# Sistema de control de gastos

## Primera iteración

Este sistema surge como respuesta a las diversas problemáticas que enfrentan las personas al momento de controlar sus gastos y finanzas. Se identificaron distintas causas, entre ellas, el uso de métodos poco eficientes y anticuados como el bloc de notas del celular o computadora, libretas de papel, o simplemente la observación del saldo bancario. Estos métodos conllevan una serie de inconvenientes, tales como:

* Falta de registros precisos.
* Desconocimiento de los gastos acumulados.
* Dificultad para identificar hábitos de consumo.
* Imposibilidad de planificar y respetar un presupuesto mensual.

El objetivo principal del sistema será ofrecer una herramienta simple e intuitiva que permita a los usuarios registrar y categorizar sus gastos, visualizar estadísticas y planificar presupuestos, mejorando así el control de sus finanzas personales.

Tecnologías a utilizar para realizar el proyecto:

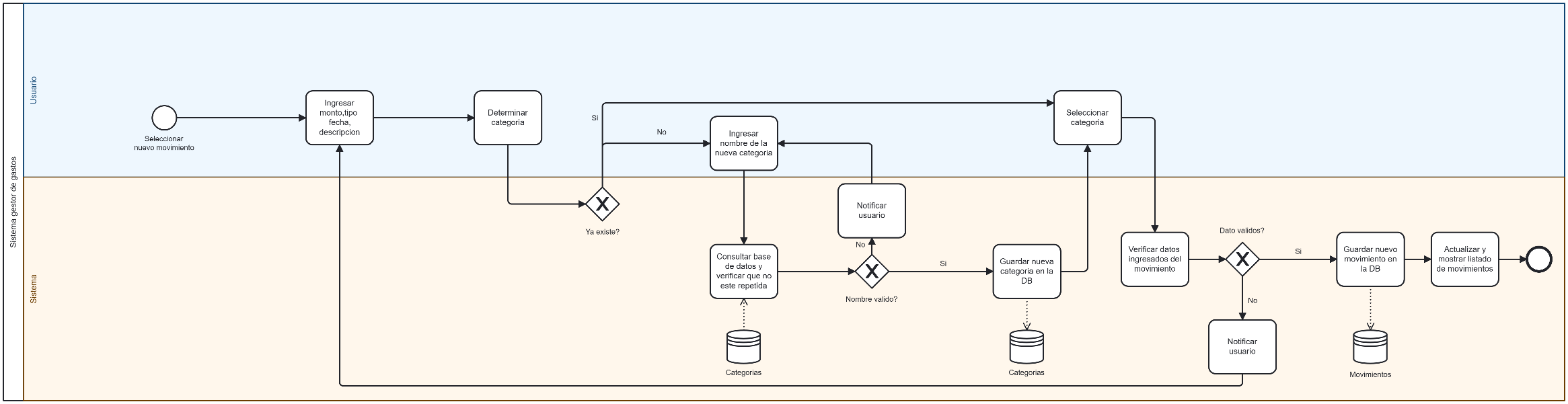
|  |  |
| --- | --- |
| Tecnología | ¿Para que se usa? |
| Visual Studio Code | Entorno de desarrollo principal, para escribir y organizar el código fuente del proyecto. |
| JavaScript | Lenguaje de programación principal que se va a usar para el desarrollo de la lógica del sistema. |
| JavaScrip con la biblioteca React | Tecnología utilizada para crear la interfaz web del sistema. |
| Vite | Herramienta usada para levantar el servidor de desarrollo de la app |
| Figma | Usada para el prototipado de la interfaz y verificación de los requerimientos. |
| StarUML | Programa utilizado para la realización de diagramas de casos de uso y de secuencia |
| Camunda Modeler | Programa utilizado para la elaboración de diagramas BPMN. |
| GitHub | Me permite crear un repocitorio de mi proyecto y presentarlo a otrar personas, y también llevar un control de los cambios em el código. |
| Recharts | Me permite la realización de gráficos para presentar en la página. |

**Requerimientos funcionales:**

1. El usuario debe poder ingresar un nuevo movimiento.
2. El usuario debe poder detallar en un movimiento el monto, tipo(ingreso/gasto), categoría, fecha y descripción.
3. El usuario debe poder crear una nueva categoría.
4. EL usuario debe poder eliminar/editar una categoría anteriormente creada.
5. El usuario debe poder editar los datos de un movimiento ya registrado.
6. El usuario debe poder eliminar un movimiento ya creado.
7. El sistema muestra una lista de todos los movimientos en orden cronológico.
8. El usuario puede filtrar la lista de movimientos por mes, tipo y categoría.
9. El sistema debe mostrar en el centro de control, para un mes determinado, el total de ingresos, gastos y el balance.
10. El usuario debe poder crear presupuestos mensuales y ser notificado cuando esté cerca de superarlo o lo haya superado.
11. El sistema debe poder generar y mostrar los siguientes gráficos de un periodo determinado:
    1. -Un grafico de barras que muestr el total de ingresos y gastos.
       * Un gráfico de torta que muestre la distribución de gastos por categoría.
12. El sistema debe generar y mostrar un gráfico de línea que represente el balance mensual (diferencia entre ingresos y gastos) a lo largo del tiempo.
13. Al agregarse/modificarse/ eliminarse un movimiento el sistema debe actualizar la lista de movimientos automáticamente.
14. El sistema debe guardar automáticamente todos los cambios y registros realizados por el usuario en la base de datos
15. El sistema al finalizar el mes debe restablecer los gastos e ingresos para iniciar el nuevo periodo.

