**Documentación del Proyecto: Plataforma Educativa**

**Introducción**

Este documento detalla el proceso de desarrollo de una plataforma educativa básica, desde la concepción de las clases principales hasta la implementación de funcionalidades complejas como reportes detallados y la integración con un sistema de control de versiones como Git. El objetivo principal era construir una herramienta que permitiera gestionar usuarios (estudiantes e instructores), cursos, evaluaciones y calificaciones, con la capacidad de generar reportes de rendimiento.

A lo largo del desarrollo, nos enfrentamos a diversos desafíos y oportunidades de aprendizaje, tanto en la lógica de programación como en el manejo de herramientas de colaboración.

**Fases del Desarrollo y Funcionalidades Implementadas**

**1. Diseño Inicial de Clases y Herencia**

Comencé el proyecto definiendo la estructura fundamental de la plataforma utilizando la Programación Orientada a Objetos (POO).

* **Clase Usuario**: La base para cualquier persona que interactúe con el sistema. Contiene atributos generales como id\_usuario, nombre y correo. Implementé un método mostrar\_info() para presentar la información básica.
* **Clases Estudiante e Instructor**: Estas clases heredan de Usuario y añaden atributos específicos (carnet para estudiantes, departamento para instructores). Modifiqué el constructor para extraer estos atributos de kwargs antes de pasarlos a la clase padre, lo que me pareció una forma elegante de manejar la inicialización.
* **Clase Evaluacion**: Representa cualquier tipo de evaluación. Contiene id\_eval, titulo, ponderacion (inicialmente peso, luego renombrado para mayor claridad) y un diccionario para calificaciones.
* **Clases Examen y Tarea**: Heredan de Evaluacion, demostrando polimorfismo, aunque en este diseño inicial no agregaron atributos o métodos únicos, sentaron las bases para posibles extensiones futuras.
* **Clase Curso**: Agrupa evaluaciones e estudiantes, y está asociada a un Instructor. Incluye métodos para inscribir\_estudiante y agregar\_evaluacion.
* **Clase Plataforma**: La clase central que orquesta todo el sistema, manteniendo diccionarios de usuarios y cursos. Contiene la lógica para registrar usuarios, crear cursos, inscribir estudiantes, etc.

**2. Implementación de la Lógica del Negocio**

Una vez definidas las estructuras, me enfoqué en la funcionalidad.

* **registrar\_usuario()**: Esta fue una de las funciones más críticas. Implementé validaciones para asegurar la integridad de los datos:
  + Formato de correo electrónico (@ y . presentes).
  + El ID de usuario debe ser numérico y único en el sistema.
  + El carnet del estudiante también debe ser numérico. Para la validación numérica, inicialmente pensé en isdigit(), pero luego opté por un enfoque más robusto con bloques try-except al intentar convertir a int(), lo que captura mejor diversos errores de formato.
* **calcular\_promedio\_final() en Curso**: Una función esencial para el rendimiento académico. Suma las notas ponderadas de cada evaluación y divide por la suma total de las ponderaciones. Tuve que asegurarme de manejar el caso de total\_ponderacion igual a cero para evitar divisiones por cero.

**3. Desarrollo de Reportes Avanzados**

Los reportes fueron un punto clave para obtener una visión clara del rendimiento académico.

* **generar\_reporte\_promedio\_bajo()**: Identifica a estudiantes con un promedio final por debajo de un umbral (65 en este caso).
* **generar\_reporte\_general()**: Muestra el promedio final de todos los estudiantes en cada curso.

El mayor desafío aquí fue la necesidad de **mostrar un desglose detallado de las notas** para entender el promedio de cada estudiante. Modifiqué ambos reportes para iterar sobre las evaluaciones de cada curso y mostrar la nota obtenida por el estudiante en cada una, incluyendo un estado ("Entregada"/"No entregada") y el porcentaje de ponderación. Esto me permitió ver por qué un estudiante tenía cierto promedio.

**4. Interfaz de Usuario y Menú**

Creé un sencillo menú basado en la terminal (menu() función) para interactuar con todas las funcionalidades de la plataforma. Esto me permitió probar cada componente a medida que lo desarrollaba. Incluí manejo básico de ValueError para entradas numéricas.

**5. Refactorización y Claridad del Código**

A lo largo del proceso, realicé varias refactorizaciones. El cambio de peso a ponderacion en las clases Evaluacion y Curso, así como en las funciones de la Plataforma, fue un cambio puramente semántico que mejoró enormemente la claridad del código, haciéndolo más intuitivo para cualquiera que lo lea.

**Gestión de Versiones con Git y GitHub**

La colaboración con un compañero y la necesidad de una exposición nos llevó a utilizar Git y GitHub para gestionar el código. El plan era dividir las funciones, que cada uno trabajara en su rama y luego fusionar los cambios. Sin embargo, me encontré con algunos desafíos iniciales en la configuración.

