Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, Vacaciones 2do semestre 2024

IPCArt – Studio Web

202300712 – Melvin Geovanni García Sumalá

Resumen

Este ensayo presenta el desarrollo de una solución integral para **IPCArt-Studio**, una plataforma innovadora que combina herramientas de creación de arte digital en píxeles con tecnologías avanzadas como APIs RESTful. Mediante una arquitectura cliente-servidor, el proyecto busca optimizar la gestión de datos y mejorar la experiencia de los usuarios a través de servicios accesibles desde cualquier navegador. El sistema emplea estructuras como matrices dispersas para almacenar imágenes y utiliza archivos XML como base de su persistencia, garantizando una solución moderna, eficiente y escalable.

Palabras clave

Pixel art, RESTful API, cliente-servidor, arquitectura REST, XML persistencia, matriz dispersa, Flask, Django.

Abstract

This essay presents the development of a comprehensive solution for **IPCArt-Studio**, an innovative platform that combines digital pixel art creation tools with advanced technologies such as RESTful APIs. Through a client-server architecture, the project seeks to optimize data management and enhance user experience via services accessible from any browser. The system employs structures like sparse matrices to store images and uses XML files as the basis for persistence, ensuring a modern, efficient, and scalable solution.

Keywords

Pixel art, RESTful API, client-server architecture, XML persistence, sparse matrix, Flask, Django.

Introducción

IPCArt-Studio es una plataforma destinada a revolucionar la creación de arte en píxeles, atendiendo las necesidades de artistas y entusiastas del arte digital. Este proyecto transforma una aplicación de escritorio tradicional en un sistema web moderno que aprovecha tecnologías como Flask para el backend y Django para el frontend.

En su ámbito técnico, **IPCArt-Studio** implementa principios de programación orientada a objetos y estructuras de datos avanzadas. Las imágenes se gestionan mediante matrices dispersas, lo que permite optimizar recursos y agilizar operaciones. El almacenamiento se realiza exclusivamente en XML, evitando el uso de bases de datos relacionales. Además, se ofrecen servicios como carga y edición de imágenes, visualización de galerías, y generación de reportes con estadísticas dinámicas.

Desarrollo del tema

El sistema **IPCArt-Studio** se organiza en una arquitectura cliente-servidor, integrando un backend desarrollado con Flask y un frontend con Django. Esta distribución garantiza escalabilidad, accesibilidad y separación de responsabilidades. El desarrollo se estructura en los siguientes módulos principales:

Backend: API en Flask

1. **Servicios RESTful:** La API proporciona endpoints como:
    - `/cargarUsuarios`: Procesa archivos XML con datos de usuarios.
    - `/cargarImagen`: Carga y transforma imágenes a formatos procesables.
    - `/login`: Valida credenciales de administrador y usuarios.
    - `/imagenes`: Devuelve una galería de imágenes procesadas.
2. **Persistencia XML:** Todos los datos se almacenan en archivos XML organizados jerárquicamente, garantizando portabilidad y legibilidad.
3. **Procesamiento de Imágenes:** Utiliza matrices dispersas para optimizar la manipulación de imágenes.

Frontend: Aplicación en Django

1. **Carga de Archivos:** Los usuarios pueden cargar y previsualizar XML antes de enviarlos al servidor.
2. **Visualización de Galerías:** Imágenes procesadas se muestran junto con sus metadatos.
3. **Estadísticas Dinámicas:** Gráficos interactivos generados con Plotly permiten analizar datos clave, como:
    - Usuarios con mayor cantidad de imágenes cargadas.
    - Número de ediciones por usuario.

Estructuras de Datos

1. **Matrices Dispersas:** Representan imágenes almacenando solo los píxeles necesarios, reduciendo el uso de memoria.
2. **Listas y Pilas:** Facilitan la gestión de usuarios, solicitudes y galerías.

Tecnologías Implementadas

1. **Flask:** Framework ligero para construir la API RESTful.
2. **Django:** Plataforma robusta para el desarrollo del frontend.
3. **Graphviz:** Visualización de estructuras como listas enlazadas y matrices dispersas.
4. **Plotly:** Generación de estadísticas visuales atractivas e interactivas.

Conclusiones

El proyecto **IPCArt-Studio** representa un avance significativo en la gestión de arte digital mediante herramientas modernas y eficientes. La transición a una arquitectura cliente-servidor asegura accesibilidad global y escalabilidad futura. El uso de estructuras de datos como matrices dispersas y la integración de APIs RESTful proporcionan una solución robusta y adaptable a las necesidades actuales del mercado.