**Procesos :**

**Ingresar un nuevo movimiento.**

****

**Generar gráficos:**

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Filtrar movimientos:**

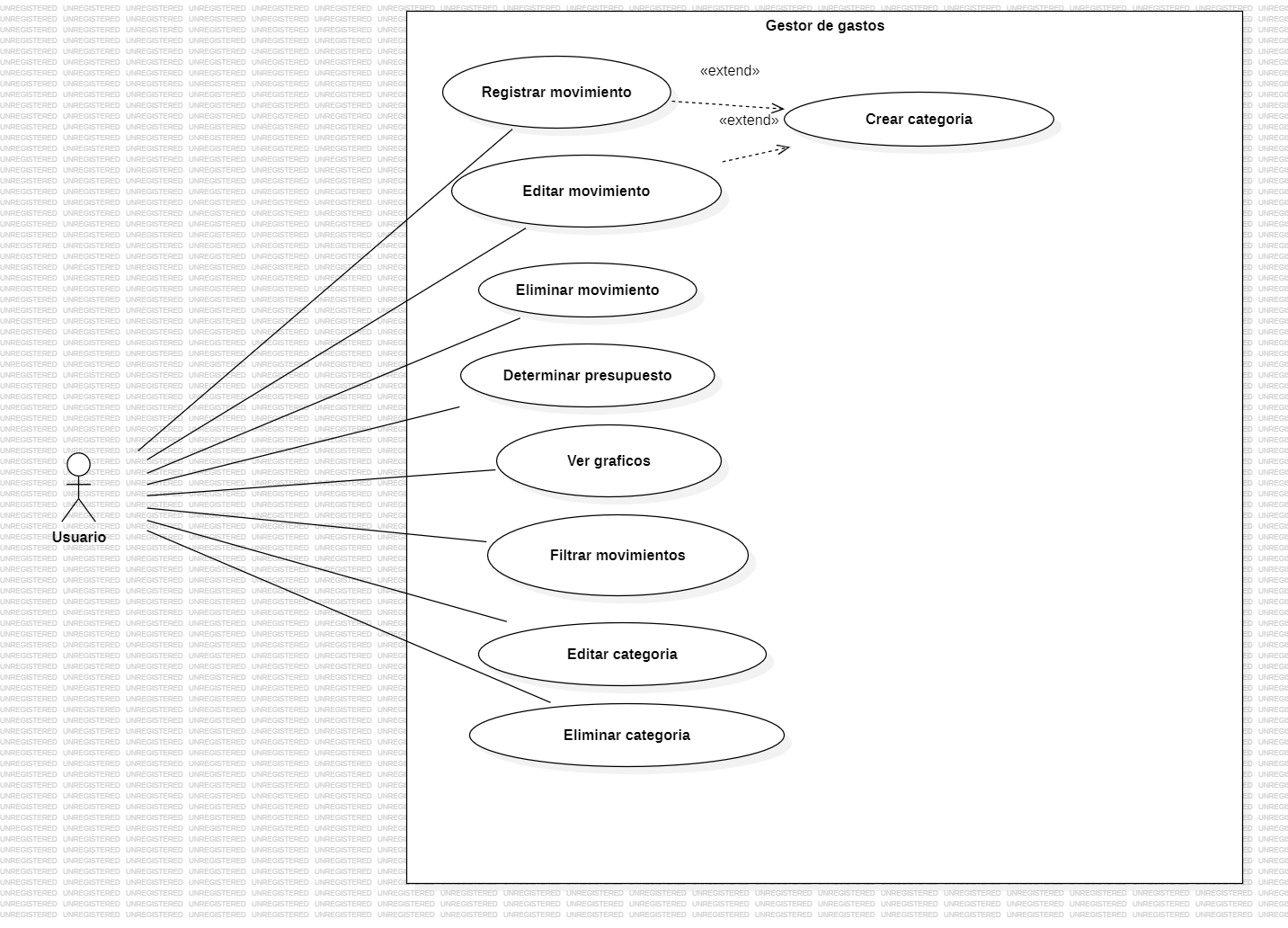
**Diagrama

El contenido generado por IA puede ser incorrecto.**

Crear presupuesto:

