

APP PARA GESTIONAR PROYECTOS

**PROYECTO DE DESARROLLO DE
APLICACIONES MULTIPLATAFORMA**
CURSO 2024/25

María Fernanda Espinoza
Fátima Esmeralda Valle Argueta

ÍNDICE

1.- Descripción	4
2.- Análisis del Mercado Actual	5
3.- Propuesta Innovadora	6
4.- Estudio de Viabilidad	7
4.1.- Descripción del modelo de negocio	7
4.2.- Embudo de marketing	9
4.3.- Estimación de la población objetivo	9
4.4.- Cálculo de la muestra poblacional necesaria para la encuesta	10
4.5.- Encuesta y resultados	10
4.6.- Planificación del tiempo	11
4.9.- Conclusiones	12
5.- Arquitectura de la Aplicación	13
5.1.- Recursos y tecnologías empleadas	15
5.2.- Diseño inicial	16
5.3.- Diseño final	17
6.- Futuras Mejoras y Conclusiones	18
7.- Pruebas y Resultados	18
8.- Manual Usuario	19
9.- Bibliografía	22
10 - Anexo I.- Principales clases del código	23
11 - Anexo 2: Opinión personal sobre la FCT	40

1.- DESCRIPCIÓN

Gestor de Tareas es una aplicación de escritorio diseñada para la organización y gestión eficiente de tareas, proyectos personales o colaborativos. Su estructura se basa en un sistema visual de **tableros, listas y tareas**, inspirado en metodologías como Kanban, y adaptado para su uso tanto individual como en equipo.

Esta aplicación permite al usuario crear múltiples tableros personalizados para diferentes áreas de trabajo (por ejemplo, académico, personal o laboral), y dentro de cada tablero organizar las tareas en listas. Cada tarea puede incluir una descripción, una fecha de vencimiento, y su estado actual (“pendiente” o “completada”), permitiendo así un seguimiento claro del progreso.

El entorno visual está construido con la biblioteca **tkbootstrap**, que ofrece una interfaz moderna, clara e intuitiva, ideal para usuarios que buscan simplicidad sin perder funcionalidad.

Funciones principales:

- **Creación y gestión de tableros:** permite tener múltiples espacios de trabajo independientes.
 - **Listas organizadas por tablero:** agrupan tareas por etapas, categorías o prioridades.
 - **Tareas con detalles:** cada tarea puede tener título, descripción, fecha de vencimiento y estado.
 - **Cambio de estado rápido:** se puede marcar cualquier tarea como completada o pendiente con un clic.
 - **Búsqueda de tableros:** filtro por nombre para localizar proyectos fácilmente.
 - **Visualización por fecha:** ordena todas las tareas de un tablero cronológicamente.
 - **Interfaz moderna y responsiva:** basada en **tkbootstrap**, con botones, estilos y navegación clara.
 - **Base de datos local (SQLite):** todos los datos se guardan de forma segura en el equipo del usuario.
-

La aplicación está pensada tanto para estudiantes que deseen organizar sus entregas y exámenes, como para profesionales que buscan una solución ligera y funcional para gestionar sus tareas del día a día. Además, gracias a su diseño modular, puede ser fácilmente ampliada con futuras funcionalidades como recordatorios, sincronización en la nube, o trabajo colaborativo.

2.- ANÁLISIS DEL MERCADO ACTUAL

El mercado de herramientas de gestión de proyectos es altamente competitivo, impulsado por la demanda de soluciones que mejoren la productividad, colaboración y eficiencia. Las aplicaciones de gestión de proyectos son esenciales en entornos de trabajo digitalizados, especialmente para equipos distribuidos o que trabajan de manera remota.

Principales Competidores:

1. Trello

- **Descripción:** Herramienta visual y fácil de usar basada en tableros, listas y tarjetas.
- **Fortalezas:** Simple, ideal para proyectos pequeños y personales, integración con otras herramientas.
- **Debilidades:** Funciones avanzadas limitadas, no adecuado para equipos grandes o proyectos complejos.

2. Asana

- **Descripción:** Plataforma robusta con herramientas avanzadas para equipos grandes y proyectos complejos.
- **Fortalezas:** Amplias funciones de seguimiento y colaboración, vistas personalizables.
- **Debilidades:** Curva de aprendizaje alta, versión gratuita limitada, costoso en su versión premium.

3. Monday.com

- **Descripción:** Plataforma visual para la gestión de proyectos con herramientas avanzadas y automatización.
- **Fortalezas:** Interfaz visual, altamente personalizable, ideal para equipos medianos a grandes.
- **Debilidades:** Costoso para equipos pequeños, curva de aprendizaje algo elevada.

4. ClickUp

- **Descripción:** Herramienta todo-en-uno para gestionar tareas, objetivos, documentos y más.
- **Fortalezas:** Amplias funcionalidades y personalización, excelente para gestionar plazos y rendimiento.
- **Debilidades:** Interfaz compleja para nuevos usuarios, puede ser abrumadora para equipos pequeños.

5. Basecamp

- **Descripción:** Plataforma sencilla para gestión de proyectos pequeños o medianos, con enfoque en la simplicidad.
- **Fortalezas:** Muy fácil de usar, ideal para equipos pequeños, comunicación en tiempo real.
- **Debilidades:** Falta de funciones avanzadas, menos visual y flexible que otras herramientas.

El mercado sigue creciendo con la transición al trabajo remoto. Aunque existen muchos competidores, **App gestor de tareas** puede destacarse al ofrecer una experiencia más personalizada y accesible, integrando tecnologías como IA y comandos de voz. Esto le permitirá convertirse en una opción preferida para equipos que buscan una herramienta eficiente y fácil de usar.

3.- PROPUESTA INNOVADORA

Aunque actualmente la aplicación ya permite una gestión visual y organizada de tareas mediante tableros, listas y tarjetas, se plantea una serie de **mejoras innovadoras** que podrían implementarse en versiones futuras para diferenciar el producto en el mercado y mejorar la experiencia de usuario:

1. Integración de Comandos por Voz

Función: Permitir que el usuario cree, edite o marque tareas como completadas utilizando comandos de voz mediante bibliotecas como `speech_recognition`.

Ventaja diferencial: Mejora notablemente la accesibilidad, especialmente para personas con movilidad reducida o usuarios multitarea.

Ejemplo:

“Crear tarea: entregar trabajo de matemáticas el viernes.”
“Marcar tarea como completada: informe mensual.”

2. Modo Productividad Inteligente (con IA básica)

Función: Analizar hábitos del usuario (frecuencia de tareas, duración estimada) y sugerir un horario optimizado diario para realizarlas.

Ventaja diferencial: Aumenta la eficiencia sin requerir configuración avanzada del usuario.

Ejemplo:

“Te sugerimos trabajar en el resumen de historia entre 17:00 y 18:00 según tu disponibilidad.”

3. Tableros Adaptativos por Tipo de Usuario

Función: Detectar el tipo de usuario (académico, personal, empresarial) y ofrecer estructuras de tablero prediseñadas.

Ventaja diferencial: Mejora la usabilidad desde el primer uso, evitando configuraciones manuales.

Ejemplo:

- Usuario estudiante → Tablero con listas: “Exámenes”, “Trabajos”, “Lecturas”.
 - Usuario emprendedor → “Pendientes”, “En proceso”, “Finalizado”.
-

4. Modo Silencio por Proyecto

Función: Desactivar notificaciones de ciertos proyectos temporalmente.

Ventaja diferencial: Mejora el enfoque del usuario en tareas prioritarias sin perder el seguimiento global.

Ejemplo:

Durante una semana de exámenes, se silencia el tablero “Ideas de negocio” para evitar distracciones.

5. Sistema de Logros y Motivación

Función: Introducir una mecánica de recompensas o insignias por completar tareas o mantener hábitos de organización.

Ventaja diferencial: Añade un componente lúdico (gamificación) que mejora la constancia y el compromiso del usuario.

Ejemplo:

🏆 ¡Felicidades! Completaste 10 tareas esta semana sin retrasos.”

4.- ESTUDIO DE VIABILIDAD

4.1.- DESCRIPCIÓN DEL MODELO DE NEGOCIO

El modelo de negocio propuesto para la aplicación **Gestor de Tareas** se basa en una estrategia **freemium**, que combina el acceso gratuito a funcionalidades esenciales con una versión premium de pago que incorpora herramientas avanzadas orientadas a mejorar la productividad, colaboración y personalización.

Este enfoque permite **atraer a estudiantes, profesionales independientes y pequeños equipos** con una versión funcional y completa, al mismo tiempo que se ofrece una versión más potente y con valor añadido para quienes necesiten más capacidades.

Versión Gratuita (Free Tier)

Accesible para cualquier usuario sin coste, incluye:

- Creación ilimitada de tableros, listas y tareas.

