República Bolivariana de Venezuela
Ministerio del Poder Popular para la Educación Universitaria
Universidad Nacional Experimental de las Telecomunicaciones e Informática
Ingeniería en Informática
Asignatura: Programación
Docente: Profesor Carlos Márquez

INFORME PROYECTO FINAL

REALIZADO POR:
SOFÍA ARTEAGA HIGUERA 31.149.998
https://github.com/kingsophh/mi_calificador_tkinker.git

Resumen

El presente informe detalla el desarrollo de una aplicación de escritorio para la gestión de calificaciones personales. El objetivo principal fue crear una herramienta funcional que permitiera a los estudiantes organizar sus notas, calcular promedios de asignaturas y obtener un promedio general de forma automática. La metodología de desarrollo implicó la exploración inicial de frameworks como Flask y PyQt, lo que llevó a la identificación de desafíos técnicos y de configuración. Ante estas dificultades, se adoptó un enfoque estratégico para migrar el proyecto a Tkinter, aprovechando su simplicidad y confiabilidad para construir un sistema robusto. El resultado es una aplicación CRUD completamente funcional, con un sistema seguro de registro e inicio de sesión, y la capacidad de importar y exportar datos de calificaciones a archivos de Excel. Este proyecto no solo cumple con los requisitos funcionales, sino que también sirve como una demostración de la capacidad de adaptación y el proceso de aprendizaje continuo en la programación.

Contenido

Introducción

El presente informe detalla el desarrollo de una aplicación de escritorio para la gestión de calificaciones personales, creada con el lenguaje de programación Python. El objetivo fundamental de este proyecto fue diseñar y construir una herramienta funcional y accesible que permitiera a los estudiantes organizar sus notas de forma eficiente, superando las limitaciones de los métodos manuales y las hojas de cálculo tradicionales. La necesidad de un sistema que no solo almacene datos, sino que también realice cálculos automáticos y proporcione una visión clara del rendimiento académico, sirvió como el principal motor para el inicio de este trabajo.

El proceso de desarrollo del proyecto representó un aprendizaje significativo en cuanto a la selección de herramientas y la adaptabilidad metodológica. Inicialmente, se consideró el uso de frameworks de desarrollo web como **Flask** y de interfaces de escritorio avanzadas como **PyQt**. Aunque estas plataformas demostraron ser robustas y versátiles, se optó por migrar el proyecto a **Tkinter**. Esta decisión se basó en la necesidad de encontrar un equilibrio entre la complejidad técnica y la rapidez de implementación, permitiendo una concentración total en la lógica de negocio del sistema. La elección final de **Tkinter** como principal biblioteca de interfaz, en conjunto con **SQLite** para la gestión de datos persistentes y **Bcrypt** para la seguridad de los usuarios, estableció una base sólida y confiable.

El resultado es una aplicación CRUD (Crear, Leer, Actualizar, Eliminar) que se distingue por su funcionalidad integral y su enfoque en la experiencia del usuario. Las características principales incluyen un sistema seguro de registro e inicio de sesión, la capacidad de ingresar calificaciones detalladas por asignatura y actividad, el cálculo automático de promedios ponderados y la gestión eficiente de la información académica. Además, se implementó la capacidad de importar y exportar datos a archivos de Excel, lo que dota a la aplicación de una

flexibilidad crucial. El presente informe documentará en detalle cada fase de este proyecto, desde la conceptualización hasta la implementación final, destacando los desafíos superados y las lecciones clave aprendidas en el camino.

Desarrollo

1. La Fase Inicial: La Búsqueda y los Primeros Desafíos

El inicio de este proyecto estuvo marcado por una fase de exploración metodológica y tecnológica. Con el objetivo de construir un gestor de calificaciones robusto y con una interfaz gráfica completa, se consideraron inicialmente dos marcos de trabajo potentes: **Flask** para una arquitectura web y **PyQt** para una aplicación de escritorio más compleja. La versatilidad de estas herramientas era prometedora, pero rápidamente surgieron desafíos que consumieron un tiempo significativo. La configuración de bases de datos, la gestión de dependencias y la integración de la lógica de negocio con la interfaz de usuario presentaron una curva de aprendizaje pronunciada, generando errores recurrentes como OperationalError y AttributeError que requerían una depuración exhaustiva. Esta etapa, aunque frustrante, fue fundamental para comprender la importancia de la elección de herramientas adecuada y la complejidad inherente a la integración de sistemas.