Diagrama

El contenido generado por IA puede ser incorrecto.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Ingresar movimiento | | | | |
| **ID: 1** | | | **Fecha:** | |
| **Descripción:** El usuario desea ingresar un nuevo movimiento detallando todos los datos | | | | |
| **Actores Principales**: Usuario | | **Actores Secundarios**: | | |
| **Observaciones:** | | | | |
| **Precondiciones**: - | | | | |
| **Post- Condiciones** | **Éxito:** Se crea un nuevo movimiento | | | |
| **Fracaso:** | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. **El caso de uso comienza cuando** el usuario desea ingresar un nuevo movimiento. | | | |  |
| 2. El usuario presiona en agregar nuevo movimiento. | | | |  |
| 3. El sistema le muestra el formulario y el usuario detalla del movimiento el tipo, monto, fecha, categoría y descripción. | | | | 3.1 El sistema le muestra el formulario y el usuario detalla del movimiento el tipo ,monto ,fecha.  3.2 EL usuario desea crear una categoría.  3.3 Se extiende al CU crear categoría |
| 4. El usuario presiona guardar movimiento. | | | |  |
| 5. Se validan los datos ingresados por el usuario y son correctos. | | | | 5.1 Se validad los datos ingresados por el usuario y no están correctos.  5.2 Se notifica al usuario que campo debe corregir.  5.3 El usuario corrige los datos y están correctos. |
| 6. Se guarda el nuevo movimiento en la base de datos y se confirma su guardado. | | | |  |
| 7. Se actualiza la lista de movimientos. | | | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Eliminar movimiento | | | | |
| **ID: 1** | | | **Fecha:** | |
| **Descripción:** El usuario desea eliminar un movimiento de la lista | | | | |
| **Actores Principales**: Usuario | | **Actores Secundarios**: | | |
| **Observaciones:** | | | | |
| **Precondiciones**: - | | | | |
| **Post- Condiciones** | **Éxito:** Se elimina con éxito el movimiento | | | |
| **Fracaso:** | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. **El caso de uso comienza cuando** el usuario desea eliminar un movimiento ya creado. | | | |  |
| 2. El usuario selecciona un movimiento y presiona el botón de eliminar. | | | |  |
| 3. El sistema solicita la confirmación | | | |  |
| 4. El usuario confirma la eliminación | | | | 4.1 El usuario cancela la confirmación.  4.2 Fin CU. |
| 5.El sistema elimina el movimiento de la base de datos y actualiza la lista de movimientos. | | | |  |
| 6. Fin CU. | | | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Crear nueva categoría | | | | |
| **ID: 1** | | | **Fecha:** | |
| **Descripción:** El usuario crea una nueva categoría personalizada. | | | | |
| **Actores Principales**: Usuario | | **Actores Secundarios**: | | |
| **Observaciones:** | | | | |
| **Precondiciones**: - | | | | |
| **Post- Condiciones** | **Éxito:** Se crea una nueva categoría y se puede reutilizar. | | | |
| **Fracaso:** | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. **El caso de uso comienza cuando**  el usuario desea crear una nueva categoría. | | | |  |
| **2.** En la sección categorías el usuario selecciona otro. | | | |  |
| 3. El sistema despliega una entrada de texto donde el usuario ingresa el nombre de la nueva categoría. | | | |  |
| 4. El sistema consulta la base de datos y valida que:  -El nombre no esté repetido (ya existente en las categorías del usuario).  - El nombre no supere los 30 caracteres. | | | |  |
| 5. Si la validación es exitosa se guarda y crea la nueva categoría y se guarda en la base de datos. | | | | 5.1 Si el nombre ya existe o excede el limite de caracteres:  - EL sistema muestra un mensaje de error indicando el motivo.  El usuario puede corregir el texto y se retorna al paso 4. |
| 6. El sistema muestra un mensaje de confirmación | | | |  |
| 7. Fin CU. | | | |  |

Requerimientos seleccionados para la primera iteración:

* El usuario debe poder ingresar un nuevo movimiento.
* El usuario debe poder detallar en un movimiento el monto, tipo(ingreso/gasto), categoría, fecha y descripción.
* El usuario debe poder eliminar un movimiento ya creado.
* El usuario debe poder crear una nueva categoría.