- Gestión visual de tareas con fechas y estados.
- Filtrado por fecha y búsqueda de tableros.
- Interfaz moderna y responsiva.
- Almacenamiento local en base de datos SQLite.

Esta versión está pensada especialmente para estudiantes, autónomos o usuarios personales que buscan una herramienta sencilla y eficiente para organizar sus tareas.

Versión Premium (Suscripción mensual o anual)

Destinada a usuarios más avanzados, equipos de trabajo o pequeñas empresas. Incluirá:

- **Comandos por voz** para crear y completar tareas sin necesidad de escribir.
 - **Asistente inteligente** para sugerencias y organización automática del día.
 - **Modo productividad inteligente** basado en hábitos del usuario.
 - **Análisis de rendimiento y estadísticas de cumplimiento de tareas.**
 - **Plantillas adaptativas según el tipo de proyecto.**
 - **Sincronización y exportación de datos (CSV, PDF, JSON).**
 - **Integración con herramientas externas** como Google Calendar, Notion o Slack.
-

Otras modalidades previstas

- **Licencias educativas:** para centros de formación profesional o institutos con precios reducidos.
 - **Licencias corporativas:** para empresas que deseen gestionar tareas de sus equipos de forma organizada y colaborativa.
 - **Planes familiares o grupales:** donde varios usuarios puedan gestionar proyectos compartidos.
-

Este modelo permite **escalar el uso de la aplicación según las necesidades del usuario**, facilitando la adopción gratuita y promoviendo la conversión hacia versiones de pago mediante funcionalidades de alto valor.

4.2.- EMBUDO DE MARKETING

El embudo de marketing se enfocará en atraer usuarios desde canales digitales y convertirlos en clientes fidelizados mediante estrategias escalonadas:

1. Atracción (TOFU - Top of the Funnel):

- Publicidad en redes sociales (Instagram, TikTok, LinkedIn).
- Posicionamiento SEO con contenido útil (blogs sobre productividad, organización y trabajo remoto).
- Publicaciones patrocinadas en foros y grupos de estudiantes y emprendedores.

2. Interés y consideración (MOFU - Middle of the Funnel):

- Landing pages con demos interactivas.
- Testimonios de usuarios reales.
- Videos mostrando cómo usar las funciones avanzadas.

3. Conversión (BOFU - Bottom of the Funnel):

- Prueba gratuita de 14 días del plan premium.
- Email marketing con ofertas limitadas.
- Comparativas con otras herramientas como Trello y Asana destacando funciones diferenciales.

4. Retención y fidelización:

- Sistema de recompensas por uso constante.
- Encuestas de satisfacción y sugerencias.
- Actualizaciones periódicas con mejoras solicitadas por usuarios.

4.3.- ESTIMACIÓN DE LA POBLACIÓN OBJETIVO

La aplicación **Gestor de Tareas** está diseñada para cubrir las necesidades de organización de un público amplio que requiere gestionar tareas y proyectos de forma eficiente, visual y flexible. Su enfoque modular y adaptable permite que sea útil en distintos contextos educativos y profesionales.

Segmentos principales de usuarios:

1. Estudiantes de bachillerato, formación profesional y universidad:

- Necesitan una herramienta para organizar sus trabajos, exámenes, entregas y prácticas.

- Valoran una interfaz intuitiva, sin complicaciones técnicas.

2. Profesionales y trabajadores en equipo:

- Personas que gestionan tareas en entornos colaborativos, tanto en empresas como en proyectos freelance.
- Requieren asignación de tareas, seguimiento por fechas y claridad visual del flujo de trabajo.

3. Usuarios técnicos y docentes (público extendido):

- Programadores, profesores, desarrolladores y técnicos que desean llevar un control personal o de equipo sobre tareas, bugs, revisiones o contenidos educativos.



Estimación específica de la muestra para análisis de viabilidad

Como parte del estudio práctico de viabilidad, se ha tomado como referencia el grupo de estudiantes del **primer curso de Desarrollo de Aplicaciones Multiplataforma (DAM/DAW)** en el centro educativo:

- **Tamaño de la población objetivo directa:** 18 estudiantes.
- **Muestra seleccionada para estudio inicial (25%):** 4 estudiantes.

Esta muestra servirá para realizar encuestas y pruebas funcionales que permitan validar la aceptación de la herramienta, identificar puntos de mejora y evaluar la viabilidad del proyecto en un entorno realista.

4.4.- CÁLCULO DE LA MUESTRA POBLACIONAL NECESARIA PARA LA ENCUESTA

https://docs.google.com/forms/d/e/1FAIpQLSf3fAB0q57AmPPakT3GO_YuUEPpaT96LWfmBfcyLreVf8vmsw/viewform?usp=dialog

4.5.- ENCUESTA Y RESULTADOS

Para conocer el interés real en la aplicación **Gestor de Tareas**, se ha diseñado y distribuido una encuesta a una muestra representativa de estudiantes del **primer curso de DAM/DAW**, quienes pertenecen al perfil objetivo principal de la aplicación. La encuesta se realizó mediante un formulario de Google Forms.



Enlace al formulario:

https://docs.google.com/forms/d/e/1FAIpQLSf3fAB0q57AmPPakT3GO_YuUEPpaT96LWfmBfcyLreVf8vmsw/viewform



Objetivos de la encuesta:

- Analizar si los estudiantes utilizan actualmente herramientas de gestión de tareas.
- Evaluar el interés por una aplicación con funciones simples y visuales.

- Validar la aceptación de funcionalidades como tableros, fechas límite, filtros o estados de tareas.
- Detectar el interés en funcionalidades avanzadas como comandos de voz o asistentes inteligentes.

Datos de la muestra:

- Población total: 18 estudiantes
- Muestra encuestada: 4 estudiantes ($\approx 25\%$)

Principales preguntas y resultados obtenidos:

Pregunta	Resultados Destacados
¿Utilizas actualmente alguna app para organizar tus tareas o proyectos?	3 de 4 respondieron que <i>sí</i> (75%)
¿Con qué frecuencia olvidas una tarea o entrega sin una app de ayuda?	2 de 4 dijeron <i>a menudo</i> , 2 dijeron <i>a veces</i>
¿Qué funcionalidades te parecen más útiles?	Todos destacaron: <i>tableros visuales</i> , <i>fechas límite</i> y <i>estados</i>
¿Te gustaría usar comandos de voz o asistentes inteligentes?	3 de 4 dijeron <i>sí</i> , 1 dijo <i>tal vez</i>
¿Preferirías una app sencilla pero personalizable en lugar de compleja?	100% respondió <i>sí</i>

Conclusiones del sondeo:

- Existe un **interés claro** por herramientas visuales para organizar tareas.
- El **nivel de adopción** de apps similares (como Trello o Notion) es parcial, lo que representa una oportunidad.
- Las funciones avanzadas como comandos por voz o recomendaciones automáticas son **muy bien valoradas**, incluso por usuarios no técnicos.
- La muestra valida que **una app sencilla, moderna y flexible** como la propuesta sería bien recibida en el entorno educativo.

4.6.- PLANIFICACIÓN DEL TIEMPO

Para garantizar una correcta ejecución del proyecto **Gestor de Tareas**, se ha establecido una planificación temporal estructurada en varias fases. Cada etapa tiene objetivos definidos y una duración estimada que permite un avance progresivo, organizado y realista.



Fases del desarrollo del proyecto

Fase	Actividades principales	Duración estimada
1. Análisis y diseño	<ul style="list-style-type: none"> - Definición de requisitos - Propuesta inicial - Diagrama entidad-relación (BD) 	1 semana
2. Desarrollo backend	<ul style="list-style-type: none"> - Creación de base de datos SQLite - Desarrollo de clase Database - Funciones CRUD 	1 semana
3. Desarrollo frontend	<ul style="list-style-type: none"> - Diseño de la interfaz con tkbootstrap - Login de usuario - Dashboard principal 	1 semana
4. Integración y pruebas	<ul style="list-style-type: none"> - Integración de backend y frontend - Pruebas funcionales completas 	1 semana
5. Documentación	<ul style="list-style-type: none"> - Redacción de memoria del proyecto - Manual de usuario - Capturas, anexos y pruebas 	1 semana
6. Presentación final	<ul style="list-style-type: none"> - Preparación de exposición - Revisión de errores - Entrega oficial 	2-3 días



Total estimado del proyecto:

Aproximadamente 5 a 6 semanas.



Observaciones:

- La planificación ha sido diseñada para adaptarse al calendario académico.
- Cada semana se centra en una parte crítica del desarrollo para asegurar un avance constante y sin sobrecarga.
- Las tareas han sido distribuidas considerando tiempos de prueba y ajustes.

