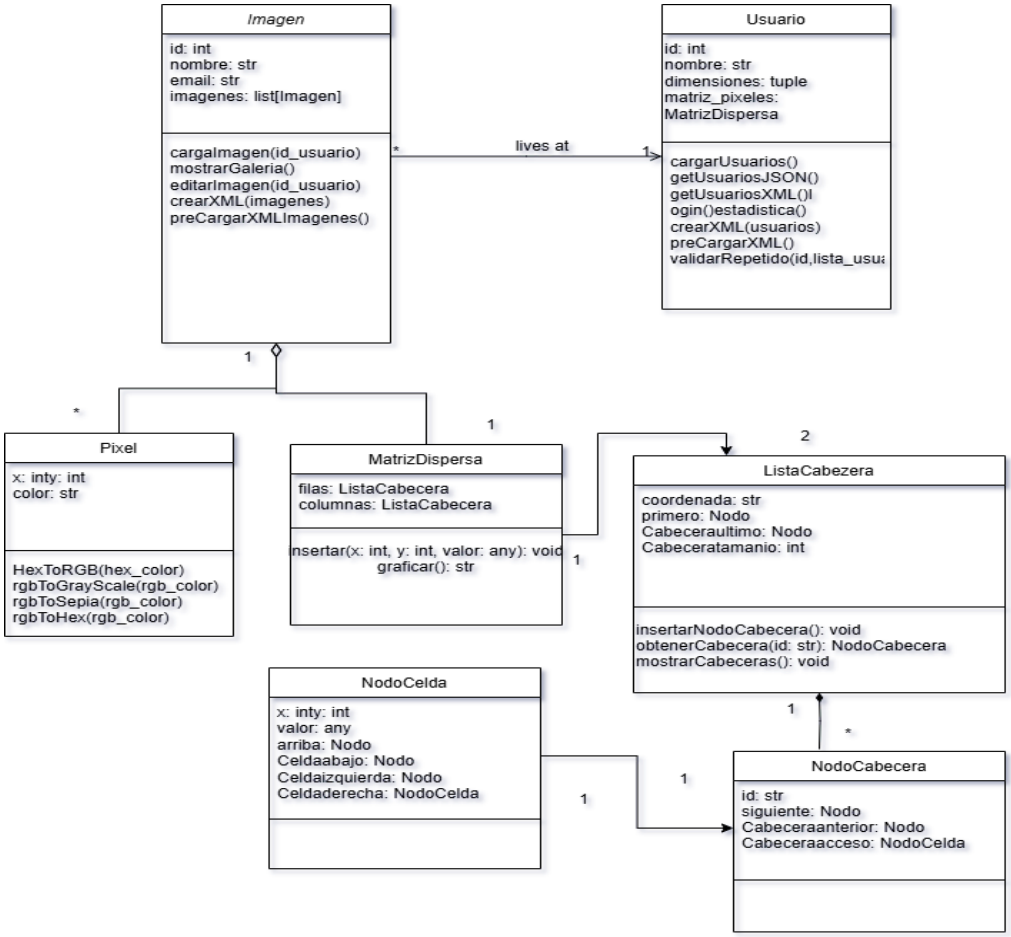

Figura 1. ampliación Diagrama de Clases, Elaboración propia,2024