2. Un Cambio de Rumbo: La Adaptabilidad como Herramienta

Tras evaluar el progreso y las dificultades encontradas, se tomó la decisión estratégica de pivotar hacia una solución más sencilla y directa. Se optó por **Tkinter**, la biblioteca estándar de Python para interfaces gráficas de usuario. Esta elección no se debió a un abandono del proyecto, sino a un acto de adaptabilidad: al elegir una herramienta con una curva de aprendizaje menos empinada, se liberó tiempo y recursos mentales para concentrarse en la lógica central del gestor de calificaciones. Este cambio permitió dejar de lado los problemas de configuración y enfocar el esfuerzo en la implementación de las funcionalidades clave, demostrando que una solución

óptima no siempre es la más compleja, sino la que mejor se adapta a las circunstancias y objetivos.

3. Un Asistente en el Viaje: El Rol de Gemini Pro

El desarrollo del proyecto no fue un proceso solitario, sino un viaje asistido por una herramienta tecnológica de vanguardia. Basándome en que una persona cercana ofreció su ayuda y una alternativa para la programación, se tomó la decisión personal de apoyarme en las capacidades de **Gemini Pro**. Esta elección se basó en la fiabilidad y la inmensa capacidad del modelo para proporcionar asistencia en tiempo real, desde la resolución de errores de sintaxis hasta la sugerencia de arquitecturas de código. La comodidad de tener un asistente de programación siempre disponible, con la capacidad de entender el contexto y generar soluciones coherentes, fue un factor determinante para superar los obstáculos técnicos y avanzar de forma constante. Este apoyo fue crucial para mantener la motivación y para aprender a depurar y refactorizar el código de manera más eficiente, evidenciando que las herramientas de IA pueden ser aliadas poderosas en el proceso educativo.

4. La Construcción Final: Implementación y Funcionalidades Clave

Con la metodología clara, se procedió a la implementación final. El gestor de calificaciones se construyó con un diseño modular que incluye las siguientes características:

- **Sistema de autenticación**: Se implementó un sistema seguro de registro e inicio de sesión utilizando **Bcrypt** para encriptar las contraseñas, garantizando la privacidad de los datos.

- **Gestión de calificaciones (CRUD)**: La aplicación permite a los usuarios **C**rear, **L**eer, **A**ctualizar y **E**liminar calificaciones, asociadas a su usuario específico, lo que asegura que la información es personal y privada.
- **Cálculo de promedios**: Se desarrolló la lógica para calcular de forma automática el promedio ponderado por materia y un promedio general, proporcionando una visión instantánea del rendimiento académico.
- **Integración con Excel**: Gracias al uso de la librería **Pandas**, la aplicación ofrece la funcionalidad de **importar y exportar** datos de calificaciones en formato .xlsx, lo que añade una capa de flexibilidad y portabilidad al sistema.

5. Reflexión sobre las Dificultades Técnicas

Las dificultades técnicas inherentes a la programación se convirtieron en la parte más valiosa del proceso. Cada error de sintaxis, cada fallo en la lógica de la base de datos o cada problema de interfaz se tradujo en una lección de resiliencia y paciencia. El proyecto me enseñó que la depuración no es solo la corrección de errores, sino una habilidad fundamental para el desarrollo. Cada obstáculo superado fue una validación de la capacidad de investigar, adaptar y encontrar soluciones, fortaleciendo mi confianza como desarrollador.

Conclusión

El desarrollo de este proyecto representó un desafío significativo, especialmente al combinarlo con el estudio simultáneo de otras dos carreras. No obstante, este proceso me enseñó el verdadero valor de la resiliencia. Cada error, cada fallo inesperado, no fue una barrera, sino una lección de depuración y perseverancia. La experiencia reafirmó mi creencia de que cuando una persona se propone lograr algo, sus capacidades son solo una parte de la ecuación; la otra, y a menudo la más importante, es el apoyo que recibe. Este proyecto es una prueba de que, con la actitud correcta y las herramientas adecuadas, se pueden superar los obstáculos más grandes. Me llevo conmigo no solo una aplicación funcional, sino también la invaluable lección de que el trabajo y el apoyo valen muchísimo para alcanzar una meta.