El enfoque modular del sistema permite una fácil extensión, mientras que la interfaz desarrollada en Django mejora la experiencia de usuario. Finalmente, el almacenamiento en XML asegura que los datos sean portables y mantenibles sin depender de bases de datos relacionales.

Tabla de endpoints

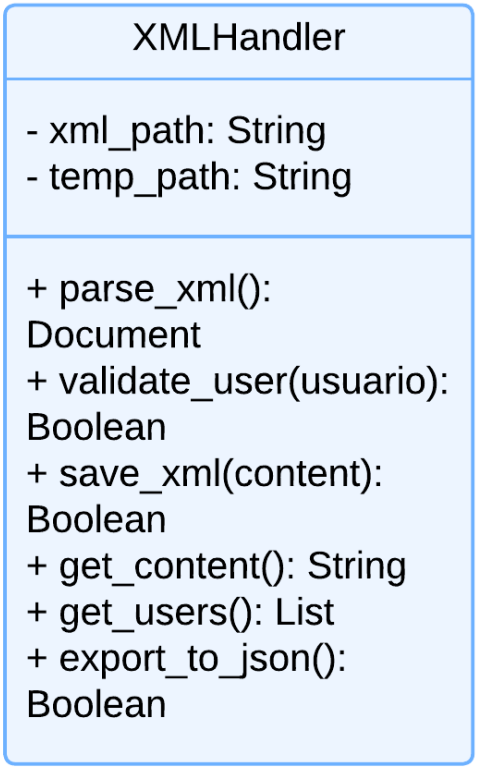
Flask URLS´s (Backend)

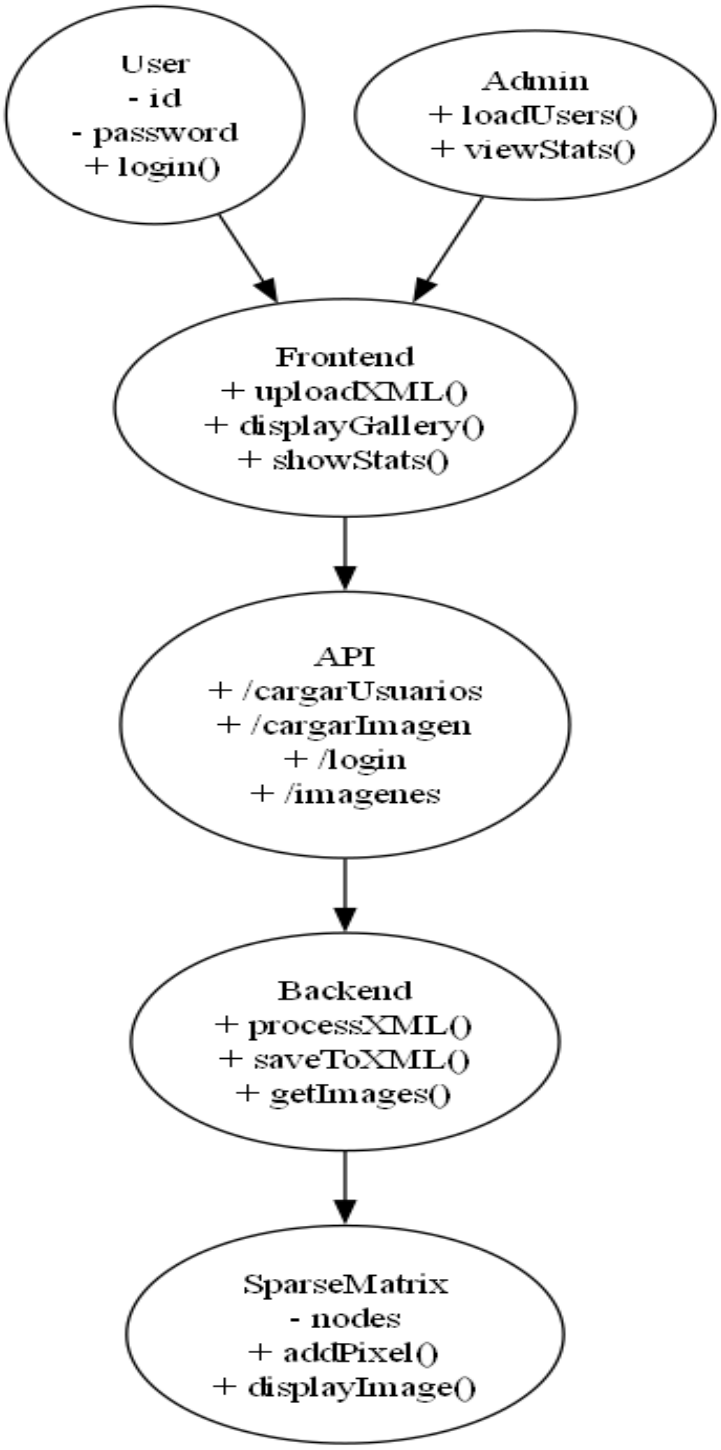
- Método: POST
  Endpoint**:** /login
  Descripción**:** Autenticación de usuarios. Acepta "AdminIPC"/"ARTIPC2" o usuarios del XML.
- Método: POST
  Endpoint**:** /upload
  Descripción**:** Carga y valida archivo XML de usuarios. Filtra usuarios válidos.
- Método: GET
  Endpoint**:** /view-xml
  Descripción**:** Retorna el contenido del XML de usuarios cargado.
- Método: GET
  Endpoint**:** /api/get-users
  Descripción**:** Obtiene lista de usuarios en formato JSON desde el XML.
- Método: GET
  Endpoint**:** /api/users
  Descripción**:** Obtiene todos los usuarios (similar a get-users).
- Método: POST
  Endpoint**:** /api/users
  Descripción**:** Crea un nuevo usuario (validando formato).
- Método: PUT
  Endpoint**:** /api/users
  Descripción**:** Actualiza información de usuario existente.
- Método: DELETE
  Endpoint**:** /api/users
  Descripción**:** Elimina un usuario por ID.
- Método: GET
  Endpoint**:** /api/xml
  Descripción**:** Obtiene el contenido del XML actual.
- Método: POST
  Endpoint**:** /api/xml
  Descripción**:** Carga un nuevo archivo XML.
- Método: GET
  Endpoint**:** /api/xml/users
  Descripción: Obtiene el XML de usuarios en formato raw.