4.9.- CONCLUSIONES

Tras completar el desarrollo y análisis del proyecto **Gestor de Tareas**, puedo afirmar que la aplicación tiene un **potencial real** para convertirse en una herramienta útil y funcional tanto en entornos académicos como profesionales. Su enfoque visual, basado en tableros, listas y tareas, permite una organización clara y ordenada, adaptada a distintos tipos de usuarios.



Puntos clave que destacan del proyecto:

- La aplicación **resuelve una necesidad concreta**: organizar tareas de forma sencilla, visual y accesible.
- Se diferencia de otras soluciones como Trello o Asana al **integrar ideas innovadoras** como comandos por voz, un asistente inteligente y tableros adaptativos según el tipo de

usuario.

- Ofrece una **experiencia moderna** gracias al uso de la librería **tkbootstrap**, que mejora la interfaz gráfica sin complicar su uso.

Modelo de negocio acertado:

El modelo freemium permite que cualquier usuario acceda a las funciones básicas sin coste, lo que la hace ideal para estudiantes o trabajadores individuales. A su vez, la versión premium abre la puerta a funcionalidades más avanzadas, lo cual genera **una vía de ingresos sostenible** y escalable.

Validación mediante encuesta:

Aunque la encuesta se realizó a una muestra reducida de estudiantes (4 personas), los resultados fueron **muy positivos**: se valoraron especialmente las funciones visuales, la simplicidad de uso y las posibles mejoras tecnológicas como comandos por voz o planificación inteligente.

Perspectiva de futuro:

Gracias a su diseño modular y escalable, la aplicación tiene mucho margen de mejora: integración en la nube, colaboración en tiempo real, notificaciones, exportación de datos, entre otras funciones. Esto permite seguir evolucionando el producto y adaptarlo a nuevos entornos o necesidades.

En resumen, el proyecto ha cumplido sus objetivos iniciales, ha sido validado por usuarios reales, y demuestra tener una base sólida para futuras mejoras y expansión. Estoy satisfecho con el resultado, tanto a nivel técnico como en cuanto al aprendizaje personal que he adquirido durante su desarrollo.

5.- ARQUITECTURA DE LA APLICACIÓN

La arquitectura del proyecto **Gestor de Tareas** está basada en un enfoque **modular y orientado a objetos**, lo que permite mantener una separación clara entre la lógica de negocio, la gestión de datos y la interfaz gráfica. Este diseño facilita tanto la comprensión del código como su mantenimiento y escalabilidad futura.

5.1. Estructura y Componentes Principales

Clases principales del sistema:

- **Database**
Encargada de gestionar todas las operaciones con la base de datos SQLite. Contiene métodos para el manejo de usuarios, tableros, listas y tareas, así como búsqueda y actualizaciones.
- **LoginWindow**
Se encarga de la interfaz de autenticación. Permite a los usuarios iniciar sesión o registrarse de forma sencilla mediante una ventana construida con **tkbootstrap**.
- **Dashboard**
Representa la interfaz principal del usuario una vez ha iniciado sesión. Desde aquí se

gestionan tableros, listas y tareas con acciones como crear, editar, eliminar, filtrar o buscar.

Interfaz gráfica (GUI):

- Construida completamente con **Tkinter** y mejorada visualmente con **ttkbootstrap**, que permite aplicar temas modernos y responsivos.
 - Uso de widgets como **Frame**, **Label**, **Entry**, **Button**, **Toplevel**, **SimpleDialog**, y **MessageBox** para lograr una experiencia intuitiva.
 - Diseño visual jerárquico: ventana principal → tableros → listas → tareas.
-

5.2. Flujo de Datos y Operaciones

1. **El usuario interactúa con la GUI** mediante botones o formularios.
 2. **Los eventos de la interfaz disparan métodos** definidos en las clases (**Dashboard**, **Database**), según la acción deseada.
 3. **Los métodos del backend acceden a la base de datos** mediante **sqlite3**, recuperan, modifican o almacenan información.
 4. **La interfaz se actualiza dinámicamente** para reflejar los cambios, ya sea al mostrar tableros, listas o tareas.
 5. Se incluyen diálogos para confirmación, creación o edición, garantizando una interacción amigable.
-

5.3. Ventajas del Diseño

- **Modularidad:** Cada archivo (login, panel, base de datos) cumple una función específica, lo que facilita mantener y ampliar el sistema.
 - **Escalabilidad:** Es posible integrar nuevas funcionalidades (notificaciones, sincronización en la nube, usuarios colaborativos) sin reestructurar el código base.
 - **Reutilización:** Clases como **Database** son reutilizables en otros proyectos con bases de datos similares.
 - **Claridad estructural:** Separar la interfaz de usuario de la lógica de negocio hace que el código sea más legible y mantenible.
 - **Persistencia real de datos:** Gracias a SQLite, toda la información se conserva entre sesiones, incluyendo usuarios, tareas y su estado.
-

5.4. Consideraciones Técnicas

- La aplicación se ejecuta en un **único hilo principal**, aprovechando el loop de eventos de Tkinter.

- La base de datos **SQLite** está implementada en un archivo local (**app.db**) y se inicializa automáticamente si no existe.
- El uso de **ttkbootstrap** permite mantener una interfaz elegante y profesional sin complicaciones.
- El sistema ha sido desarrollado para **entornos Windows y Linux** con Python 3.x instalado.
- Todos los datos están protegidos por claves primarias y relaciones entre entidades mediante **claves foráneas (FOREIGN KEY)** con borrado en cascada.
-

5.1.- RECURSOS Y TECNOLOGÍAS EMPLEADAS

Lenguaje: Python 3.x

Se usará **Python 3.x**, que es la versión moderna y actualizada del lenguaje. Python es muy popular para proyectos de escritorio por su facilidad y la gran cantidad de bibliotecas disponibles.

GUI: Tkinter + ttkbootstrap

- **Tkinter**: Es la biblioteca estándar de Python para crear interfaces gráficas (ventanas, botones, formularios, etc.). Es sencilla y viene incluida con Python, por lo que no necesitas instalar nada extra.
 - **ttkbootstrap**: Es una extensión de Tkinter que permite darle una apariencia más moderna y atractiva, parecida a los estilos de Bootstrap (muy usados en web). Esto mejora mucho el aspecto visual sin complicar el código.
-

Base de datos: SQLite integrada localmente con esquema relacional

- **SQLite**: Es una base de datos ligera que se guarda en un solo archivo local, ideal para aplicaciones de escritorio. No requiere instalación ni configuración de un servidor externo.
 - **Esquema relacional**: Significa que la base de datos tendrá tablas con relaciones entre ellas (por ejemplo, usuarios, roles, productos, etc.), para organizar bien la información.
-

Estructura de carpetas y archivos

Esto ayuda a organizar el código para que sea fácil de mantener y entender. Cada archivo tiene una función concreta:

- **database.py**
Aquí estará el código encargado de conectar con la base de datos SQLite, ejecutar consultas (SELECT, INSERT, UPDATE, DELETE) y devolver los resultados. También puede incluir funciones para crear las tablas si no existen.
- **login.py**
Este archivo contiene la interfaz de inicio de sesión (ventana donde el usuario pone su

usuario y contraseña). Además, manejará la lógica para validar el usuario y permitir o denegar el acceso.

- **panel.py**
Representa el "dashboard" o panel principal que ve el usuario una vez que ha iniciado sesión. Aquí se mostrarán las funcionalidades o datos relevantes según el usuario.
- **main.py**
Es el archivo principal que "une" todo el proyecto. Por ejemplo, es el que inicia la aplicación, llama al login, luego muestra el panel si el login fue exitoso, y gestiona el flujo general del programa.

5.2- Diseño inicial

Diseño Inicial - Diagrama Entidad-Relación (ER)

Entidades y Atributos

Entidad	Atributos	Comentarios
Tablero	- id_tablero (PK) - nombre	Representa un tablero donde se agrupan listas
Lista	- id_lista (PK) - nombre - id_tablero (FK)	Una lista dentro de un tablero
Tarea	- id_tarea (PK) - nombre - descripcion - fecha_vencimiento - prioridad - estado - id_lista (FK)	Una tarea específica dentro de una lista

Relaciones

- Un **Tablero** puede contener muchas **Listas**
(Relación 1 a N, un tablero tiene varias listas)
- Una **Lista** pertenece a un solo **Tablero**
(Relación N a 1)
- Una **Lista** contiene muchas **Tareas**
(Relación 1 a N, una lista tiene varias tareas)
- Una **Tarea** pertenece a una única **Lista**
(Relación N a 1)

Explicación

- Cada **Tablero** es una colección general donde se agrupan varias **Listas**.
- Cada **Lista** pertenece a un solo tablero y contiene varias **Tareas**.
- Cada **Tarea** es un ítem individual que tiene su nombre, descripción, fecha límite, prioridad, estado y pertenece a una lista concreta.