Referencias

Bancroft, D. (2023). Python GUI programming with Tkinter. Packt Publishing.

The bcrypt project. (2024). Python library for the bcrypt hashing function. Recuperado de https://pypi.org/project/bcrypt/

GeeksforGeeks. (2024). Python programming language. Recuperado de https://www.geeksforgeeks.org/python-programming-language/

McKinney, W. (2017). Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

Pandas Development Team. (2024). Pandas: Powerful Python data analysis toolkit. Recuperado de https://pandas.pydata.org/docs/

Python Software Foundation. (2024). sqlite3 — DB-API 2.0 interface for SQLite databases. Recuperado de https://docs.python.org/3/library/sqlite3.html

Python Software Foundation. (2024). The Python tutorial. Recuperado de https://docs.python.org/3/tutorial/

Python Software Foundation. (2024). Tkinter — Python interface to Tcl/Tk. Recuperado de https://docs.python.org/3/library/tkinter.html

Wes McKinney, & the pandas development team. (2024). User Guide. Recuperado de https://pandas.pydata.org/docs/user_guide/index.html

Wes McKinney, & the pandas development team. (2024). API Reference. Recuperado de https://pandas.pydata.org/docs/reference/index.html

ANEXOS

Anexo I: Código Fuente de la Aplicación

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import sqlite3
from bcrypt import hashpw, gensalt, checkpw
from functools import partial
import pandas as pd
import io

DATABASE = 'calificaciones.db'

def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

def create_tables():
    """Crea las tablas de la base de datos si no existen."""
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY,
            username TEXT NOT NULL UNIQUE,
            password_hash BLOB NOT NULL
        )
    ''')
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS calificaciones (
            id INTEGER PRIMARY KEY,
            materia TEXT NOT NULL,
            nota REAL NOT NULL,
            peso REAL DEFAULT 1.0,
            nombre_actividad TEXT NOT NULL,
            user_id INTEGER NOT NULL,
            FOREIGN KEY (user_id) REFERENCES users (id)
        )
    ''')
    conn.commit()
    conn.close()

class LoginRegisterApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Iniciar Sesión / Registro")
        self.root.geometry("350x250")
        self.current_user_id = None
        self.main_window = None

        self.create_login_widgets()

    def create_login_widgets(self):
```

```
            for widget in self.root.winfo_children():
                widget.destroy()

            self.login_frame = ttk.Frame(self.root, padding="20")
            self.login_frame.pack(expand=True)

            ttk.Label(self.login_frame, text="Usuario:", font=('Helvetica',
12)).grid(row=0, column=0, pady=5)
            self.username_entry = ttk.Entry(self.login_frame, width=25)
            self.username_entry.grid(row=0, column=1, pady=5)

            ttk.Label(self.login_frame, text="Contraseña:", font=('Helvetica',
12)).grid(row=1, column=0, pady=5)
            self.password_entry = ttk.Entry(self.login_frame, show="*", width=25)
            self.password_entry.grid(row=1, column=1, pady=5)

            ttk.Button(self.login_frame, text="Iniciar Sesión",
command=self.check_login).grid(row=2, column=0, pady=15, padx=5)
            ttk.Button(self.login_frame, text="Registrarse",
command=self.show_register_widgets).grid(row=2, column=1, pady=15, padx=5)

        def show_register_widgets(self):
            for widget in self.root.winfo_children():
                widget.destroy()

            self.register_frame = ttk.Frame(self.root, padding="20")
            self.register_frame.pack(expand=True)

            ttk.Label(self.register_frame, text="Usuario:", font=('Helvetica',
12)).grid(row=0, column=0, pady=5)
            self.reg_username_entry = ttk.Entry(self.register_frame, width=25)
            self.reg_username_entry.grid(row=0, column=1, pady=5)

