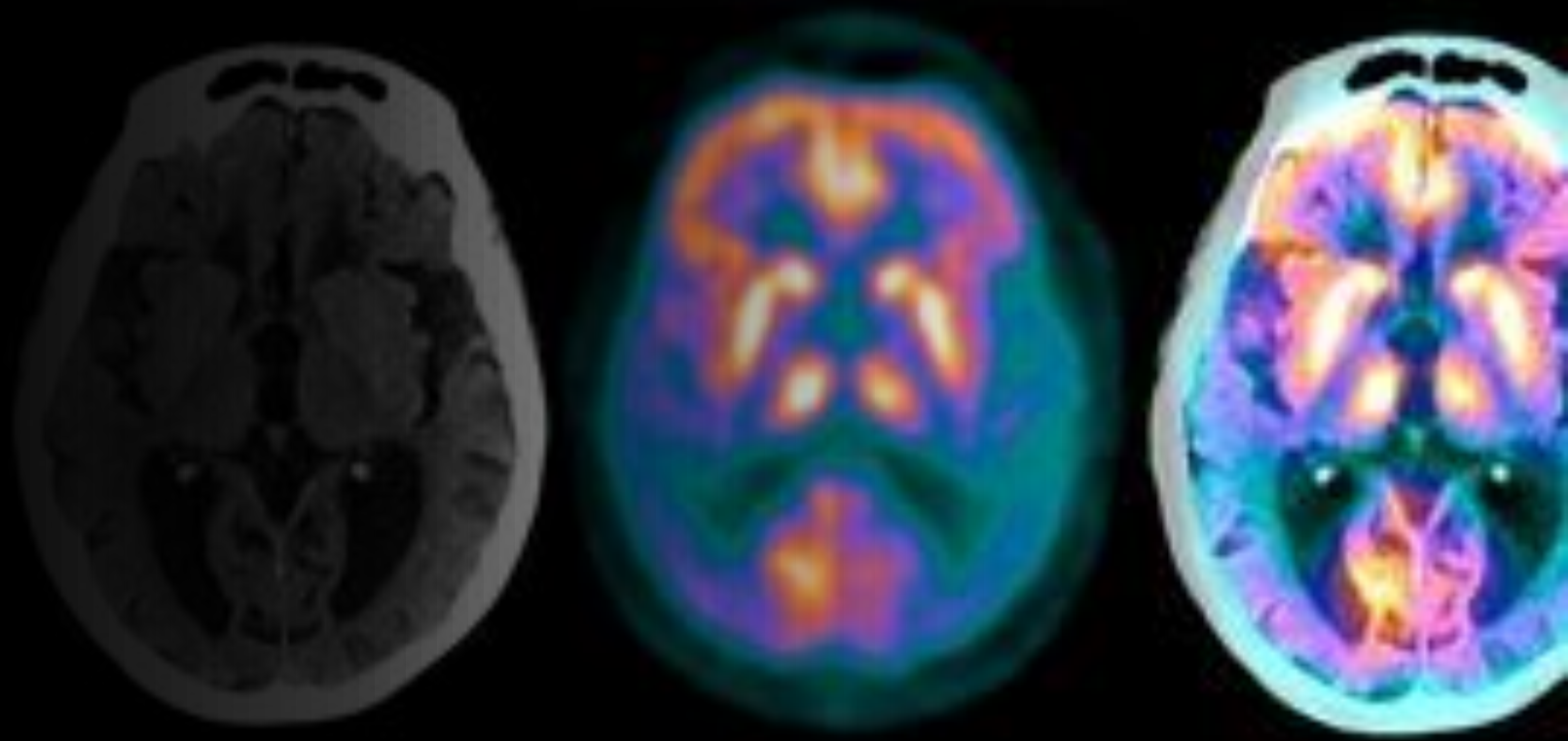
Memoria fallando Caso de Amador

Integrantes: Aidan Reid
Cristián Contreras
Martín López
Maximiliano Saavedra
Sara Román

Asignatura: Programación Orientada a Objetos

Profesora: Maria Bozo Álbarez

Fecha: 02/12/2025



Caso

Amador tiene 74 años vive solo y ha empezado a exprimentar fallos de memoria: deja la estufa encendida, olvida tomar sus medicamentos o no recuerda si ya comio. Aunque su familia este preocupada, el quiere manter su independiencia el mayor tiempo posible.