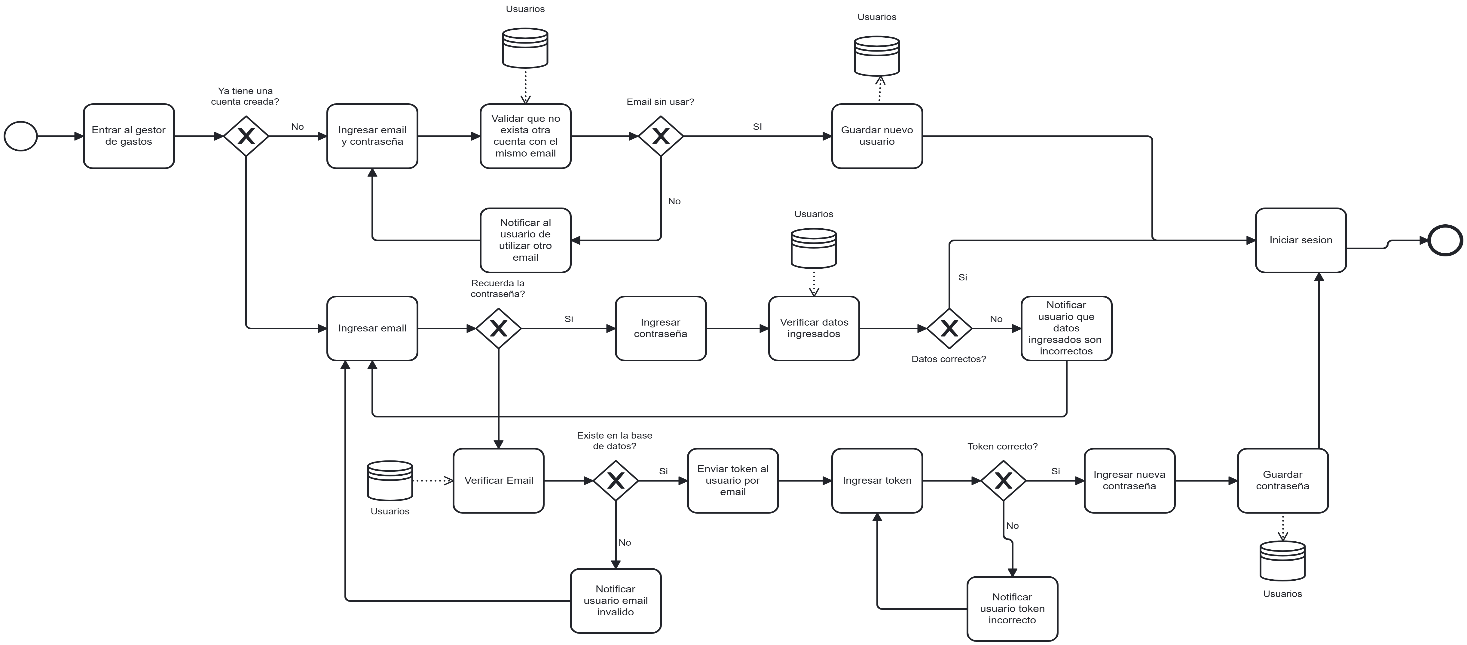
## Segunda Iteracion

### Requerimientos:

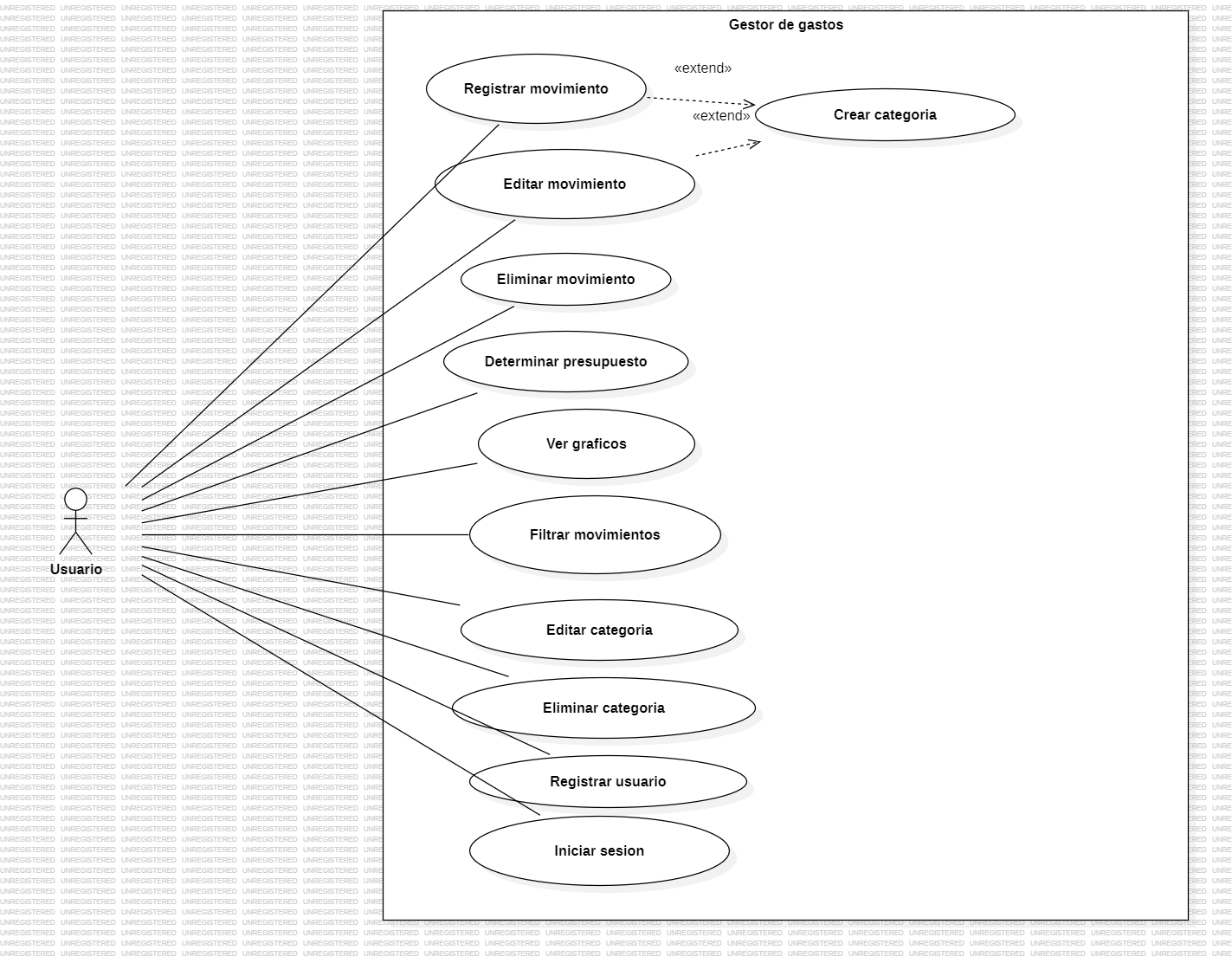
Nuevos requerimientos funcionales:

* El usuario debe poder crearse una cuenta para el gestor de gastos con su email y crear una contraseña.
* EL usuario debe poder ingresar a su cuenta con el email y contraseña anteriormente creada.
* El sistema debe verificar que no existan dos cuentas con el mismo email.
* El sistema debe permitir al usuario cambiar la contraseña si se la olvida.