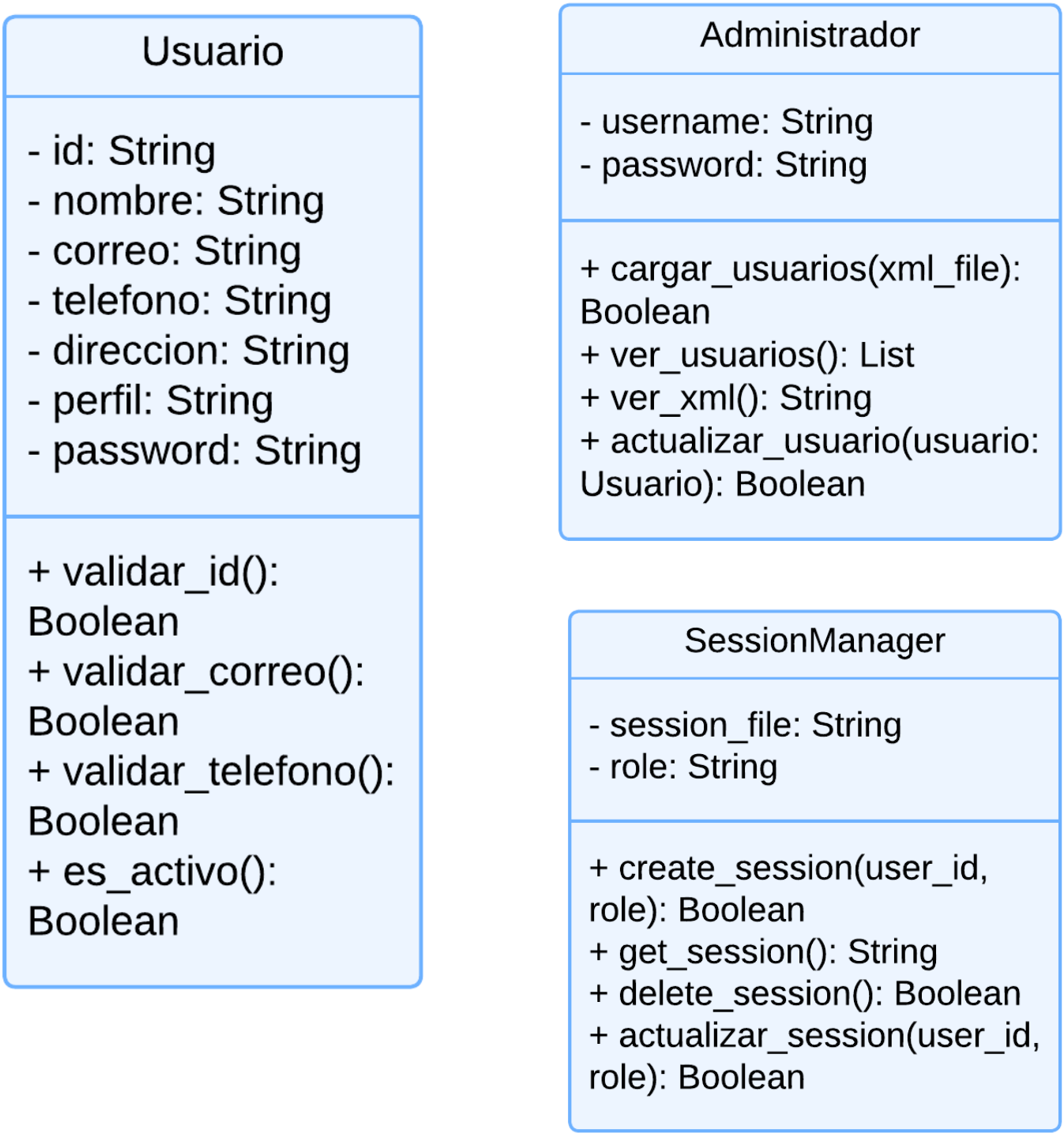
Django URL´s (Frontend)