5.3-Diseño final

Separación en clases

Para mantener el código organizado y facilitar el mantenimiento, se ha estructurado la aplicación en tres clases principales, cada una con responsabilidades claras:

- **Database**
Gestiona la conexión con la base de datos SQLite y las operaciones CRUD (crear, leer, actualizar, borrar) sobre las tablas Tablero, Lista y Tarea. Centraliza toda la persistencia de datos.
- **LoginWindow**
Maneja la interfaz gráfica y la lógica de autenticación del usuario. Incluye los formularios para el ingreso de credenciales y la validación contra la base de datos.
- **Dashboard**
Representa el panel principal de la aplicación tras el inicio de sesión exitoso. Gestiona la visualización dinámica y la interacción con los tableros, listas y tareas, permitiendo crear, modificar y eliminar elementos en tiempo real.

Lógica dividida

La lógica de la aplicación se divide en tres áreas principales para mejorar la claridad y el flujo:

- **Autenticación**
Controla el acceso seguro mediante la validación de usuarios en el LoginWindow, garantizando que solo usuarios registrados puedan acceder al Dashboard.
- **Navegación**
Administra la transición entre ventanas (login y dashboard), así como la navegación interna dentro del dashboard (entre tableros, listas y tareas).
- **Persistencia**
Encargada de todas las operaciones sobre la base de datos, permitiendo que cualquier cambio en la interfaz se refleje inmediatamente en el almacenamiento local.

Experiencia fluida

- La interfaz gráfica se actualiza dinámicamente en respuesta a las acciones del usuario, sin necesidad de reiniciar la aplicación o la ventana.
- Las operaciones sobre tableros, listas y tareas (creación, edición, eliminación) se reflejan en tiempo real, mejorando la experiencia de usuario.

- Se implementan controles visuales intuitivos y responsivos con **ttkbootstrap** para ofrecer un diseño moderno y agradable.

6.- FUTURAS MEJORAS Y CONCLUSIONES


Futuras mejoras para el próximo año

1. **Persistencia de datos con base de datos**
Implementar una base de datos (por ejemplo, SQLite o MySQL) para almacenar permanentemente las tareas, listas y tableros. Esto permitirá que los datos se mantengan entre sesiones y se mejore la gestión y recuperación de la información.
2. **Autenticación y perfiles de usuario**
Añadir un sistema de usuarios con login y perfiles personalizados, para que cada usuario pueda gestionar sus propios tableros y tareas, aumentando la seguridad y personalización del sistema.
3. **Funcionalidades colaborativas**
Incorporar funcionalidades de colaboración en tiempo real, como asignar tareas a diferentes usuarios, comentarios, y notificaciones para mejorar el trabajo en equipo.
4. **Interfaz más avanzada y responsiva**
Mejorar el diseño visual y la experiencia de usuario, integrando más temas, animaciones y una interfaz responsiva para dispositivos móviles.
5. **Integración con servicios externos**
Conectar la aplicación con APIs de calendario, correo electrónico y otras herramientas de productividad para sincronizar tareas y recordatorios.
6. **Exportación e importación de datos**
Añadir opciones para exportar las tareas y tableros en formatos como CSV, JSON o PDF, así como importar datos desde otros sistemas.
7. **Automatización y recordatorios**
Implementar automatizaciones para enviar alertas, recordatorios y permitir reglas personalizadas para la gestión de tareas.

7.- PRUEBAS Y RESULTADOS

Pruebas nuevas realizadas sobre el panel:

7.6. Crear tablero

- **Objetivo:** Añadir un tablero nuevo desde el panel principal.
- **Resultado esperado:** Se agrega y muestra en la lista de tableros.
-  **Resultado:** correcto.

7.7. Buscar tablero

- **Objetivo:** Localizar un tablero escribiendo parte del nombre.
- **Resultado esperado:** Aparecen coincidencias parciales.
- ☒ Resultado: correcto.

7.8. Filtrar tareas por fecha

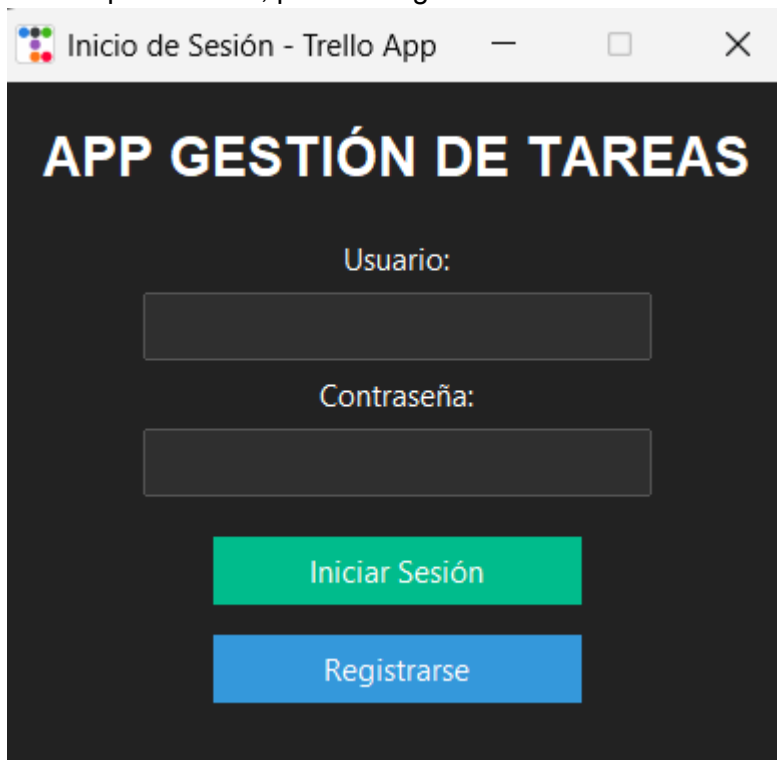
- **Objetivo:** Ver las tareas ordenadas cronológicamente.
- **Resultado esperado:** Vista limpia y ordenada por fecha.
- ☒ Resultado: correcto.

8.- MANUAL USUARIO

8. MANUAL DE USUARIO (actualización)

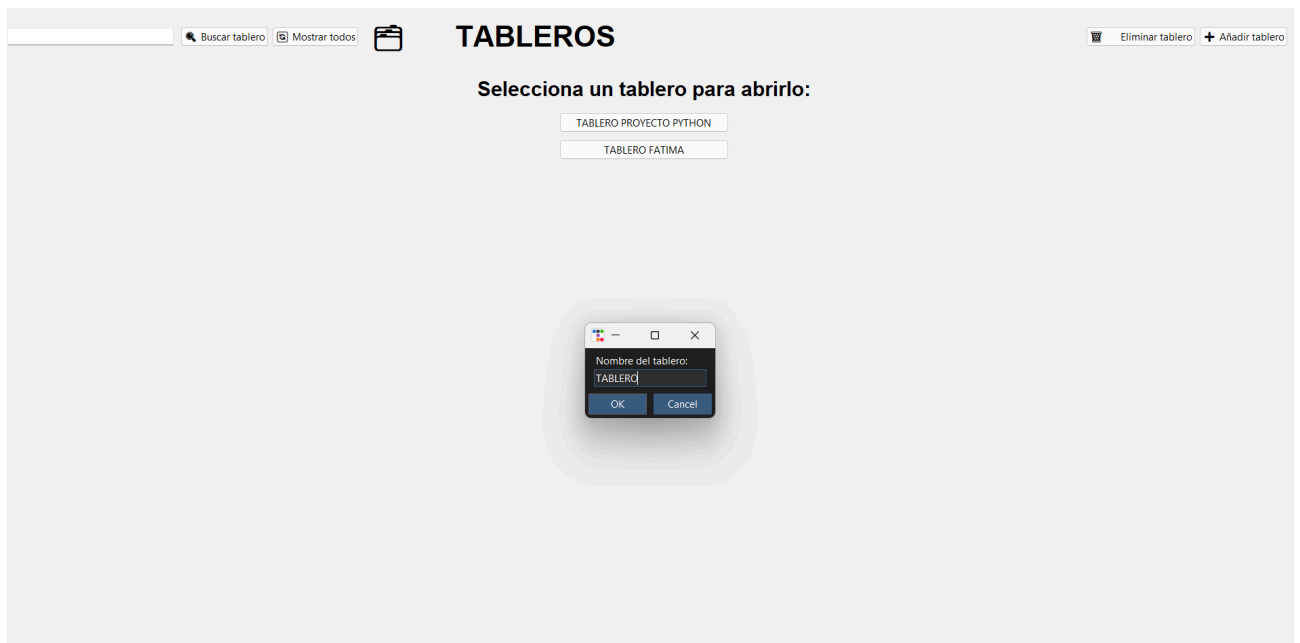
1. Iniciar sesión o registrarse

- Ejecutar el archivo principal.
- Rellenar usuario y contraseña.
- Si es la primera vez, pulsar "Registrarse".



