## ***Introducción***

En esta segunda iteración del proyecto de un gestor de gastos, se continúa con el proceso de evolución y mejora de la aplicación, consolidando las bases necesarias para su crecimiento progresivo. Durante esta fase, se han incorporado nuevas funcionalidades orientadas a enriquecer la experiencia del usuario y ampliar la capacidad operativa del sistema.

Asimismo, se ha avanzado en la definición y adopción de una arquitectura de software adecuada, que permita una organización más clara del código y facilite su mantenimiento y escalabilidad. Otro aspecto abordado ha sido la clasificación y gestión de errores, lo cual busca mejorar la robustez del sistema, simplificar la identificación de fallos y brindar retroalimentación más útil al usuario. Estas acciones conjuntas no solo optimizan la iteración actual, sino que también allanan el camino para futuras fases del proyecto, promoviendo un desarrollo más eficiente, ordenado y sostenible.

Adicionalmente, se ha trabajado en el diseño y definición de los *endpoints* de la API REST que servirá como intermediario entre el frontend y el backend de la aplicación. Esta interfaz facilitará la comunicación entre los distintos módulos del sistema, permitiendo realizar operaciones como la creación, edición, consulta y eliminación de transacciones, objetivos y etiquetas de manera estructurada y eficiente.

## ***Objetivos***

Durante esta segunda iteración del desarrollo del gestor de gastos, se plantean los siguientes objetivos específicos:

1. **Incorporar nuevas funcionalidades** que amplíen las capacidades del sistema, respondiendo a las necesidades identificadas en la etapa anterior y mejorando la interacción del usuario con la aplicación.
2. **Definir y aplicar una arquitectura de software adecuada**, que permita una mayor escalabilidad, mantenibilidad y claridad en la estructura del código, facilitando futuras expansiones y mejoras del sistema.
3. **Establecer un sistema de clasificación y gestión de errores**, con el fin de detectar y manejar de manera efectiva los posibles fallos, brindando mensajes claros al usuario y fortaleciendo la estabilidad general de la aplicación.

4. **Preparar la base para futuras iteraciones**, asegurando que las decisiones tomadas en esta fase contribuyan a un desarrollo continuo más ordenado, eficiente y sostenible.
5. **Preparar la base para futuras iteraciones**, asegurando que las decisiones tomadas en esta fase contribuyan a un desarrollo continuo más ordenado, eficiente y sostenible.

***Elección de una Arquitectura***

En esta iteración se adoptó una **arquitectura por capas**, con el propósito de organizar el sistema de forma estructurada y facilitar su mantenimiento y evolución a largo plazo. Esta arquitectura divide la aplicación en diferentes niveles de responsabilidad, permitiendo separar claramente la lógica del negocio de la interfaz de usuario, y promover así un desarrollo más limpio, escalable y sostenible.

Actualmente, el sistema ya cuenta con una separación inicial entre las capas de presentación, lógica de negocio y acceso a datos. Sin embargo, como parte de la planificación para futuras iteraciones, estas capas se formalizarán de la siguiente manera:

1. **Capa de Presentación** : encargada de mostrar la interfaz visual al usuario e interpretar sus acciones (formularios, botones, navegación). Esta capa no contiene lógica de negocio ni accede directamente a los datos.