            ttk.Label(self.register_frame, text="Contraseña:", font=('Helvetica',
12)).grid(row=1, column=0, pady=5)
            self.reg_password_entry = ttk.Entry(self.register_frame, show="*",
width=25)
            self.reg_password_entry.grid(row=1, column=1, pady=5)

            ttk.Button(self.register_frame, text="Registrar",
command=self.register_user).grid(row=2, column=0, pady=15, padx=5)
            ttk.Button(self.register_frame, text="Cancelar",
command=self.create_login_widgets).grid(row=2, column=1, pady=15, padx=5)

        def register_user(self):
            username = self.reg_username_entry.get()
            password = self.reg_password_entry.get().encode('utf-8')

            if not all([username, password]):
                messagebox.showerror("Error", "Todos los campos son obligatorios.")
                return

            hashed_password = hashpw(password, gensalt())
```

```
                conn = get_db_connection()
                try:
                    conn.execute("INSERT INTO users (username, password_hash) VALUES
(?, ?)", (username, hashed_password))
                    conn.commit()
                    messagebox.showinfo("Registro Exitoso", "Usuario registrado con
éxito.")
                    self.create_login_widgets()
                except sqlite3.IntegrityError:
                    messagebox.showerror("Error", "El nombre de usuario ya existe.")
                finally:
                    conn.close()

            def check_login(self):
                username = self.username_entry.get()
                password = self.password_entry.get().encode('utf-8')

                conn = get_db_connection()
                user = conn.execute("SELECT * FROM users WHERE username = ?",
(username,)).fetchone()
                conn.close()

                if user and checkpw(password, user['password_hash']):
                    self.current_user_id = user['id']
                    self.main_window = MainApp(tk.Toplevel(self.root),
self.current_user_id)
                    self.root.withdraw()
                else:
                    messagebox.showerror("Error", "Usuario o contraseña inválidos.")

        class MainApp:
            def __init__(self, root, user_id):
                self.root = root
                self.user_id = user_id
                self.root.title("Gestor de Calificaciones")
                self.root.geometry("800x600")

                self.create_widgets()
                self.load_calificaciones()
                self.calculate_promedios()

            def create_widgets(self):
                self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

                # Frame para añadir calificación
                add_frame = ttk.Frame(self.root, padding="10")
                add_frame.pack(fill=tk.X)

                ttk.Label(add_frame, text="Materia:").pack(side=tk.LEFT, padx=5)
                self.materia_entry = ttk.Entry(add_frame, width=15)
                self.materia_entry.pack(side=tk.LEFT, padx=5)

                ttk.Label(add_frame, text="Actividad:").pack(side=tk.LEFT, padx=5)
                self.actividad_entry = ttk.Entry(add_frame, width=15)
```

```
            self.actividad_entry.pack(side=tk.LEFT, padx=5)

            ttk.Label(add_frame, text="Nota:").pack(side=tk.LEFT, padx=5)
            self.nota_entry = ttk.Entry(add_frame, width=8)
            self.nota_entry.pack(side=tk.LEFT, padx=5)

            ttk.Label(add_frame, text="Peso:").pack(side=tk.LEFT, padx=5)
            self.peso_entry = ttk.Entry(add_frame, width=8)
            self.peso_entry.pack(side=tk.LEFT, padx=5)

            ttk.Button(add_frame, text="Añadir",
command=self.add_calificacion).pack(side=tk.LEFT, padx=5)
            ttk.Button(add_frame, text="Importar Excel",
command=self.import_calificaciones).pack(side=tk.LEFT, padx=5)
            ttk.Button(add_frame, text="Exportar a Excel",
command=self.export_calificaciones).pack(side=tk.LEFT, padx=5)

            # Frame para promedios
            promedios_frame = ttk.Frame(self.root, padding="10")
            promedios_frame.pack(fill=tk.X)
            self.promedio_label = ttk.Label(promedios_frame, text="Promedio
General: ", font=('Helvetica', 14, 'bold'))
            self.promedio_label.pack(side=tk.LEFT, padx=10)

            self.promedios_tree = ttk.Treeview(promedios_frame, columns=("materia",
"promedio"), show="headings", height=5)
            self.promedios_tree.heading("materia", text="Materia")
            self.promedios_tree.heading("promedio", text="Promedio")
            self.promedios_tree.pack(fill=tk.X, pady=5)