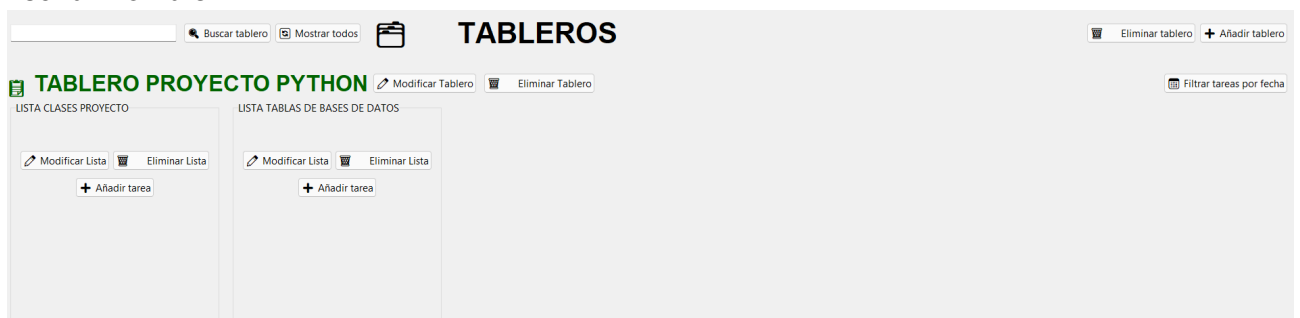
2. Crear tablero

- Pulsar "+ Añadir tablero".
- Escribir un nombre.
- Aparecerá en la lista.



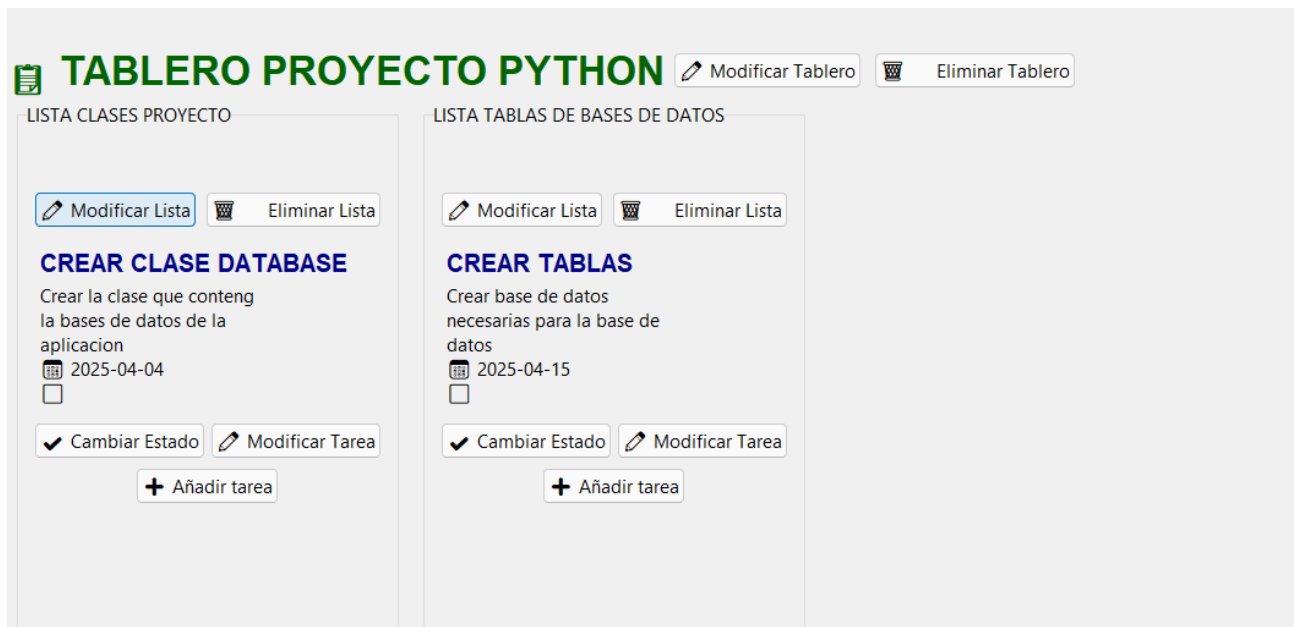
3. Crear lista

- Seleccionar un tablero.
- Pulsar "+ Añadir lista".
- Escribir nombre.



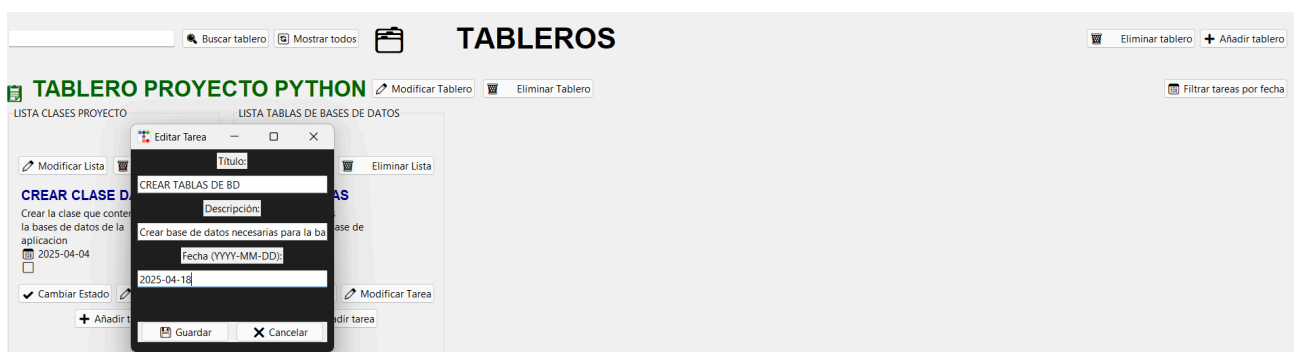
4. Añadir tarea

- Dentro de una lista, pulsar "+ Añadir tarea".
- Ingresar título, descripción y fecha.



5. Editar/eliminar tarea o lista

- Usar botones  y  junto a cada elemento.




6. Marcar tareas como completadas

- Pulsar "✓ Cambiar Estado".

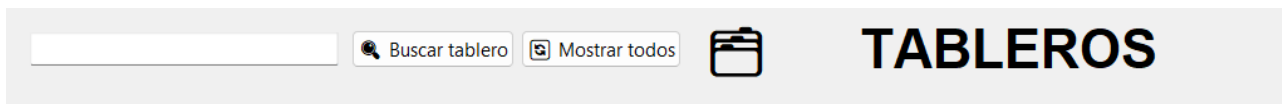


7. Filtrar por fecha

- Dentro del tablero, pulsar "  Filtrar tareas por fecha".



8. Buscar tablero o mostrar todos



9.- BIBLIOGRAFÍA

EMBUDO DE MARKETING

[TOFU, MOFU y BOFU: qué son, ejemplos e implementación](#)

[The Pros and Cons of Using Trello Software](#)

[The Pros and Cons of using Trello](#)

[SoftwareProjectManagers.net+9Bridge24+9Bridge24+9ProjectManagers.net+4ProjectManagers.net+4ProjectManagers.net+4](#)

[Asana Pros and Cons: Top 4 Advantages & Disadvantages](#)

[The Pros and Cons of Using Asana](#)

[SoftwareReddit+2FreshBooks+2upbase.io+2ProjectManagers.net+6Bridge24+6Bridge24+6](#)

[monday.com Review: Features, Pros, Cons, and Pricing](#)

[Project Management Software Monday.com: An In-depth](#)

[Analysisproject-management.com+6Tech.co+6Tech.co+6ProofHub](#)

[The Pros and Cons of Using ClickUp Software](#)

[Honest ClickUp Review 2025: Pros, Cons, Features &](#)

[PricingBridge24+5ProjectManagers.net+5ProjectManagers.net+5Connecteam+2Connecteam+2Connecteam+](#)

[The Pros and Cons of Using Basecamp Software](#)

[Personal Review of Basecamp Project Management](#)

[ToolAceProject+3Bridge24+3Bridge24+3springmanconsulting.com+1springmanconsulting.com+1](#)

10 - ANEXO I.- PRINCIPALES CLASES DEL CÓDIGO

Para que todo funcione hay q importar la libreria ttkbootstrap y tkinter.

