Documento de Justificación de Diseño

1. Arquitectura General y Tecnologías Utilizadas

La implementación de la User Story se dividió en dos componentes principales: componente de presentación, Frontend y componente de servicios, Backend. Interactuando mediante API REST manteniendo una separación clara de responsabilidades entre la interfaz de usuario y la lógica de negocio.

- **Backend:** se desarrolló utilizando Python, con los frameworks Flask y Pytest.
    - **Flask** se utilizó para exponer los servicios a través de una API REST, debido a su simplicidad y facilidad de integración con el enfoque TDD.
    - **Pytest** se seleccionó como framework de testing automatizado, permitiendo escribir pruebas legibles y mantener una ejecución continua de validaciones durante las etapas del ciclo TDD.
    - **Python** por su sintaxis clara, la rapidez de desarrollo y el conocimiento previo del equipo, facilitando la aplicación disciplinada del enfoque Red-Green-Refactor.

- **Frontend:** se implementó con React, complementado con Bootstrap.
    - **React** fue elegido por su enfoque basado en componentes reutilizables y su eficiente manejo del estado, lo que simplifica la actualización dinámica de la interfaz según las acciones del usuario.
    - **Bootstrap** permitió garantizar un diseño responsive y una experiencia de usuario consistente sin invertir tiempo en detalles de estilado manual.

2. Aplicación del Ciclo TDD

Se aplicó el enfoque de **Desarrollo Conducido por Pruebas (TDD)** siguiendo las tres fases del ciclo:

1. **Red:** se escribieron pruebas unitarias que inicialmente fallaban al no existir aún la funcionalidad implementada.

2. **Green:** se desarrolló el código mínimo necesario para que las pruebas pasaran exitosamente.

3. **Refactor:** se mejoró la estructura del código, eliminando duplicaciones y aplicando buenas prácticas de diseño sin romper las pruebas existentes.

De esta forma, las pruebas sirvieron como especificación ejecutable del comportamiento esperado, garantizando que la funcionalidad cumpliera con los **criterios de aceptación** definidos en la User Story.

3. Decisiones tomadas:

- Datos: Al no obtener respuestas respecto a los horarios y días de actividades, los datos fueron mokeados. (como no está aclarado, de momento mostramos todos los días que tienen disponible X actividad, junto a sus horarios y cupos.)

- Persistencia de Datos: Los datos los manejamos en un archivo CSV, por motivos de simplicidad, siguiendo el siguiente formato:

  nombre, dia, horario, cupo_disponible, requiere_talle, visitantes

  - nombre: ..... Nombre de la actividad
  - día: ..... Dia de la actividad formato AAAA-MM-DD
  - horario: ..... Horario de la actividad
  - cupo_disponible: ..... Cantidad de cupos disponibles de la actividad
  - requiere_talle: ..... Valor Bool de si es necesario el talle del visitante
  - visitantes: ..... Lista de DNI de los visitantes inscriptos

- Diseño de Clases y Entidades: Actividad, GestorActividades y Visitante: Estas clases fueron creadas para representar de manera explícita las entidades del dominio de la aplicación.
  - **Actividad** contiene información de nombre, días, horarios y cupos.
  - **GestorActividades** maneja la lógica de inscripción, verificación de cupos y persistencia de datos.
  - **Visitante** representa a cada usuario que participa de una actividad, incluyendo datos como nombre, DNI, edad y talle si corresponde.

- API REST: El archivo app.py, ubicado en la carpeta /backend, expone los endpoints de la API REST que permiten acceder a los servicios del sistema. Cada endpoint corresponde a una operación sobre los recursos de la aplicación.

- Tests: el archivo test_tp.py dentro de las carpetas backend/tests/, contiene todos los tests que realizamos, incluyendo las pruebas de usuario.