### Diagrama BPMN:



### Diagrama casos de uso:



### Prototipo Iniciar sesion:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

### Descripcion de casos de uso:

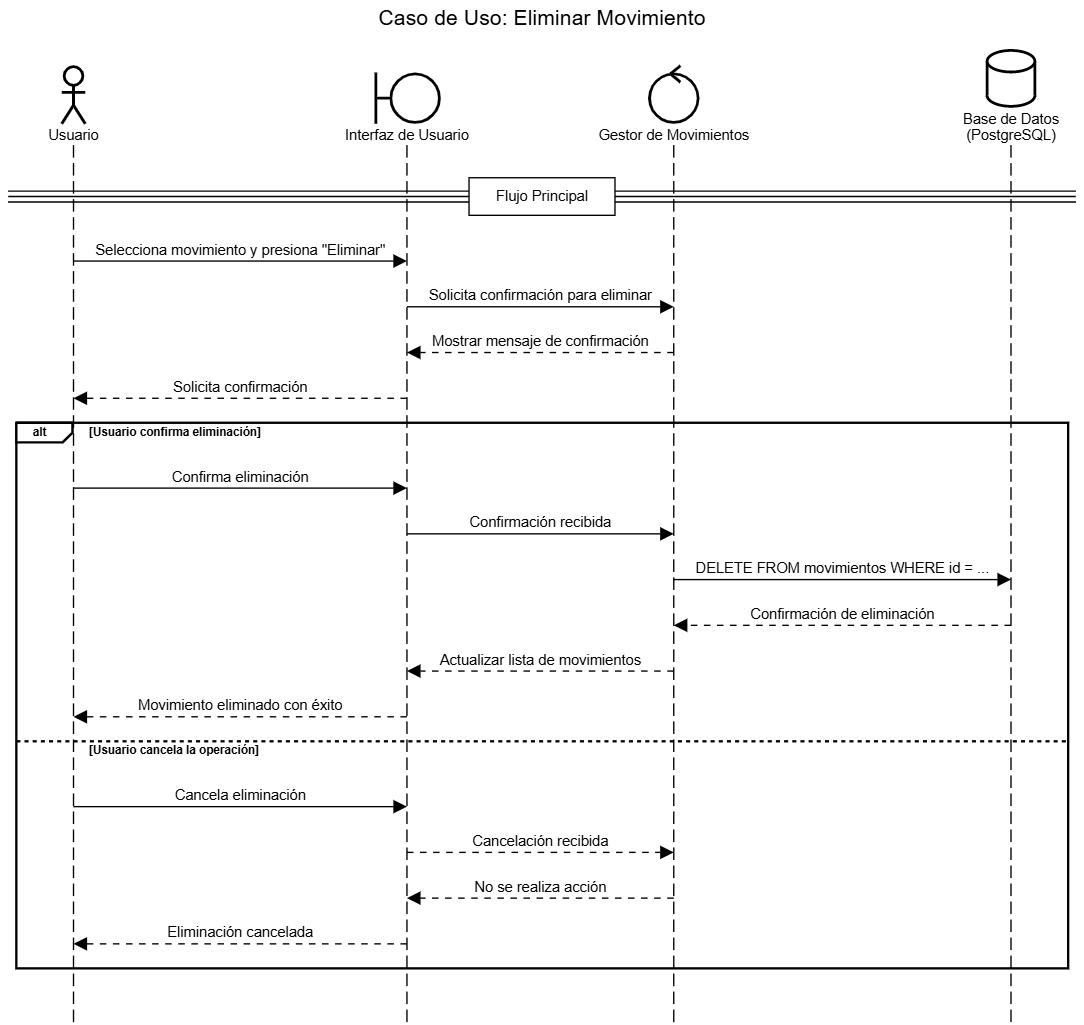
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Ver gráficos | | | | |
| **ID: 1** | | | **Fecha:** | |
| **Descripción:** Se le muestran los gráficos al usuario en la sección de gráficos | | | | |
| **Actores Principales**: Usuario | | **Actores Secundarios**: | | |
| **Observaciones:** | | | | |
| **Precondiciones**: - | | | | |
| **Post- Condiciones** | **Éxito:** Se muestran correctamente los gráficos al usuario. | | | |
| **Fracaso:** | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. **El caso de uso comienza cuando** el usuario ingresa al centro de control y presiona en el botón de gráficos. | | | |  |
| 1. El sistema obtiene con éxito de la base de datos los movimientos correspondientes al periodo corriente | | | | * 1. El sistema obtiene sin éxito los movimientos de la base de datos.   2. Se notifica al usuario   3. Fin CU. |
| 1. El sistema genera con éxito los gráficos y se los muestra al usuario. | | | | * 1. Ocurre un error y no se generan los gráficos.   2. Se notifica al usuario.   3. Fin CU. |
| 1. El usuario selecciona otro periodo para ver los gráficos | | | | * 1. El usuario no desea ver los gráficos de otro periodo   2. Fin CU. |
| 1. EL sistema obtiene con éxito de la base de datos los movimientos correspondientes al periodo seleccionado por el usuario. | | | | * 1. El sistema obtiene sin éxito los movimientos de la base de datos.   2. Se notifica al usuario   5.3 Fin CU. |
| 1. El sistema genera con éxito los gráficos y se los muestra al usuario. | | | | * 1. Ocurre un error y no se generan los gráficos.   2. Se notifica al usuario.   6.3Fin CU. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Crear Presupuesto | | | | |
| **ID: 1** | | | **Fecha:** | |
| **Descripción:** El usuario crea un nuevo presupuesto para el periodo corriente | | | | |
| **Actores Principales**: Usuario | | **Actores Secundarios**: | | |
| **Observaciones:** | | | | |
| **Precondiciones**: - | | | | |
| **Post- Condiciones** | **Éxito:** Se crea con éxito un nuevo presupuesto | | | |
| **Fracaso:** | | | |
| Flujo principal | | | | Flujo Alternativo |
| 1. **El caso de uso comienza cuando** el usuario desea determinar un presupuesto y presiona en el centro de control crear presupuesto. | | | |  |
| 1. El usuario ingresa el monto del presupuesto. | | | |  |
| 1. Se verifica el monto y es correcto. | | | | * 1. Se verifica el monto ingresado y no es correcto.   2. Se notifica al usuario y se le solicita volver a ingresar el monto. |
| 1. Se carga el nuevo presupuesto con éxito en la base de datos y se notifica al usuario. | | | | * 1. Ocurre un error en la carga del nuevo presupuesto en la base de dato y se lo notifica al usuario.   2. Fin CU |
| 1. El sistema inicia el seguimiento del presupuesto. | | | |  |