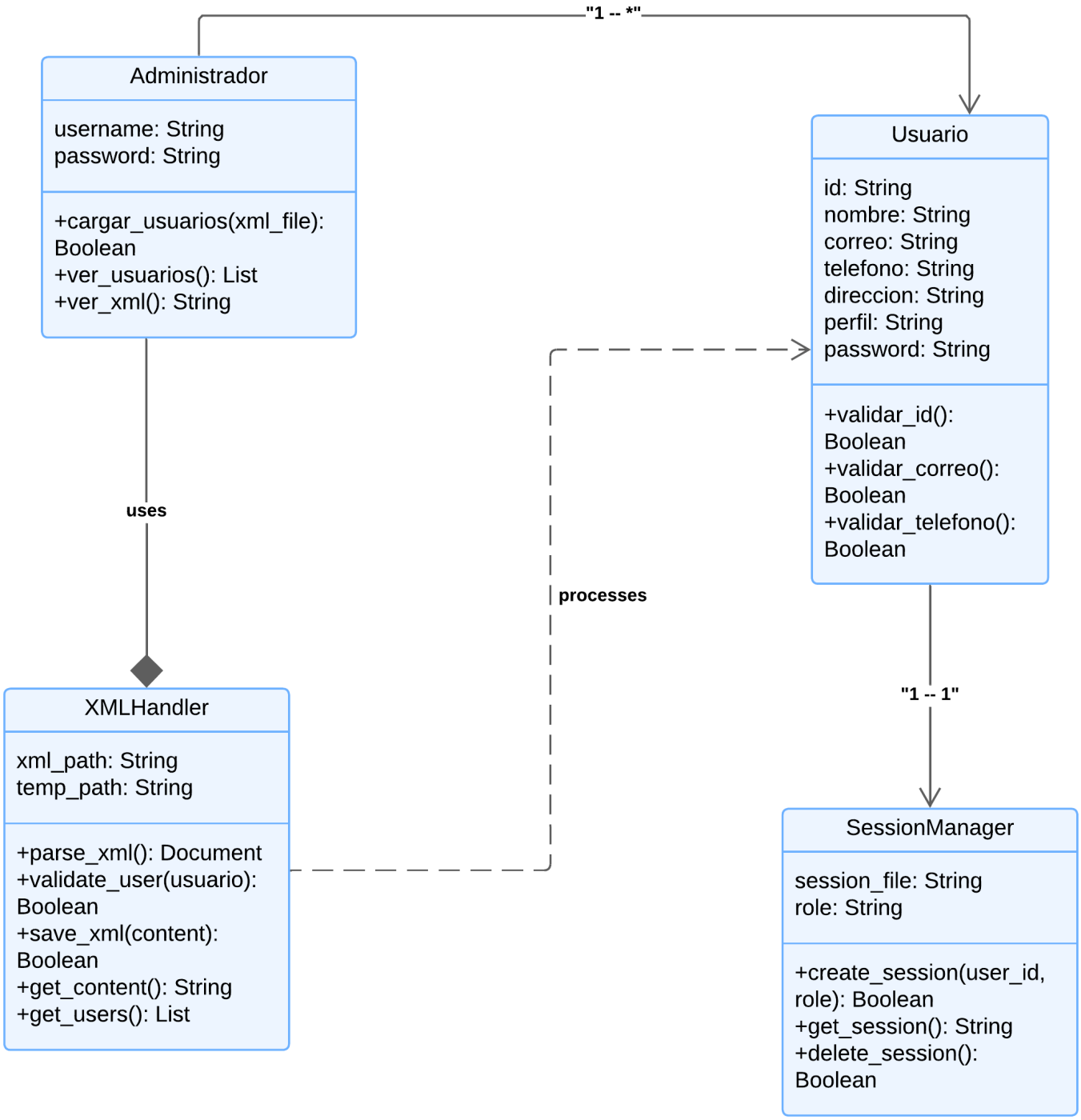
- Método: GET/POST
  Endpoint**:** /
  Descripción**:** Página de login.
- Método: GET
  Endpoint**:** /logout/
  Descripción**:** Cierra sesión actual.
- Método: GET/POST
  Endpoint**:** /CargarUsuarios/
  Descripción**:** Vista para admin: carga de archivos XML.
- Método: GET
  Endpoint**:** /Galeria/
  Descripción**:** Vista para usuarios: galería de imágenes.
- Método: GET
  Endpoint**:** /VerUsuarios/
  Descripción**:** Vista para admin: lista de usuarios.
- Método: GET
  Endpoint**:** /VerXML/
  Descripción**:** Vista para admin: contenido XML actual.
- Método: GET
  Endpoint**:** /CargarImagen/
  Descripción**:** Vista para usuarios: subir imagen.
- Método: GET
  Endpoint**:** /EditarImagen/
  Descripción**:** Vista para usuarios: editar imagen.
- Método: GET
  Endpoint**:** /Ayuda/
  Descripción**:** Vista de ayuda y documentación.