            # Tabla de calificaciones
            table_frame = ttk.Frame(self.root, padding="10")
            table_frame.pack(fill=tk.BOTH, expand=True)

            self.table = ttk.Treeview(table_frame, columns=("materia", "actividad",
"nota", "peso", "id"), show="headings")
            self.table.heading("materia", text="Materia")
            self.table.heading("actividad", text="Actividad")
            self.table.heading("nota", text="Nota")
            self.table.heading("peso", text="Peso")
            self.table.column("id", width=0, stretch=tk.NO)
            self.table.pack(fill=tk.BOTH, expand=True)

            # Botones de acción
            action_frame = ttk.Frame(self.root, padding="10")
            action_frame.pack(fill=tk.X)
            ttk.Button(action_frame, text="Editar",
command=self.edit_calificacion).pack(side=tk.LEFT, padx=5)
            ttk.Button(action_frame, text="Eliminar",
command=self.delete_calificacion).pack(side=tk.LEFT, padx=5)
            ttk.Button(action_frame, text="Cerrar Sesión",
command=self.logout).pack(side=tk.RIGHT, padx=5)

        def load_calificaciones(self):
```

```
                for row in self.table.get_children():
                    self.table.delete(row)

                conn = get_db_connection()
                calificaciones = conn.execute('SELECT * FROM calificaciones WHERE
user_id = ?', (self.user_id,)).fetchall()
                conn.close()

                for calificacion in calificaciones:
                    self.table.insert('', tk.END, values=(calificacion['materia'],
calificacion['nombre_actividad'], calificacion['nota'], calificacion['peso'],
calificacion['id']))

            def calculate_promedios(self):
                for row in self.promedios_tree.get_children():
                    self.promedios_tree.delete(row)

                conn = get_db_connection()
                materias = conn.execute('SELECT DISTINCT materia FROM calificaciones
WHERE user_id = ?', (self.user_id,)).fetchall()

                total_promedio = 0
                total_peso = 0

                for materia in materias:
                    calificaciones = conn.execute('SELECT nota, peso FROM
calificaciones WHERE materia = ? AND user_id = ?', (materia['materia'],
self.user_id)).fetchall()

                    promedio_materia = 0
                    peso_materia = 0

                    for cal in calificaciones:
                        promedio_materia += cal['nota'] * cal['peso']
                        peso_materia += cal['peso']

                    if peso_materia > 0:
                        promedio_final_materia = promedio_materia / peso_materia
                        self.promedios_tree.insert('', tk.END,
values=(materia['materia'], f"{promedio_final_materia:.2f}"))
                        total_promedio += promedio_materia
                        total_peso += peso_materia

                if total_peso > 0:
                    promedio_general = total_promedio / total_peso
                    self.promedio_label.config(text=f"Promedio General:
{promedio_general:.2f}")
                else:
                    self.promedio_label.config(text="Promedio General: N/A")

                conn.close()

            def add_calificacion(self):
                materia = self.materia_entry.get()
```

```
            actividad = self.actividad_entry.get()
            nota = self.nota_entry.get()
            peso = self.peso_entry.get()

            if not all([materia, actividad, nota, peso]):
                messagebox.showerror("Error", "Todos los campos son obligatorios.")
                return

            try:
                nota = float(nota)
                peso = float(peso)
            except ValueError:
                messagebox.showerror("Error", "La nota y el peso deben ser números
válidos.")
                return

            conn = get_db_connection()
            conn.execute('INSERT INTO calificaciones (materia, nota, peso,
nombre_actividad, user_id) VALUES (?, ?, ?, ?, ?)', (materia, nota, peso, actividad,
self.user_id))
            conn.commit()
            conn.close()

            self.materia_entry.delete(0, tk.END)
            self.actividad_entry.delete(0, tk.END)
            self.nota_entry.delete(0, tk.END)
            self.peso_entry.delete(0, tk.END)

            self.load_calificaciones()
            self.calculate_promedios()

        def edit_calificacion(self):
            selected_item = self.table.focus()
            if not selected_item:
                messagebox.showerror("Error", "Seleccione una calificación para
editar.")
                return