Objetivo

La propuesta planteada es crear una aplicación a través de python que ayude con las tareas diarias y necesarias para el día a día de Amador, tales como almorzar, ducharse, tomar el medicamento, entre otros. Proponemos una que la app funcione tipo calendario, siguiendo un horario establecido por un tercero para que el usuario pueda seguirla completamente, con un sistema de alarmas y un registro de tareas.

Main

- Inicia la aplicación.
- Crea el Controlador.
- Carga la primera pantalla.
- Ejecuta el ciclo principal

main.py | Configuración | Modelo.py | Controlador.py | Interfaz.py | calendario.py

C: > Users > Lenovo > Desktop > ULS 2°Semestre 2025 > POO > Proyecto 3 > main.py > ...

```python
"""Main solo para mostrar la pantalla del usuario usando Tkinter."""

from Controlador import Controlador

def main():
    # Usar el controlador para que la interfaz se conecte al modelo y lea el .json
    controlador = Controlador()
    controlador.vista.cambiar_pantalla("InicioScreen")
    controlador.ejecutar()

if __name__ == "__main__":
    main()
```

Modelo

- Guarda y carga los recordatorios desde JSON.
- Crea, modifica y elimina los recordatorios.
- Marca tareas como completadas.
- Maneja: diario, semana y mes
- Registra las acciones en el historial

main.py | Configuración | Modelo.py | Controlador.py | Interfaz.py | cale

C: > Users > Lenovo > Desktop > ULS 2°Semestre 2025 > POO > Proyecto 3 > Modelo.py > ...

```python
"""Modelo de la aplicación de recordatorios, sin dependencias de Kivy."""

import json
import os
from datetime import datetime
import sys

class MemoriaModelo:
    """Lógica y gestión de datos para la aplicación de recordatorios."""

    def __init__(self):
        # Siempre busca el recordatorios.json en la carpeta del proyecto
        base_dir = os.path.dirname(os.path.abspath(__file__))
        self.data_path = os.path.join(base_dir, 'recordatorios.json')
        # Filtros
        self.filtro_realizadas = 'prioritarias'
        self.filtro_dia_realizadas = 'todos'
        self.filtro_semana_realizadas = 'todas'
        self.filtro_mes_realizadas = 'todos'
        self._rec_edit_id = None

    def cargar_todos(self):
        if not os.path.exists(self.data_path):
            return []
        try:
            with open(self.data_path, 'r', encoding='utf-8') as f:
                return json.load(f)
        except Exception:
            return []

    def guardar_recordatorio(self, rec):
        datos = self.cargar_todos()
        # Si tienes algo como esto, coméntalo:
        # for r in datos:
```

Interfaz

Modelo.py | Controlador.py | Interfaz.py | calendario.py | Confianza en el área de trabajo

enovo > Desktop > ULS 2°Semestre 2025 > POO > Proyecto 3 > Interfaz.py > ...

```python
import tkinter as tk
from tkinter import ttk
from tkinter import font as tkfont, messagebox, simpledialog
import os
import hashlib

# Medisafe-inspired palette
BG = "#f4f8fb"          # app background (very light)
PRIMARY = "#2b87f0"     # primary blue-ish
ACCENT = "#14c1a3"      # accent (teal)
CARD = "#ffffff"         # card background
TEXT = "#0f1722"        # primary text
MUTED = "#6b7280"       # muted text
SHADOW = "#e6eefc"      # subtle shadow toner

# Widgets personalizados
def dark_label(parent, text, size=14, bold=False):
    font = ("Segoe UI", size, "bold" if bold else "normal")
    return tk.Label(parent, text=text, fg=TEXT, bg=BG, font=font)

def info_label(parent, text, size=12):
    return tk.Label(parent, text=text, fg=MUTED, bg=BG, anchor="w", font=("Segoe UI", size))

def colored_button(parent, text, command=None, size=14, icon=None):
    btn_text = f"{icon} {text}" if icon else text
    return ttk.Button(parent, text=btn_text, command=command, style="TButton")

def day_toggle(parent, text, var):
    return tk.Checkbutton(parent, text=text, fg=PRIMARY, bg=BG, selectcolor=SHADOW, variable=var, font=("Segoe UI", 10, "bold"))


def header(parent, title=None, subtitle=None):
    """Create a Medisafe-like header with a logo and optional subtitle."""
    hdr = tk.Frame(parent, bg=BG)
```