**Explicar las decisiones de diseño tomadas**

Se decidió usar programación orientada a objetos para organizar el sistema en clases como usuario, estudiante, instructor, curso, evaluación, examen y tarea. esto permitió tener el código más ordenado, reutilizable y fácil de mantener. además, se incluyó un menú principal para que el programa sea interactivo.

**¿Cómo manejaron la herencia y el polimorfismo en su diseño? ¿qué ventajas les brindó?**

* Se aplicó herencia en estudiante e instructor que heredan de usuario, y también en examen y tarea que heredan de evaluación.
* Se usó polimorfismo en el método mostrar\_info, ya que cada clase lo implementa de manera distinta (un estudiante muestra su carné y un instructor su departamento).
* La ventaja fue evitar código duplicado y poder trabajar de forma general con objetos que se comportan diferente según el tipo.

**¿Qué estructuras de datos fueron más útiles y por qué?**

* se usaron diccionarios (usuarios, cursos, calificaciones) porque permiten acceder rápido a la información mediante claves como id de usuario o código de curso.
* se usaron listas (estudiantes, evaluaciones) porque permiten guardar elementos en orden y recorrerlos fácilmente.

**¿Qué errores comunes anticiparon y cómo los controlaron en el código?**

* correos inválidos → se validó que tengan @ y id de usuario repetidos o no numéricos → se usó try-except y validaciones.
* carné no numérico → también controlado con try-except.
* notas fuera de rango → se controla lanzando error si no está entre 0 y 100.
* intentos de inscribir usuarios que no son estudiantes → se muestra mensaje de error.
* esto ayudó a que el programa no se caiga y dé mensajes claros al usuario.

**¿Cómo organizaron su trabajo usando git y github?**

* se creó un repositorio en github con ramas separadas:
* rama principal (main) para la versión estable.
* ramas de desarrollo para probar nuevas funciones (ej. registro-usuarios, reportes).
* se hicieron commits frecuentes con mensajes claros.
* se usaron pull requests para integrar cambios después de revisarlos.

**¿Qué mejoras harían si tuvieran más tiempo?**

* crear interfaz gráfica en tkinter para hacerlo más amigable.
* conectar con una base de datos para guardar información en lugar de usar solo memoria.
* permitir exportar los reportes en pdf o excel.
* mejorar la seguridad del sistema de usuarios con contraseñas.

**Errores y sus soluciones**

| Errores Identificados | Soluciones Aplicadas |
| --- | --- |
| **Error de Sintaxis #1: Falta de paréntesis en print** En la clase Plataforma, en el método registrar\_usuario, la línea print "Tipo de usuario inválido." generaba un error SyntaxError. **Descripción del problema:** Se olvidó el uso de paréntesis alrededor de la cadena de texto, lo cual es necesario en Python 3 para la función print. | **Corrección de Sintaxis** Se corrigió la línea para que funcionara correctamente en Python 3, agregando los paréntesis necesarios. La línea quedó de la siguiente manera: print("Tipo de usuario inválido.") |
| **Error Lógico #1: Cálculo de ponderación incorrecto** En el método calcular\_promedio\_final de la clase Curso, el cálculo total\_promedio\_ponderado += evaluacion.calcular\_nota\_ponderada(id\_estudiante) devolvía valores extraños. **Descripción del problema:** La función calcular\_nota\_ponderada ya estaba devolviendo una nota multiplicada por la ponderación. Al sumarla en el bucle, se duplicaba la ponderación, lo que resultaba en un promedio final incorrecto y exageradamente alto. | **Ajuste Lógico en la Fórmula** Se revisó el código y se encontró que la fórmula de cálculo estaba duplicada. Se modificó el método calcular\_nota\_final para que solo devolviera la nota sin ponderar, y la ponderación se realizó en el método calcular\_promedio\_final para evitar la doble multiplicación. Se agregó una variable temporal para el cálculo y se ajustaron los nombres de las variables para mayor claridad. |
| **Error de Sintaxis #2: Atributo mal llamado** En la clase Estudiante, el método \_\_init\_\_ generaba un AttributeError. **Descripción del problema:** Se detectó que el nombre de la variable para el carnet no coincidía en todas las partes del código, causando una inconsistencia. En la línea self.carnet = kwargs.pop('carnet', None), la palabra carnet se había escrito incorrectamente como carne. | **Renombrado de Atributo** Se revisó la ortografía del atributo en todas las líneas de código relacionadas. Se corrigió el nombre de la variable de carne a **carnet** en la línea de inicialización, asegurando que el atributo fuera consistente en todo el programa. |
| **Error Lógico #2: Condicional de tipo incorrecto** En el método registrar\_usuario, la validación de if tipo == estudiante: no funcionaba como se esperaba y permitía el registro de cualquier tipo de usuario. **Descripción del problema:** El código comparaba la variable tipo (una cadena de texto) con el nombre de una clase (el objeto estudiante), lo cual siempre resultaba en False y hacía que el if fuera ineficaz. La lógica correcta era comparar la variable tipo con la cadena literal "estudiante". | **Corrección de la Condición** Se ajustó la condición para comparar el tipo de usuario con la cadena correcta. Se encerró el texto "estudiante" entre comillas para que Python lo reconociera como una cadena de texto, y se modificó la línea a if tipo == "estudiante":. También se agregó .lower() al input para hacer la validación insensible a mayúsculas. |
| **Error de Lógica #3: Fallo en la iteración** En la función menu, al registrar un usuario instructor, el código no solicitaba el departamento. **Descripción del problema:** El código para solicitar el departamento estaba dentro de la condición if tipo == "estudiante". Esto significaba que la línea nunca se ejecutaba para los usuarios de tipo "instructor", causando que el atributo departamento se pasara como None. | **Reordenamiento de la Lógica del Menú** Se movió la lógica de entrada de datos para carnet y departamento fuera de las condiciones if principales, asegurando que se soliciten los datos correctos para cada tipo de usuario antes de llamar a la función de registro. Se añadió un elif para manejar el caso del instructor de manera independiente. |