            item_values = self.table.item(selected_item)['values']
            calificacion_id = item_values[4]

            edit_window = tk.Toplevel(self.root)
            edit_window.title("Editar Calificación")

            conn = get_db_connection()
            calificacion = conn.execute('SELECT * FROM calificaciones WHERE id =
?', (calificacion_id,)).fetchone()
            conn.close()

            ttk.Label(edit_window, text="Materia:").grid(row=0, column=0, padx=5,
pady=5)
            materia_entry = ttk.Entry(edit_window)
            materia_entry.insert(0, calificacion['materia'])
            materia_entry.grid(row=0, column=1, padx=5, pady=5)
```

```
            ttk.Label(edit_window, text="Actividad:").grid(row=1, column=0, padx=5,
pady=5)
            actividad_entry = ttk.Entry(edit_window)
            actividad_entry.insert(0, calificacion['nombre_actividad'])
            actividad_entry.grid(row=1, column=1, padx=5, pady=5)

            ttk.Label(edit_window, text="Nota:").grid(row=2, column=0, padx=5,
pady=5)
            nota_entry = ttk.Entry(edit_window)
            nota_entry.insert(0, calificacion['nota'])
            nota_entry.grid(row=2, column=1, padx=5, pady=5)

            ttk.Label(edit_window, text="Peso:").grid(row=3, column=0, padx=5,
pady=5)
            peso_entry = ttk.Entry(edit_window)
            peso_entry.insert(0, calificacion['peso'])
            peso_entry.grid(row=3, column=1, padx=5, pady=5)

            def save_edit():
                materia = materia_entry.get()
                actividad = actividad_entry.get()
                nota = nota_entry.get()
                peso = peso_entry.get()

                try:
                    nota_f = float(nota)
                    peso_f = float(peso)
                except ValueError:
                    messagebox.showerror("Error", "La nota y el peso deben ser
números válidos.", parent=edit_window)
                    return

                conn = get_db_connection()
                conn.execute('UPDATE calificaciones SET materia = ?,
nombre_actividad = ?, nota = ?, peso = ? WHERE id = ?', (materia, actividad, nota_f,
peso_f, calificacion_id))
                conn.commit()
                conn.close()

                self.load_calificaciones()
                self.calculate_promedios()
                edit_window.destroy()

            ttk.Button(edit_window, text="Guardar Cambios",
command=save_edit).grid(row=4, column=0, columnspan=2, pady=10)

        def delete_calificacion(self):
            selected_item = self.table.focus()
            if not selected_item:
                messagebox.showerror("Error", "Seleccione una calificación para
eliminar.")
                return
```

```
            confirm = messagebox.askyesno("Confirmación", "¿Está seguro de que
desea eliminar esta calificación?")
            if confirm:
                calificacion_id = self.table.item(selected_item)['values'][4]
                conn = get_db_connection()
                conn.execute('DELETE FROM calificaciones WHERE id = ?',
(calificacion_id,))
                conn.commit()
                conn.close()

                self.load_calificaciones()
                self.calculate_promedios()

        def logout(self):
            self.root.master.deiconify()
            self.root.destroy()

        def on_closing(self):
            self.root.master.deiconify()
            self.root.destroy()

        def import_calificaciones(self):
            file_path = filedialog.askopenfilename(filetypes=[("Excel files",
"*.xlsx")])
            if not file_path:
                return

            try:
                df = pd.read_excel(file_path)
                conn = get_db_connection()
                for index, row in df.iterrows():
                    conn.execute('INSERT INTO calificaciones (materia, nota, peso,
nombre_actividad, user_id) VALUES (?, ?, ?, ?, ?)',
                                 (row['materia'], row['nota'], row['peso'],
row['nombre_actividad'], self.user_id))
                conn.commit()
                conn.close()
                messagebox.showinfo("Importación Exitosa", "Calificaciones
importadas con éxito.")
                self.load_calificaciones()
                self.calculate_promedios()
            except Exception as e:
                messagebox.showerror("Error de Importación", f"Hubo un error al
importar el archivo: {e}")

        def export_calificaciones(self):
            try:
                conn = get_db_connection()
                df = pd.read_sql_query('SELECT materia, nombre_actividad, nota,
peso FROM calificaciones WHERE user_id = ?', conn, params=(self.user_id,))
                conn.close()

                file_path = filedialog.asksaveasfilename(defaultextension=".xlsx",
filetypes=[("Excel files", "*.xlsx")], initialfile="Calificaciones")
```