- Muestra las pantallas de la aplicación.
- Captura las acciones del usuario
- Envia las acciones al Controlador.
- Actualiza la UI según lo que diga el Controlador.
- Organiza las pantallas como inicio: usuario y administrador, etc

Controlador

- Conectar la Vista con el Modelo.
- Procesar las acciones del usuario.
- Validar la información ingresada.
- Actualizar datos en el Modelo.
- Refrescar las pantallas con los datos correctos.
- Gestionar el rol Usuario / Administrador.

main.py | Configuración | Modelo.py | Controlador.py | Interfaz.py | calendario.py | Confianza en el

C: > Users > Lenovo > Desktop > ULS 2°Semestre 2025 > POO > Proyecto 3 > Controlador.py > Controlador

```python
from Modelo import MemoriaModelo
from Interfaz import App
import os

class Controlador:
    def __init__(self):
        self._asegurar_archivo_json()
        self.modelo = MemoriaModelo()
        self.vista = App(controlador=self)
        # Al iniciar, muestra los pendientes si está en UsuarioScreen
        self._asegurar_archivo_historial()
        self.rol_actual = "Usuario"  # por defecto
        self.actualizar_pendientes_usuario()

    def _asegurar_archivo_json(self):
        base_dir = os.path.dirname(os.path.abspath(__file__))
        data_path = os.path.join(base_dir, 'recordatorios.json')
        if not os.path.exists(data_path):
            with open(data_path, 'w', encoding='utf-8') as f:
                f.write('[]')

    # Métodos para conectar eventos de la vista con el modelo
    def leer_pendientes(self):
        self.actualizar_pendientes_usuario()

    def actualizar_pendientes_usuario(self):
        # Obtiene los recordatorios no completados y no eliminados SOLO para hoy
        from datetime import datetime
        hoy = datetime.today().date()
        pendientes = [
            r for r in self.modelo.cargar_todos()
            if not r.get('completado') and not r.get('eliminado') and r.get('fecha') == hoy.strftime("%Y-%m-%d")
        ]
        frame = self.vista.frames.get("UsuarioScreen")
```

Calendario

main.py | Configuración | Modelo.py | Controlador.py | Interfaz.py | calendario.py

C: > Users > Lenovo > Desktop > ULS 2°Semestre 2025 > POO > Proyecto 3 > calendario.py > ...

```python
import os
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
from datetime import datetime, timedelta
import tkinter.font as tkfont
from PIL import Image, ImageTk
import calendar

# Paleta Medisafe (igual que Interfaz.py)
BG = "#f4f8fb"
PRIMARY = "#2b87f0"
ACCENT = "#14c1a3"
CARD = "#ffffff"
TEXT = "#0f1722"
MUTED = "#6b7280"
SHADOW = "#e6eefc"

# Solo importa GestorEventos y Evento si están disponibles (para modo GUI)
try:
    from Modelo import GestorEventos, Evento
except ImportError:
    GestorEventos = None
    Evento = None

class Calendario:
    def __init__(self):
        self.eventos = {}
        self.root = None
        self.celdas = {}
        self.fecha_actual = datetime.today().replace(day=1)

    def mostrar_calendario_gui(self):
        self.root = tk.Tk()
        self._configurar_ventana()
```

- Muestra un calendario interactivo
- Permite elegir una fecha con doble clic
- Entrega la fecha seleccionada a Interfaz o a Controlador.
- Navega entre meses de forma facil.

Json y historial

JSON: Guarda las acciones de usuario para ser mostradas en la APP

Historial: Resgistra las acciones de usuario y adminastrador en un archivo.txt

Conclusion

Se ocupo la arquitectura de MCV.
Se diferencia entre usuario y administrador.
Se decidio ocupar una mejor interfaz y mas intuitiva.
Funciona el sistema de recordatorios y se puede agregar otras funciones.