MAIN

```
import ttkbootstrap as tb

# from ttkbootstrap.constants import * # Removed due to unresolved import
from database import init_db
from login import LoginWindow
from panel import Dashboard

# Función que se ejecuta al iniciar sesión correctamente
def iniciar_dashboard(usuario_id):
    # Cierra la ventana de login y abre el panel principal
    root.destroy()
    app = Dashboard(usuario_id)
    app.mainloop()

# Punto de entrada principal de la aplicación
if __name__ == "__main__":
    # Inicializar la base de datos (crear tablas si no existen)
    init_db()

    # Crear ventana principal de login
    root = tb.Window(themename="darkly")
    root.title("Gestor de Tareas - Inicio de Sesión")
    LoginWindow(root, iniciar_dashboard)
    root.mainloop()
```

LOGIN

```
import ttkbootstrap as ttk
from tkinter import messagebox
from database import Database

class LoginWindow:
    def __init__(self, root, app_callback):
        self.db = Database()
        self.app_callback = app_callback
        self.root = root

        self.root.title("Inicio de Sesión - Trello App")
        self.root.geometry("400x350")
        self.root.resizable(False, False)

        ttk.Label(self.root, text="APP GESTIÓN DE TAREAS",
font=("Helvetica", 16, "bold")).pack(pady=20)

        ttk.Label(self.root, text="Usuario:").pack()
        self.username_entry = ttk.Entry(self.root, width=30)
        self.username_entry.pack(pady=5)
```

```

        ttk.Label(self.root, text="Contraseña:").pack()
        self.password_entry = ttk.Entry(self.root, show="*", width=30)
        self.password_entry.pack(pady=5)

        ttk.Button(self.root, text="Iniciar Sesión", width=20,
bootstyle="success", command=self.login).pack(pady=15)
        ttk.Button(self.root, text="Registrarse", width=20,
bootstyle="info", command=self.register).pack()

    def login(self):
        username = self.username_entry.get().strip()
        password = self.password_entry.get().strip()

        if not username or not password:
            return messagebox.showwarning("Campos Vacíos", "Por favor,
ingrese usuario y contraseña.")

        try:
            user = self.db.verificar_usuario(username, password)
            if user:
                # No destruimos aquí la ventana, el callback se encargará de
eso
                self.app_callback(user[0]) # Llamamos al callback con el id
del usuario
            else:
                messagebox.showerror("Error de Inicio de Sesión",
"Credenciales incorrectas.")
        except Exception as e:
            messagebox.showerror("Error", f"Ocurrió un error:\n{e}")

    def register(self):
        username = self.username_entry.get().strip()
        password = self.password_entry.get().strip()

        if not username or not password:
            return messagebox.showwarning("Campos Vacíos", "Por favor,
ingrese usuario y contraseña.")

        try:
            if self.db.usuario_existe(username):
                return messagebox.showwarning("Usuario Existente", "El
nombre de usuario ya está en uso.")

            self.db.registrar_usuario(username, password)
            messagebox.showinfo("Registro Exitoso", "Usuario registrado
correctamente. Ahora puede iniciar sesión.")
            self.username_entry.delete(0, 'end')

```



```

        self.password_entry.delete(0, 'end')
    except Exception as e:
        messagebox.showerror("Error", f"No se pudo registrar el
usuario:\n{e}")

DATABASE
import sqlite3

def get_connection():
    return sqlite3.connect("app.db")

def init_db():
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS usuarios (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS tableros (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nombre TEXT NOT NULL,
            usuario_id INTEGER,
            FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE
CASCADE
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS listas (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nombre TEXT NOT NULL,
            tablero_id INTEGER,
            FOREIGN KEY (tablero_id) REFERENCES tableros(id) ON DELETE
CASCADE
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS tareas (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            titulo TEXT NOT NULL,

```

```

        descripcion TEXT,
        fecha TEXT,
        estado TEXT DEFAULT 'pendiente',
        lista_id INTEGER,
        FOREIGN KEY (lista_id) REFERENCES listas(id) ON DELETE CASCADE
    )
    """

    conn.commit()
    conn.close()

class Database:
    def verificar_usuario(self, username, password):
        with get_connection() as conn:
            cur = conn.cursor()
            cur.execute("SELECT id FROM usuarios WHERE username = ? AND
password = ?", (username, password))
            return cur.fetchone()

    def usuario_existe(self, username):
        with get_connection() as conn:
            cur = conn.cursor()
            cur.execute("SELECT id FROM usuarios WHERE username = ?",
(username,))
            return cur.fetchone()

    def registrar_usuario(self, username, password):
        with get_connection() as conn:
            cur = conn.cursor()
            cur.execute("INSERT INTO usuarios (username, password) VALUES
(?, ?)", (username, password))
            conn.commit()

    def obtener_tableros(self, usuario_id):
        with get_connection() as conn:
            cur = conn.cursor()
            cur.execute("SELECT id, nombre FROM tableros WHERE usuario_id =
?", (usuario_id,))
            return cur.fetchall()

    def crear_tablero(self, nombre, usuario_id):
        with get_connection() as conn:
            cur = conn.cursor()
            cur.execute("INSERT INTO tableros (nombre, usuario_id) VALUES
(?, ?)", (nombre, usuario_id))
            conn.commit()

```

```

def actualizar_nombre_tablero(self, tablero_id, nuevo_nombre):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("UPDATE tableros SET nombre = ? WHERE id = ?",
(nuevo_nombre, tablero_id))
        conn.commit()

def buscar_tableros(self, usuario_id, filtro):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("SELECT id, nombre FROM tableros WHERE usuario_id =
? AND nombre LIKE ?",
                    (usuario_id, f'%{filtro}%'))
        return cur.fetchall()

def eliminar_tablero(self, tablero_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM tableros WHERE id = ?", (tablero_id,))
        conn.commit()

def obtener_listas(self, tablero_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("SELECT id, nombre FROM listas WHERE tablero_id =
?", (tablero_id,))
        return cur.fetchall()

def crear_lista(self, nombre, tablero_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("INSERT INTO listas (nombre, tablero_id) VALUES (?,
?)", (nombre, tablero_id))
        conn.commit()

def actualizar_nombre_lista(self, lista_id, nuevo_nombre):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("UPDATE listas SET nombre = ? WHERE id = ?",
(nuevo_nombre, lista_id))
        conn.commit()

def eliminar_lista(self, lista_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM listas WHERE id = ?", (lista_id,))
        conn.commit()

```

```

def obtener_tareas(self, lista_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("SELECT id, titulo, descripcion, fecha, estado FROM
tareas WHERE lista_id = ?", (lista_id,))
        return cur.fetchall()

def crear_tarea(self, titulo, descripcion, fecha, lista_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("""
            INSERT INTO tareas (titulo, descripcion, fecha, estado,
lista_id)
            VALUES (?, ?, ?, 'pendiente', ?)
        """, (titulo, descripcion, fecha, lista_id))
        conn.commit()

def actualizar_tarea(self, tarea_id, titulo, descripcion, fecha,
estado):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("""
            UPDATE tareas
            SET titulo = ?, descripcion = ?, fecha = ?, estado = ?
            WHERE id = ?
        """, (titulo, descripcion, fecha, estado, tarea_id))
        conn.commit()

def eliminar_tarea(self, tarea_id):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("DELETE FROM tareas WHERE id = ?", (tarea_id,))
        conn.commit()

def actualizar_estado_tarea(self, tarea_id, nuevo_estado):
    with get_connection() as conn:
        cur = conn.cursor()
        cur.execute("UPDATE tareas SET estado = ? WHERE id = ?",
(nuevo_estado, tarea_id))
        conn.commit()

```

PANEL O INTERFAZ

```

import ttkbootstrap as tb
from ttkbootstrap.constants import *
from database import get_connection

```

```

import tkinter as tk
from tkinter import simpledialog, Toplevel, messagebox

class Dashboard(tk.Window):
    def __init__(self, usuario_id):
        super().__init__(themename="darkly")
        self.title("Gestor de Tareas - Tableros")
        self.state('zoomed')
        self.usuario_id = usuario_id
        self.tablero_actual_id = None

        top_frame = tk.Frame(self, padding=10)
        top_frame.pack(side=TOP, fill=X)

        self.entry_busqueda = tk.Entry(top_frame, width=30)
        self.entry_busqueda.pack(side=LEFT, padx=10)

        tk.Button(top_frame, text="🔍 Buscar tablero",
bootstyle="primary-outline", command=self.buscar_tablero).pack(side=LEFT)
        tk.Button(top_frame, text="↺ Mostrar todos",
bootstyle="info-outline", command=self.cargar_tableros).pack(side=LEFT,
padx=5)

        tk.Label(top_frame, text="📁 TABLEROS", font=("Arial", 26,
"bold")).pack(side=LEFT, padx=10)

        tk.Button(top_frame, text="+ Añadir tablero", bootstyle="success",
command=self.agregar_tablero).pack(side=RIGHT)
        tk.Button(top_frame, text="🗑 Eliminar tablero", bootstyle="danger",
command=self.eliminar_tablero).pack(side=RIGHT, padx=5)

        self.tablero_frame = tk.Frame(self, padding=10)
        self.tablero_frame.pack(fill=BOTH, expand=YES)

        self.tableros = []
        self.cargar_tableros()