```
                    if file_path:
                        df.to_excel(file_path, index=False,
sheet_name='Calificaciones')
                        messagebox.showinfo("Exportación Exitosa", f"Calificaciones
exportadas a:\n{file_path}")
                except Exception as e:
                    messagebox.showerror("Error de Exportación", f"Hubo un error al
exportar el archivo: {e}")

        if __name__ == "__main__":
            create_tables()
            root = tk.Tk()
            app = LoginRegisterApp(root)
            root.mainloop()
```

Anexos II: Glosario De Términos

Este glosario define los términos más relevantes utilizados en el desarrollo del proyecto, facilitando la comprensión de los conceptos clave.

- CRUD: Un acrónimo de "Create, Read, Update, Delete". Se refiere a las cuatro operaciones básicas que se realizan en una base de datos para gestionar datos. En este proyecto, se aplica a la gestión de calificaciones.
- Depuración (Debugging): Proceso de identificar, analizar y eliminar errores (bugs) en el código de un programa. Es una habilidad fundamental en el desarrollo de software.
- Framework: Un conjunto de herramientas, bibliotecas y convenciones predefinidas que sirven como esqueleto para desarrollar una aplicación. Los frameworks como Flask o PyQt ahorran tiempo al proveer una estructura base.
- GUI (Graphical User Interface): Una interfaz gráfica de usuario que permite a las personas interactuar con el software a través de elementos visuales como ventanas, botones e iconos, en lugar de usar comandos de texto.
- Librería: Un conjunto de funciones y clases preescritas que se pueden usar para realizar tareas específicas. En este proyecto, bcrypt, pandas y tkinter son librerías.
- Pip: El sistema de gestión de paquetes estándar para Python, utilizado para instalar y gestionar librerías.
- Resiliencia: En el contexto de la programación, se refiere a la capacidad del desarrollador para recuperarse de errores, superar obstáculos técnicos y mantener la motivación a pesar de las dificultades.

- SQLite: Un motor de base de datos relacional ligero y de código abierto que se integra directamente en la aplicación. No necesita un servidor externo para funcionar, lo que lo hace ideal para aplicaciones de escritorio pequeñas.

Anexos III: Guía de Uso

Esta sección proporciona los pasos necesarios para instalar las dependencias y ejecutar la aplicación desde la línea de comandos.

- Requisitos previos: Asegúrate de tener Python 3.x instalado en tu sistema.
- Instalación de librerías: Abre la terminal o el símbolo del sistema, navega hasta la carpeta del proyecto y ejecuta el siguiente comando para instalar todas las dependencias necesarias:

  ```
  Bash

  pip install bcrypt pandas openpyxl
  ```

- Este comando asegura que tu entorno de Python tenga todas las librerías necesarias para que la aplicación funcione correctamente.
- Ejecución del código: Una vez que las librerías estén instaladas, puedes ejecutar el archivo principal de la aplicación con el siguiente comando:

  ```
  Bash

  python app.py
  ```

- Uso de la aplicación:
- La aplicación se abrirá en la ventana de Inicio de Sesión / Registro.
- Registro: Si es tu primera vez, crea una nueva cuenta con un nombre de usuario y una contraseña. La aplicación generará la base de datos calificaciones.db automáticamente.
- Inicio de Sesión: Ingresa las credenciales que acabas de crear para acceder al gestor de calificaciones.
- Gestión de Notas: Puedes utilizar los campos de entrada para añadir nuevas calificaciones y la tabla para visualizar, editar o eliminar las notas existentes.

- Importar/Exportar: Utiliza los botones correspondientes para importar calificaciones desde un archivo Excel (.xlsx) o para exportar tus datos a uno.

Anexo IV: Enlace al repositorio de GitHub

https://github.com/kingsophh/mi_calificador_tkinker.git