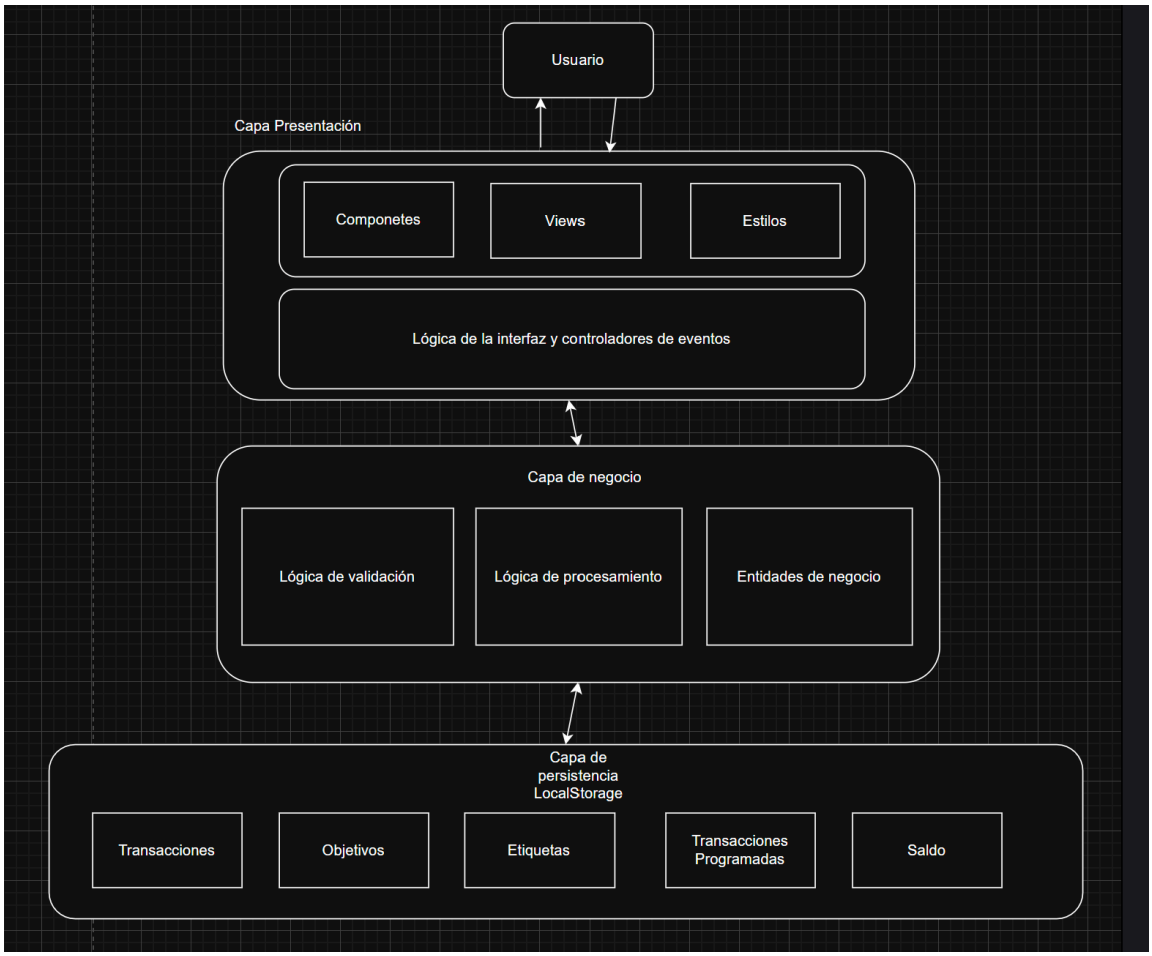
2. **Capa de Servicios o Lógica de Presentación** : constituida por funciones específicas que gestionan las peticiones HTTP hacia el backend, transforman los datos si es necesario y manejan errores de comunicación.

3. **Capa de Negocio** : responsable de procesar las peticiones recibidas, aplicar las reglas de negocio, validar datos y coordinar las operaciones con la base de datos. Es el núcleo lógico del sistema.