```

```

def agregar_tablero(self):
    nombre = simpledialog.askstring("Nuevo Tablero", "Nombre del
tablero:")
    if nombre:
        conn = get_connection()
        cur = conn.cursor()
        cur.execute("INSERT INTO tableros (nombre, usuario_id) VALUES
(?, ?)", (nombre, self.usuario_id))
        conn.commit()
        conn.close()
        self.cargar_tableros()

def eliminar_tablero(self):
    if not self.tablero_actual_id:
        return
    if messagebox.askyesno("Confirmar", "¿Eliminar este tablero y todo
su contenido?"):
        conn = get_connection()
        cur = conn.cursor()
        cur.execute("DELETE FROM tableros WHERE id = ?",
(self.tablero_actual_id,))
        conn.commit()
        conn.close()
        self.tablero_actual_id = None
        self.cargar_tableros()

def cargar_tableros(self):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT id, nombre FROM tableros WHERE usuario_id = ?",
(self.usuario_id,))
    self.tableros = cur.fetchall()
    conn.close()

    for widget in self.tablero_frame.wininfo_children():
        widget.destroy()

    if not self.tableros:
        tb.Label(self.tablero_frame, text="No hay tableros creados",
font=("Arial", 16), bootstyle="warning").pack(pady=20)
    return

```

```

        tb.Label(self.tablero_frame, text="Selecciona un tablero para
abrirlo:", font=("Arial", 18, "bold")).pack(pady=10)

        for tablero_id, nombre in self.tableros:
            tb.Button(self.tablero_frame, text=nombre, bootstyle="info
outline", width=30,
                        command=lambda tid=tablero_id, n=nombre:
self.mostrar_tablero(tid, n)).pack(pady=5)

    def mostrar_tablero(self, tablero_id, nombre):
        self.tablero_actual_id = tablero_id
        for widget in self.tablero_frame.winfo_children():
            widget.destroy()

        # Frame con título y botones de tablero
        top_tablero_frame = tb.Frame(self.tablero_frame)
        top_tablero_frame.pack(fill=X, pady=5)

        # Nombre tablero en verde oscuro
        label = tb.Label(top_tablero_frame, text=f"📋 {nombre}",
font=("Arial", 20, "bold"), foreground="#006400") # dark green
        label.pack(side=LEFT)

        tb.Button(top_tablero_frame, text="✏️ Modificar Tablero",
bootstyle="warning",
                    command=lambda:
self.editar_nombre_tablero(tablero_id)).pack(side=LEFT, padx=5)
        tb.Button(top_tablero_frame, text="🗑️ Eliminar Tablero",
bootstyle="danger",
                    command=self.eliminar_tablero).pack(side=LEFT, padx=5)

        tb.Button(top_tablero_frame, text=f"📅 17 Filtrar tareas por fecha",
bootstyle="info",
                    command=self.filtrar_tareas_por_fecha).pack(side=RIGHT)

        listas_frame = tb.Frame(self.tablero_frame)
        listas_frame.pack(fill=BOTH, expand=YES)

```

```

        conn = get_connection()
        cur = conn.cursor()
        cur.execute("SELECT id, nombre FROM listas WHERE tablero_id = ?",
(tablero_id,))
        listas = cur.fetchall()
        conn.close()

        for lista in listas:
            self.crear_columna_lista(listas_frame, lista[0], lista[1])

            tb.Button(self.tablero_frame, text="+ Añadir lista",
bootstyle="success",
                        command=lambda:
self.agregar_lista(tablero_id)).pack(pady=10)

        def crear_columna_lista(self, parent, lista_id, nombre):
            frame = tb.Labelframe(parent, text=nombre, padding=10, width=220,
bootstyle="success")
            frame.pack(side=LEFT, padx=10, fill=Y)

            # Título personalizado con color rojo y tamaño 18
            tb.Label(frame, font=("Arial", 14, "bold"),
foreground="#FF0000").pack(pady=(0, 5))

            # Botones modificar y eliminar lista
            btns_frame = tb.Frame(frame)
            btns_frame.pack(fill=X, pady=5)
            tb.Button(btns_frame, text="✎ Modificar Lista",
bootstyle="warning",
                        command=lambda:
self.editar_nombre_lista(lista_id)).pack(side=LEFT, padx=2)
            tb.Button(btns_frame, text="🗑 Eliminar Lista", bootstyle="danger",
                        command=lambda:
self.eliminar_lista(lista_id)).pack(side=RIGHT, padx=2)

        conn = get_connection()
        cur = conn.cursor()
        cur.execute("SELECT id, titulo, descripcion, fecha, estado FROM
tarefas WHERE lista_id = ?", (lista_id,))
        tareas = cur.fetchall()
        conn.close()

```



```

for tarea in tareas:
    estado = "✅" if tarea[4] == "completada" else "❌"
    # Título tarea en azul oscuro + resto texto normal:
    text = f"{tarea[1]}\n{tarea[2]}\n📅 {tarea[3]}\n{estado}"

    # Usamos un frame para separar título y resto y personalizar:
    tarea_frame = tb.Frame(frame, padding=5)
    tarea_frame.pack(pady=5, fill=X)

    # Label título azul oscuro
    titulo_label = tb.Label(tarea_frame, text=tarea[1],
font=("Arial", 12, "bold"), foreground="#00008B", anchor=W, justify=LEFT)
    titulo_label.pack(fill=X)

    # Label descripción y fecha y estado en color neutro
    detalle_label = tb.Label(tarea_frame, text=f"{tarea[2]}\n📅 {tarea[3]}\n{estado}", anchor=W, justify=LEFT, wraplength=180)
    detalle_label.pack(fill=X)

    tarea_btns = tb.Frame(frame)
    tarea_btns.pack(fill=X, pady=2)

    tb.Button(tarea_btns, text="✅ Cambiar Estado",
bootstyle="secondary",
                command=lambda tid=tarea[0]:
self.cambiar_estado_tarea(tid)).pack(side=LEFT, padx=2)
    tb.Button(tarea_btns, text="✏️ Modificar Tarea",
bootstyle="warning",
                command=lambda tid=tarea[0]:
self.editar_tarea(tid)).pack(side=LEFT, padx=2)

    tb.Button(frame, text="✚ Añadir tarea", bootstyle="warning",
command=lambda: self.agregar_tarea(lista_id)).pack(pady=5)

def agregar_lista(self, tablero_id):
    nombre = simpledialog.askstring("Nueva Lista", "Nombre de la
lista:")
    if nombre:
        conn = get_connection()
        cur = conn.cursor()
        cur.execute("INSERT INTO listas (nombre, tablero_id) VALUES (?,
?)", (nombre, tablero_id))
        conn.commit()
        conn.close()

```

```

        self.mostrar_tablero(tablero_id,
self.obtener_nombre_tablero(tablero_id))

    def eliminar_lista(self, lista_id):
        if messagebox.askyesno("Eliminar Lista", "¿Eliminar esta lista y
todas sus tareas?"):
            conn = get_connection()
            cur = conn.cursor()
            cur.execute("DELETE FROM listas WHERE id = ?", (lista_id,))
            conn.commit()
            conn.close()
            self.recargar_tablero()

    def agregar_tarea(self, lista_id):
        def guardar():
            titulo = entry_titulo.get().strip()
            descripcion = entry_desc.get().strip()
            fecha = entry_fecha.get().strip()
            if titulo:
                conn = get_connection()
                cur = conn.cursor()
                cur.execute("INSERT INTO tareas (titulo, descripcion, fecha,
estado, lista_id) VALUES (?, ?, ?, ?, ?)",
                    (titulo, descripcion, fecha, "pendiente",
lista_id))
                conn.commit()
                conn.close()
                top.destroy()
                self.recargar_tablero()
            else:
                messagebox.showwarning("Campo vacío", "El título de la tarea
no puede estar vacío.")

        def cancelar():
            top.destroy()

        top = Toplevel(self)
        top.title("Nueva Tarea")
        top.geometry("300x350")

        tb.Label(top, text="Título de la tarea:").pack(pady=5)
        entry_titulo = tb.Entry(top)

```

```

entry_titulo.pack(pady=5)

tb.Label(top, text="Descripción:").pack(pady=5)
entry_desc = tb.Entry(top)
entry_desc.pack(pady=5)

tb.Label(top, text="Fecha (YYYY-MM-DD):").pack(pady=5)
entry_fecha = tb.Entry(top)
entry_fecha.pack(pady=5)

btn_frame = tb.Frame(top)
btn_frame.pack(pady=15)

tb.Button(btn_frame, text="💾 Guardar", bootstyle="primary",
command=guardar).pack(side=LEFT, padx=5)
tb.Button(btn_frame, text="❌ Cancelar", bootstyle="danger",
command=cancelar).pack(side=RIGHT, padx=5)

def editar_tarea(self, tarea_id):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT titulo, descripcion, fecha, estado FROM tareas
WHERE id = ?", (tarea_id,))
    tarea = cur.fetchone()
    conn.close()

    if not tarea:
        return

    def guardar():
        nuevo_titulo = entry_titulo.get().strip()
        nueva_desc = entry_desc.get().strip()
        nueva_fecha = entry_fecha.get().strip()

        if nuevo_titulo:
            conn = get_connection()
            cur = conn.cursor()
            cur.execute("UPDATE tareas SET titulo=?, descripcion=?,
fecha=? WHERE id=?",
                        (nuevo_titulo, nueva_desc, nueva_fecha, tarea_id))
            conn.commit()
            conn.close()