### Diagramas de secuencia:

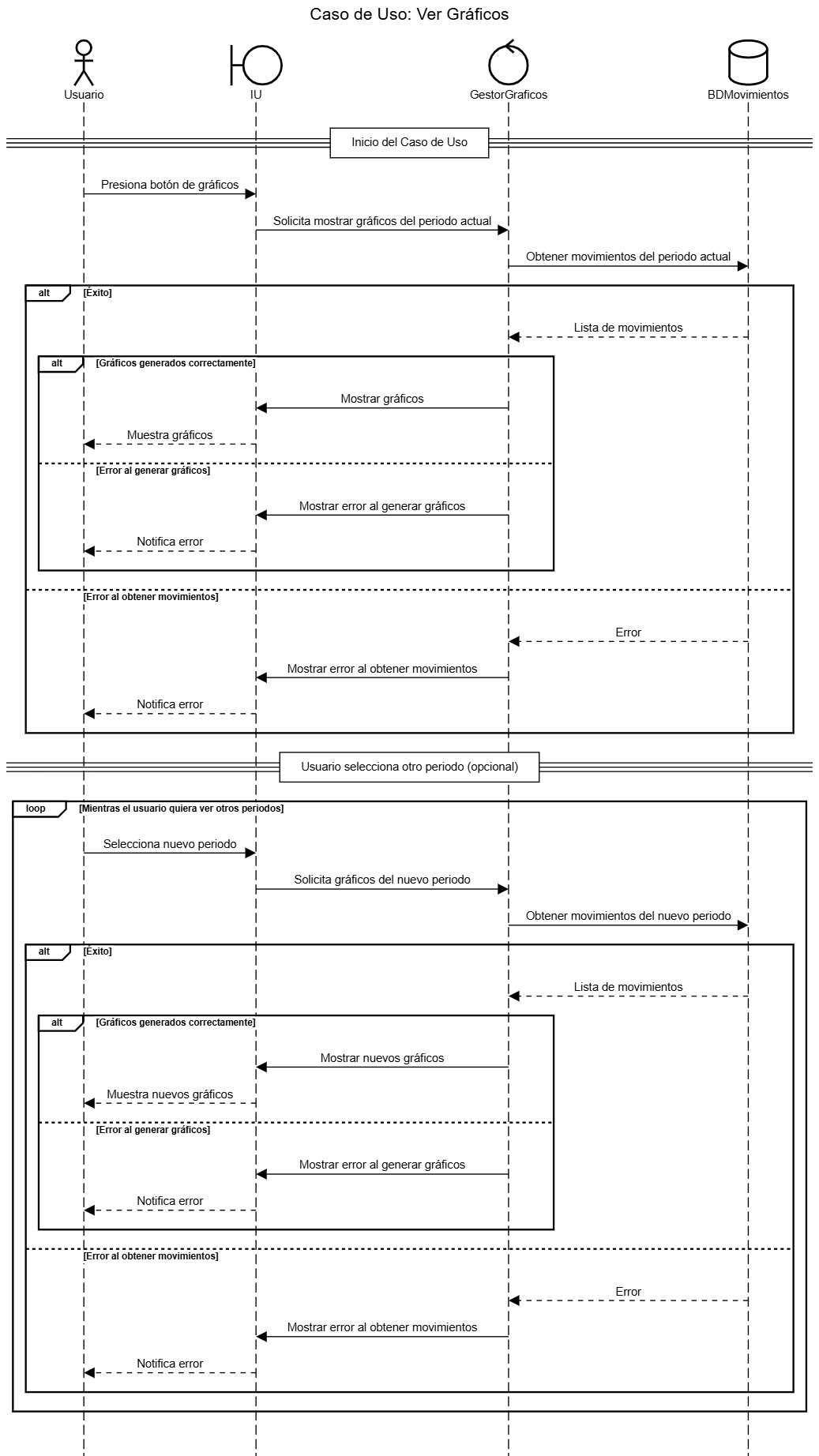
Diagrama

El contenido generado por IA puede ser incorrecto.