Anexos


XMLHandler
- xml_path: String
- temp_path: String
+ parse_xml(): Document
+ validate_user(usuario): Boolean
+ save_xml(content): Boolean
+ get_content(): String
+ get_users(): List
+ export_to_json(): Boolean


User
- id
- password
+ login()
Admin
+ loadUsers()
+ viewStats()
Frontend
+ uploadXML()
+ displayGallery()
+ showStats()
API
+ /cargarUsuarios
+ /cargarImagen
+ /login
+ /imagenes
Backend
+ processXML()
+ saveToXML()
+ getImages()
SparseMatrix
- nodes
+ addPixel()
+ displayImage()


Usuario
- id: String
- nombre: String
- correo: String
- telefono: String
- direccion: String
- perfil: String
- password: String
+ validar_id(): Boolean
+ validar_correo(): Boolean
+ validar_telefono(): Boolean
+ es_activo(): Boolean
Administrador
- username: String
- password: String
+ cargar_usuarios(xml_file): Boolean
+ ver_usuarios(): List
+ ver_xml(): String
+ actualizar_usuario(usuario: Usuario): Boolean
SessionManager
- session_file: String
- role: String
+ create_session(user_id, role): Boolean
+ get_session(): String
+ delete_session(): Boolean
+ actualizar_session(user_id, role): Boolean


"1 -- *"
Administrador
username: String
password: String
+cargar_usuarios(xml_file): Boolean
+ver_usuarios(): List
+ver_xml(): String
Usuario
id: String
nombre: String
correo: String
telefono: String
direccion: String
perfil: String
password: String
+validar_id(): Boolean
+validar_correo(): Boolean
+validar_telefono(): Boolean
uses
processes
"1 -- 1"
XMLHandler
xml_path: String
temp_path: String
+parse_xml(): Document
+validate_user(usuario): Boolean
+save_xml(content): Boolean
+get_content(): String
+get_users(): List
SessionManager
session_file: String
role: String
+create_session(user_id, role): Boolean
+get_session(): String
+delete_session(): Boolean