DOCUMENTO 2: INFORME TÉCNICO

Informe Técnico del Proyecto

Proyecto: Gestor de Torneos con Python y PySide6 **Fecha:** 29 de Enero de 2026

1. Estructura del Proyecto

El software sigue el patrón de arquitectura **MVC (Modelo-Vista-Controlador)** para asegurar la escalabilidad y el mantenimiento del código.

Desglose de Directorios

- **/Controllers**: Contiene la lógica de negocio.
    - main_controller.py: Orquestador principal. Gestiona la navegación y la carga de recursos.
    - calendario_controller.py: Gestiona la lógica compleja del torneo (algoritmos de cruces, validación de fechas y gestión de resultados).
    - equipos_controller.py y participantes_controller.py: Controladores CRUD.
- **/Views**: Contiene la interfaz gráfica (main_window.py) generada a partir de diseños .ui.
- **/Resources**: Almacena recursos estáticos (imágenes img/ y estilos qss/).
- **/Models**: *Nota Arquitectónica:* Siguiendo los requisitos de modularidad, esta carpeta no contiene el código fuente de la base de datos. El modelo se ha externalizado (ver punto 2).

2. Decisiones Técnicas Relevantes

2.1. Modularidad y Librería Externa (Acceso a Datos)

Para cumplir con el requisito de *"no incluir el módulo de base de datos en el proyecto"*, se implementó una arquitectura de librería instalable.

- **Implementación:** Se creó el paquete torneo_db con su propio setup.py.
- **Ventaja:** Permite desacoplar totalmente la persistencia de datos de la interfaz gráfica.
- **Persistencia:** Se utiliza la librería estándar pathlib para localizar el directorio de usuario (home) y crear allí la carpeta de datos (TorneoFutbolData). Esto evita problemas de permisos en Program Files y asegura que los datos sobrevivan a reinstalaciones.

2.2. Empaquetado y Distribución (.exe)

Se utilizó **PyInstaller** para generar el ejecutable.

- **Gestión de Rutas (sys._MEIPASS):** Uno de los desafíos fue manejar las rutas de recursos (imágenes y estilos) que cambian cuando la aplicación se ejecuta como un archivo único (.exe). Se implementó una función obtener_ruta_recurso() en el controlador principal que detecta dinámicamente si la aplicación corre en modo *script* o en modo *frozen* (compilado), ajustando las rutas automáticamente.

2.3. Lógica del Torneo

En lugar de utilizar una estructura de base de datos rígida para las rondas, se optó por un enfoque dinámico:

- El sistema evalúa el número de equipos disponibles al inicio.
- Calcula automáticamente la fase de inicio (Octavos, Cuartos, etc.).
- Los emparejamientos se realizan mediante algoritmos aleatorios (random.shuffle), asegurando imparcialidad.
- La clasificación no se almacena, se **calcula** en tiempo real consultando el historial de partidos, lo que garantiza que siempre esté sincronizada con los marcadores.

3. Solución de Problemas (Troubleshooting)

Durante el desarrollo se abordaron varios retos técnicos:

- **Sincronización de Estadísticas:** Se detectó que al sumar goles a un jugador individual, el marcador global no se actualizaba. **Solución:** Se implementó un sistema de retorno de valores en el diálogo modal de jugadores, forzando la actualización de la UI principal al cerrarse la ventana secundaria.
- **Interfaz Táctil/Accesible:** Los controles numéricos estándar eran pequeños. **Solución:** Se crearon componentes personalizados (CustomSpinBox) con botones de gran tamaño para facilitar la introducción de resultados.
- **Error de Librería Externa en Ejecutable:** PyInstaller no detectaba automáticamente la librería torneo_db al estar fuera del proyecto. **Solución:** Se configuró el proceso de compilación (build) utilizando el parámetro --hidden-import y --paths para vincular explícitamente el módulo externo.

4. Tecnologías Utilizadas

- **Lenguaje:** Python 3.10+
- **Framework GUI:** PySide6 (Qt for Python)
- **Base de Datos:** SQLite
- **Empaquetado:** PyInstaller