**Errores en Git Hub**

**Tabla de Errores y Soluciones de Git y GitHub**

| Errores Identificados | Soluciones Aplicadas |
| --- | --- |
| **Problema #1: Error al subir cambios a la rama principal (main)** Al intentar ejecutar git push origin main, se obtuvo un error que indicaba que la rama remota estaba adelantada y que los cambios habían sido rechazados. **Diagnóstico del problema:** Esto sucede porque el historial del repositorio local no está sincronizado con el historial del repositorio remoto en GitHub. Otro colaborador ya había subido cambios y tu rama local no los incluía, lo que generaba un conflicto potencial. | **Flujo de trabajo ajustado** Para corregir esto, se ejecutó git pull origin main para traer y fusionar los cambios más recientes del repositorio remoto. Luego, se volvió a intentar el push (git push origin main), lo que permitió que los cambios se subieran correctamente una vez que el historial de ambas ramas estuvo unificado. |
| **Problema #2: Conflicto al fusionar (merge) ramas** Al intentar fusionar una rama de trabajo (feature-login) a la rama principal (main) con git merge feature-login, Git reportó un conflicto de fusión. **Diagnóstico del problema:** Este error ocurre cuando ambos colaboradores modifican las mismas líneas de código en el mismo archivo. Git no puede decidir qué versión mantener y necesita una intervención manual para resolver el conflicto. | **Resolución manual de conflictos** Se abrió el archivo en conflicto y se localizaron los marcadores de Git (<<<<<<<, =======, >>>>>>>) que indicaban las diferencias. Se editó el código para combinar las versiones deseadas, eliminando los marcadores. Finalmente, se ejecutó git add [nombre\_del\_archivo] y luego git commit -m "Resolución de conflicto de merge" para finalizar la fusión. |
| **Problema #3: No se pudo clonar el repositorio** Un compañero intentó clonar el repositorio remoto usando git clone y recibió el error fatal: repository not found. **Diagnóstico del problema:** Este problema suele deberse a un error en la URL del repositorio, ya sea por un error tipográfico, una URL incompleta o un repositorio que no es público. | **Verificación de la URL del repositorio** Se verificó la URL directamente en la página de GitHub, asegurándose de que estuviera completa y fuera correcta. La URL HTTPS se copió de nuevo desde la página principal del repositorio para evitar errores de escritura y se utilizó en el comando git clone, resolviendo el problema. |
| **Problema #4: "Commit" sin mensaje en Vim** Al ejecutar un comando que requería un mensaje de "commit", se abrió una ventana del editor de texto Vim, y no se pudo salir. **Diagnóstico del problema:** Esto ocurre cuando Git necesita una descripción de un "commit" y se abre el editor por defecto (Vim). La falta de familiaridad con los comandos básicos del editor puede impedir que se guarde el mensaje o se cierre la ventana. | **Guardar y salir del editor** Se le explicó cómo usar los comandos básicos de Vim. Para salir y guardar el mensaje predeterminado, se le indicó presionar la tecla Esc y luego escribir :wq seguido de Enter. Esto guardó el mensaje y completó la operación de Git, permitiendo continuar con el flujo de trabajo. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Integrante** | **Carné** | **Tarea asignada** | **Evidencia en Git Hub** | **Fecha de entrega parcial** | **Validación docente** |
| Davis José Alexis García López | 1554725 | Menú interactivo  Clase Usuario  Clase estudiante  Clase instructor | Commits subidos previamente a Git Hub | 17 de septiembre del 2025 |  |
| Juan Pablo Sánchez Santiagos | 1554925 | Clase plataforma  Clase evaluación  Clase tarea  Clase curso | Commits subidos previamente a Git Hub | 17 de septiembre del 2025 |  |