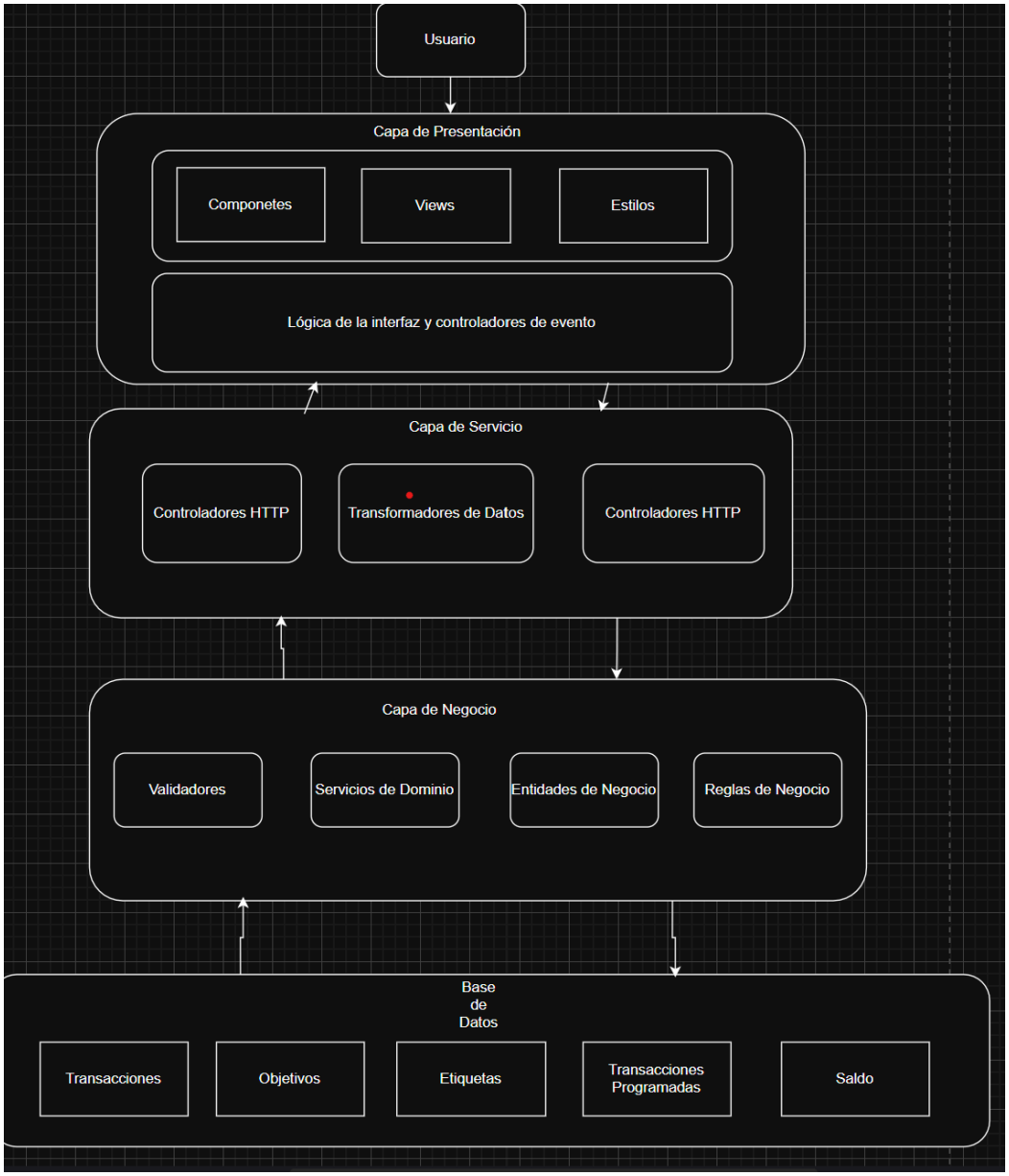
4. **Capa de Datos** : encargada del almacenamiento persistente, consulta, actualización y eliminación de la información, garantizando su integridad y disponibilidad controlada únicamente a través del backend.

La implementación de esta arquitectura requirió **una reestructuración del código existente**, reorganizando componentes y funciones para alinearlos con las nuevas responsabilidades definidas por capa. Este esfuerzo permitió establecer una base más sólida y modular que facilitará el desarrollo de nuevas funcionalidades en iteraciones futuras.

- Diagrama de la actual arquitectura de prueba:




- Diagrama de arquitectura futura:




## ***Definición de los Endpoints de la API REST***

A continuación, se presenta una tabla con los endpoints definidos para la API REST del sistema, organizados por entidad. Esta definición estandariza las operaciones de acceso y manipulación de datos, permitiendo una integración clara entre el frontend y el backend mediante métodos HTTP como GET, POST, PUT y DELETE.

| Método | Endpoint | Descripción | Respuesta esperada |
|---|---|---|---|
| GET | /api/goals | Obtener todos los objetivos | 200 OK – Lista de objetivos |
| POST | /api/goals | Crear un nuevo objetivo | 201 Created – Objetivo creado<br>400 Bad Request – Datos inválidos |
| PUT | /api/goals/:id | Actualizar un objetivo existente | 200 OK – Objetivo actualizado<br>404 Not Found – ID inexistente |
| GET | /api/goals?prop1=val1&prop2=val2 | Filtrar los objetivos por valores seleccionados. | 200 OK – Lista de objetivos filtrada. |
| DELETE | /api/goals/:id | Eliminar un objetivo | 204 No Content – Eliminación exitosa<br>404 Not Found – ID inexistente |
| GET | /api/transactions | Obtener todas las transacciones | 200 OK – Lista de transacciones |
| GET | /api/transactions?prop1=val1&prop2=val2 | Filtrar las transacciones por valores seleccionados. | 200 OK – Lista de transacciones filtrada. |
| POST | /api/transactions | Crear una nueva transacción | 201 Created – Transacción creada<br>400 Bad Request – Datos inválidos |
| PUT | /api/transactions/:id | Actualizar una transacción | 200 OK – Transacción actualizada<br>404 Not Found – ID inexistente |