Tabla, Calendario

El contenido generado por IA puede ser incorrecto.



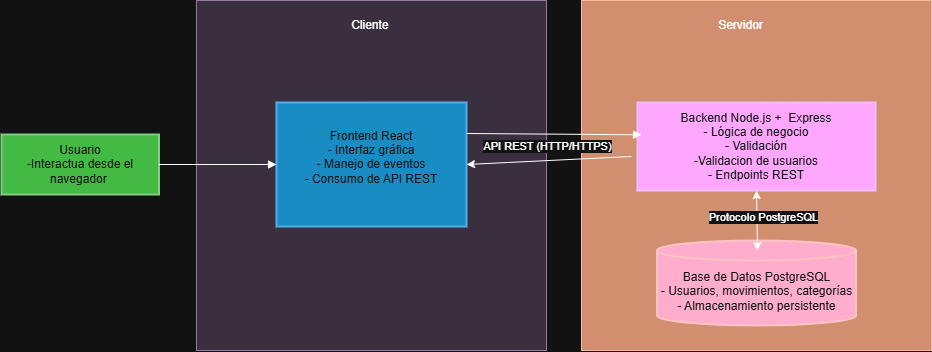
Diagrama

El contenido generado por IA puede ser incorrecto.

## Arquitectura

Para este proyecto se decidió utilizar una arquitectura cliente/servidor porque permite separar la parte visual del sistema (la interfaz) de la lógica y del manejo de los datos. El cliente, desarrollado con React, se encarga de mostrar la información y recibir las acciones del usuario. Por otro lado, el backend se desarrollara usando Node.js y Express para que el servidor procese esas acciones, realice las validaciones necesarias y guarde los datos . Para almacenar la información se va a usar PostgreSQL que es un sistema de gestión de bases de datos relacional, la cual permite guardar, organizar y consultar datos. Esta arquitectura ayuda a que el sistema sea más organizado, fácil de mantener y escalar si en el futuro quiero agregar nuevas funciones.

### Diagrama de la arquitectura



Componentes principales de la arquitectura:

Cliente (Frontend):

* Desarrollado en React.js
* Se encarga de la presentación de la información y la interacción con el usuario.
* Consume la API REST del backend para obtener y enviar datos.

Servidor (Backend)

* Desarrollado con Node.js y Express.
* Maneja la lógica de negocio, incluendo la autenticación de usuarios, el registro de movimientos, la gestión de categorías y la creación de presupuestos.
* Expone una API REST, compuesta por múltiples endpoints que permiten el acceso a la información del sistema.

Base de datos (persistencia de datos)

* Implementada en PostgreSQL.
* Estructurada con entidades como usuarios,movimientos,categorías y presupuestos.
* Relaciona los datos mediante claves foráneas, para garantizar la integridad de los datos.

Explicación del flujo de interacción:

1. El usuario accede a la interfaz web desde su navegador y realiza acciones como iniciar sesión, registrar gastos, definir presupuestos y visualizar gráficos.
2. El frontend envía una solicitud HTTP al backend, utilizando el método correspondiente. La solicitud incluye información necesaria en formato JSON.
3. El backend procesa la solicitud, validando los datos recibidos.
4. El backend consulta la base de datos en PostgreSQL para obtener o almacenar datos.
5. El backend genera una respuesta en formato JSON, conteniendo la información solicitada o el resultado de la operación, incluyendo los códigos de estado HTPP para indicar el resultado de la solicitud y lo envía al frontend.
6. El frontend recibe la respuesta y actualiza la interfaz.

### Listado de endpoints de la API REST:

|  |  |  |
| --- | --- | --- |
| Método | URL | Descripción |
| POST | |  | | --- | |  | | /api/auth/register | | Registrar un nuevo usuario en el sistema |
| POST | /api/auth/login | |  | | --- | | Iniciar sesión con email y contraseña | |  | |
| POST | /api/auth/reset-password | Solicitar el cambio de contraseña |
| GET | /api/movements | Obtener la lista de movimientos del usuario |
| POST | /api/movements | Registrar un nuevo movimiento |
| PUT | /api/movements/{id} | Editar un movimiento existente |
| DELETE | /api/movements/{id} | Eliminar un movimiento |
| GET | /api/budget | Obtener el presupuesto mensual del usuario |
| POST | /api/budget | Crear un nuevo presupuesto mensual |
| GET | /api/statistics/income-vs-expense | Obtener datos para grafico de ingresos y gastos |
| GET | /api/statistics/category-distribution | Obtener datos para grafico de distribución de gastos |
| GET | /api/statistics/monthly-balance | Obtener datos para grafico de evolución del balance mensual |
| GET | /api/categories | Obtener lista de categorías |
| POST | /api/categories | Crear nueva categoría |
| PUT | /api/categories/{id} | Editar una categoría existente |
| DELETE | /api/categories/{id} | Eliminar una categoría |

Requerimientos realizados en esta implementación:

RF7. El sistema muestra una lista de todos los movimientos en orden cronológico del mostrando el más reciente primero.

RF9: El sistema debe mostrar en el centro de control, para un mes determinado, el total de ingresos, gastos y el balance.