```

```

        top.destroy()
        self.recargar_tablero()
    else:
        messagebox.showwarning("Campo vacío", "El título de la tarea
no puede estar vacío.")

    top = Toplevel(self)
    top.title("Editar Tarea")
    top.geometry("300x300")

    tb.Label(top, text="Título:").pack(pady=5)
    entry_titulo = tb.Entry(top)
    entry_titulo.insert(0, tarea[0])
    entry_titulo.pack(pady=5, fill=X, padx=10)

    tb.Label(top, text="Descripción:").pack(pady=5)
    entry_desc = tb.Entry(top)
    entry_desc.insert(0, tarea[1])
    entry_desc.pack(pady=5, fill=X, padx=10)

    tb.Label(top, text="Fecha (YYYY-MM-DD):").pack(pady=5)
    entry_fecha = tb.Entry(top)
    entry_fecha.insert(0, tarea[2])
    entry_fecha.pack(pady=5, fill=X, padx=10)

    btn_frame = tb.Frame(top)
    btn_frame.pack(side=BOTTOM, pady=15, fill=X, padx=10)

    tb.Button(btn_frame, text="💾 Guardar", bootstyle="primary",
command=guardar).pack(side=LEFT, expand=True, fill=X, padx=5)
    tb.Button(btn_frame, text="❌ Cancelar", bootstyle="danger",
command=top.destroy).pack(side=RIGHT, expand=True, fill=X, padx=5)

    def editar_nombre_tablero(self, tablero_id):
        nombre_actual = self.obtener_nombre_tablero(tablero_id)
        nuevo_nombre = simplifiedialog.askstring("Editar Tablero", "Nuevo
nombre:", initialvalue=nombre_actual)
        if nuevo_nombre and nuevo_nombre.strip():
            conn = get_connection()
            cur = conn.cursor()
            cur.execute("UPDATE tableros SET nombre=? WHERE id=?",
(nuevo_nombre.strip(), tablero_id))
            conn.commit()
            conn.close()
            self.cargar_tableros()

```

```

def editar_nombre_lista(self, lista_id):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT nombre, tablero_id FROM listas WHERE id = ?",
(lista_id,))
    fila = cur.fetchone()
    conn.close()
    if not fila:
        return
    nombre_actual, tablero_id = fila
    nuevo_nombre = simpledialog.askstring("Editar Lista", "Nuevo
nombre:", initialvalue=nombre_actual)
    if nuevo_nombre and nuevo_nombre.strip():
        conn = get_connection()
        cur = conn.cursor()
        cur.execute("UPDATE listas SET nombre=? WHERE id=?",
(nuevo_nombre.strip(), lista_id))
        conn.commit()
        conn.close()
        self.mostrar_tablero(tablero_id,
self.obtener_nombre_tablero(tablero_id))

def cambiar_estado_tarea(self, tarea_id):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT estado FROM tareas WHERE id = ?", (tarea_id,))
    fila = cur.fetchone()
    if fila:
        nuevo_estado = "completada" if fila[0] == "pendiente" else
"pendiente"
        cur.execute("UPDATE tareas SET estado=? WHERE id=?",
(nuevo_estado, tarea_id))
        conn.commit()
        conn.close()
        self.recargar_tablero()

def recargar_tablero(self):
    if self.tablero_actual_id:
        nombre = self.obtener_nombre_tablero(self.tablero_actual_id)
        self.mostrar_tablero(self.tablero_actual_id, nombre)
    else:
        self.cargar_tableros()

def obtener_nombre_tablero(self, tablero_id):

```

```

        conn = get_connection()
        cur = conn.cursor()
        cur.execute("SELECT nombre FROM tableros WHERE id = ?",
(tablero_id,))
        fila = cur.fetchone()
        conn.close()
        return fila[0] if fila else ""

def filtrar_tareas_por_fecha(self):
    if not self.tablero_actual_id:
        return

    conn = get_connection()
    cur = conn.cursor()
    query = """
        SELECT t.id, t.titulo, t.descripcion, t.fecha, t.estado, l.id,
l.nombre
        FROM tareas t
        JOIN listas l ON t.lista_id = l.id
        WHERE l.tablero_id = ?
        ORDER BY date(t.fecha) ASC
    """
    cur.execute(query, (self.tablero_actual_id,))
    tareas = cur.fetchall()
    conn.close()

    for widget in self.tablero_frame.wininfo_children():
        widget.destroy()

    tb.Label(self.tablero_frame,
            text=f"Tareas ordenadas por fecha para el tablero:
{self.obtener_nombre_tablero(self.tablero_actual_id)}",
            font=("Arial", 16, "bold")).pack(pady=10)

    for tarea in tareas:
        tid, titulo, desc, fecha, estado, lista_id, lista_nombre = tarea
        estado_icon = "✅" if estado == "completada" else "❌"
        texto = f"[{lista_nombre}] {titulo}\n{desc}\n{
fecha}\n{estado_icon}"
        frame = tb.Frame(self.tablero_frame, padding=5,
bootstyle="info")
        frame.pack(fill=X, padx=10, pady=5)

```

```

        tb.Label(frame, text=texto, anchor=W,
justify=LEFT).pack(side=LEFT, fill=X, expand=YES)

        btns = tb.Frame(frame)
        btns.pack(side=RIGHT)

        tb.Button(btns, text="✓ Cambiar Estado", bootstyle="secondary",
        command=lambda tid=tid:
self.cambiar_estado_tarea(tid)).pack(side=LEFT, padx=2)
        tb.Button(btns, text="✎ Modificar Tarea", bootstyle="warning",
        command=lambda tid=tid:
self.editar_tarea(tid)).pack(side=LEFT, padx=2)

        tb.Button(self.tablero_frame, text="⬅ Volver al tablero",
bootstyle="info", command=self.recargar_tablero).pack(pady=10)

def buscar_tablero(self):
    texto = self.entry_búsqueda.get().strip()
    if not texto:
        self.cargar_tableros()
        return
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("SELECT id, nombre FROM tableros WHERE usuario_id = ?
AND nombre LIKE ?", (self.usuario_id, f"%{texto}%"))
    resultados = cur.fetchall()
    conn.close()

    for widget in self.tablero_frame.winfo_children():
        widget.destroy()

    if not resultados:
        tb.Label(self.tablero_frame, text="No se encontraron tableros.",
font=("Arial", 16), bootstyle="warning").pack(pady=20)
        return

    for tablero_id, nombre in resultados:

```

```
        tb.Button(self.tablero_frame, text=nombre, bootstyle="info
outline", width=30,
                    command=lambda tid=tablero_id, n=nombre:
self.mostrar_tablero(tid, n)).pack(pady=5)
```

11 - ANEXO 2: OPINIÓN PERSONAL SOBRE LA FCT

Mi experiencia en la FCT ha sido muy enriquecedora tanto a nivel profesional como personal. Durante este período, he podido poner en práctica los conocimientos teóricos adquiridos en clase y, al mismo tiempo, aprender nuevas metodologías, herramientas y dinámicas propias del entorno laboral real.

Uno de los aspectos más valiosos ha sido el aprendizaje mediante la observación y la práctica directa. He colaborado en tareas reales, he resuelto problemas cotidianos y me he adaptado a los tiempos y exigencias del sector. Esto me ha ayudado a desarrollar habilidades como el trabajo en equipo, la comunicación profesional, la responsabilidad y la gestión del tiempo.

Además, he tenido la oportunidad de recibir consejos y orientación de profesionales con experiencia, lo que me ha motivado a seguir formándome y mejorando como futuro técnico en desarrollo de aplicaciones multiplataforma. También he comprendido la importancia de estar actualizado en tecnologías emergentes y de continuar aprendiendo de forma autónoma.

En definitiva, considero que la FCT es una parte fundamental del ciclo formativo, ya que facilita una transición realista y progresiva hacia el mundo laboral. Me siento agradecido por la oportunidad y satisfecho con todo lo que he aprendido y vivido durante esta etapa.

Elaborado por:

María Fernanda Espinoza
Fátima Esmeralda Valle Argueta

Fecha de entrega:

13/06/2025

Email:

esmeraldaargueta2006@gmail.com