| | | | |
|---|---|---|---|
| DELETE | /api/transactions/:id | Eliminar una transacción | 204 No Content – Eliminación exitosa<br>404 Not Found – ID inexistente |
| GET | /api/labels | Obtener todas las etiquetas | 200 OK – Lista de etiquetas |
| GET | /api/labels?prop1=val1&prop2=val2 | Filtrar las etiquetas por valores seleccionados. | 200 OK – Lista de etiquetas filtrada. |
| POST | /api/labels | Crear una nueva etiqueta | 201 Created – Etiqueta creada<br>400 Bad Request – Datos inválidos |
| PUT | /api/labels/:id/frecuencia | Actualizar la frecuencia de uso de una etiqueta | 200 OK – Frecuencia actualizada<br>400 Bad Request – Datos inválidos<br>404 Not Found – ID inexistente |
| DELETE | /api/labels/:id | Eliminar una etiqueta | 204 No Content – Eliminación exitosa<br>404 Not Found – ID inexistente |
| GET | /api/saldo | Obtener el estado actual del saldo | 200 OK – { "total": 1200, "guardadoObjetivos": [200, 100], "disponible": 900 } |
| POST | /api/saldo/sumar | Sumar un monto al saldo total | 200 OK – { "total": 1300, "disponible": 1000 }<br>400 Bad Request – inválido |

| | | | |
|---|---|---|---|
| POST | /api/saldo/restar | Restar un monto del saldo total (sin afectar lo guardado para objetivos) | 200 OK – { "total": 1100, "disponible": 800 }<br>400 Bad Request – insuficiente |
| POST | /api/saldo/reiniciar | Reinicia el saldo total y los montos guardados para objetivos | 200 OK – { "total": 0, "guardadoObjetivos": [], "disponible": 0 } |
| POST | /api/saldo/objetivo/{id}/guardar | Guardar un monto para un objetivo específico | 200 OK – Monto guardado correctamente<br>400 Bad Request – Datos inválidos o insuficiente disponible<br>404 Not Found – ID de objetivo no encontrado |
| POST | /api/saldo/objetivo/{id}/liberar | Liberar el monto guardado para un objetivo | 200 OK – Monto liberado correctamente<br>400 Bad Request – Datos inválidos<br>404 Not Found – ID de objetivo no encontrado |

| | | | |
|---|---|---|---|
| GET | /api/scheduled-transactions | Obtener todas las transacciones programadas | 200 OK – Lista de transacciones programadas |
| GET | /api/scheduled-transactions ?prop1=val1&prop2=val2 | Filtrar las transacciones programas por valores seleccionados. | 200 OK – Lista de transacciones programas filtrada. |

| | | | |
|---|---|---|---|
| POST | /api/scheduled-transactions | Crear una nueva transacción programada | 201 Created – Transacción programada creada<br>400 Bad Request – Datos inválidos |
| PUT | /api/scheduled-transactions/:id | Actualizar una transacción programada | 200 OK – Transacción programada actualizada<br>404 Not Found – ID inexistente |
| DELETE | /api/scheduled-transactions/:id | Eliminar una transacción programada | 204 No Content – Eliminación exitosa<br>404 Not Found – ID inexistente |

***Clasificación de errores***

Durante esta segunda iteración, se identificaron y corrigieron varios errores provenientes de la primera versión del sistema. Además, debido a la reestructuración del código para adaptarlo a la nueva arquitectura por capas, es posible que hayan surgido nuevos errores aún no detectados.

Por esta razón, se definió una clasificación de errores que servirá como guía para reaccionar de manera más efectiva ante futuros fallos. Esta clasificación permite organizar los errores según su origen (cliente, servidor, validación, comunicación, etc.) y facilita su detección, diagnóstico y resolución. Así, se contribuye a mantener la calidad del sistema y se optimiza el trabajo de mantenimiento y desarrollo continuo.

| **Gravedad** | **Descripción** | **Impacto** | **Acción requerida** |
|---|---|---|---|
| Crítica | No sé renderizar correctamente las transacciones en la lista. | No cumple con el requerimiento funcional de mostrar las transacciones realizadas. | Revisión y testeo urgente en la cajita de lógica y/ó testeo |

| | | | |
|---|---|---|---|
| Critico | Al editar una transacción no se guardan los valores correctamente, y altera la estructura interna del elemento almacenado en localStorage. | Al alterar la estructura interna del objeto, inutiliza toda la estructura lógica del programa afectando a todas las capas | Revisión en la lógica de persistencia. |
| Funcional | El sistema no muestra mensaje de error al ingresar nuevos valores inválidos al editar. | La verificación funciona correctamente, sin afectar los funcionalidades, pero puede generar confusión al usuario. | Revisión en la interacción de la lógica con la presentación. |
| Estico | Al mostrar el listado de las transacciones se muestran en orden de creación y no por su fecha. | Estético, no tiene impacto en las funcionalidades. | Corregir código del componente del renderizado. |
| Estético | Error en el estilo de los botones, y textos descentrados | Estético, no tiene impacto en las funcionalidades | Corrección no urgente, problema del frontEnd. |
| Estético | Asimetrías en el formulario | Estético, no tiene impacto en las funcionalidades | Corrección no urgente, problema del frontEnd. |