## Tercer iteración:

### Listado actualizado de endpoints de la API REST:

|  |  |  |
| --- | --- | --- |
| Método | URL | Descripción |
| GET | /api/auth/me | Obtener usuario identificado |
| POST | |  | | --- | |  | | /api/auth/register | | Registrar un nuevo usuario en el sistema |
| POST | /api/auth/login | |  | | --- | | Iniciar sesión con email y contraseña | |  | |
| POST | /api/auth/reset-password | Solicitar el cambio de contraseña |
| GET | /api/movements | Obtener la lista de movimientos del usuario |
| POST | /api/movements | Registrar un nuevo movimiento |
| PUT | /api/movements/{id} | Editar un movimiento existente |
| DELETE | /api/movements/{id} | Eliminar un movimiento |
| GET | /api/budget | Obtener el presupuesto mensual del usuario |
| POST | /api/budget | Crear un nuevo presupuesto mensual |
| GET | /api/statistics/income-vs-expense | Obtener datos para grafico de ingresos y gastos |
| GET | /api/statistics/category-distribution | Obtener datos para grafico de distribución de gastos |
| GET | /api/statistics/monthly-balance | Obtener datos para grafico de evolución del balance mensual |
| GET | /api/categories | Obtener lista de categorías |
| POST | /api/categories | Crear nueva categoría |
| PUT | /api/categories/{id} | Editar una categoría existente |
| DELETE | /api/categories/{id} | Eliminar una categoría |
| GET | |  | | --- | |  |  |  | | --- | | /api/dashboard | | |  | | --- | |  |  |  | | --- | | Obtener resumen mensual: total de ingresos, gastos y balance. | |

### Requerimientos Funcionales (actualizado):

**RF1.** El usuario debe poder ingresar un nuevo movimiento.

**RF2.** El usuario debe poder detallar en un movimiento el monto, tipo (ingreso/gasto), categoría, fecha y descripción.

**RF3.** El usuario debe poder crear una nueva categoría.

**RF4.** El usuario debe poder eliminar o editar una categoría anteriormente creada.

**RF5.** El usuario debe poder editar los datos de un movimiento ya registrado.

**RF6.** El usuario debe poder eliminar un movimiento ya creado.

**RF7.** El sistema muestra una lista de todos los movimientos en orden cronológico.

**RF8.** El usuario puede filtrar la lista de movimientos por mes, tipo y categoría.

**RF9.** El sistema debe mostrar en el centro de control, para un mes determinado, el total de ingresos, gastos y el balance.

**RF10.** El usuario debe poder crear presupuestos mensuales y ser notificado cuando esté cerca de superarlo o lo haya superado.

**RF11.** El sistema debe poder generar y mostrar los siguientes gráficos de un periodo determinado:  
a. Un gráfico de barras que muestre el total de ingresos y gastos.  
b. Un gráfico de torta que muestre la distribución de gastos por categoría.

**RF12.** El sistema debe generar y mostrar un gráfico de línea que represente el balance mensual a lo largo del tiempo.

**RF13.** Al agregarse, modificarse o eliminarse un movimiento, el sistema debe actualizar la lista de movimientos automáticamente.

**RF14.** El sistema debe guardar automáticamente todos los cambios y registros realizados por el usuario en la base de datos.

**RF15.** El sistema, al finalizar el mes, debe restablecer los gastos e ingresos para iniciar el nuevo periodo.

**RF16.** El usuario debe poder crearse una cuenta para el gestor de gastos con su email y contraseña.

**RF17.** El usuario debe poder ingresar a su cuenta con el email y contraseña previamente creados.

**RF18.** El sistema debe verificar que no existan dos cuentas con el mismo email.

**RF19.** El sistema debe permitir al usuario cambiar la contraseña si se la olvida.

### Implementación:

En esta iteración se realizó el desarrollo del backend y frontend del gestor de gastos aplicando las tecnologías mencionadas anteriormente. Se implementaron los siguientes requerimientos funcionales:

**RF1.** El usuario debe poder ingresar un nuevo movimiento.

**RF2.** El usuario debe poder detallar en un movimiento el monto, tipo (ingreso/gasto), categoría, fecha y descripción.

**RF3.** El usuario debe poder crear una nueva categoría.

**RF5.** El usuario debe poder editar los datos de un movimiento ya registrado.

**RF6.** El usuario debe poder eliminar un movimiento ya creado.

**RF7.** El sistema muestra una lista de todos los movimientos en orden cronológico.

**RF9.** El sistema debe mostrar en el centro de control, para un mes determinado, el total de ingresos, gastos y el balance.

**RF10.** El usuario debe poder crear presupuestos mensuales y ser notificado cuando esté cerca de superarlo o lo haya superado.

**RF13.** Al agregarse, modificarse o eliminarse un movimiento, el sistema debe actualizar la lista de movimientos automáticamente.

**RF16.** El usuario debe poder crearse una cuenta para el gestor de gastos con su email y contraseña.

**RF17.** El usuario debe poder ingresar a su cuenta con el email y contraseña previamente creados.

**RF18.** El sistema debe verificar que no existan dos cuentas con el mismo email.

# Cuarta iteración

## Requerimientos no funcionales

**RNF1: EL Sistema debe estar protegido a accesos no autorizados a rutas privadas.**

**Descripción**:  
El sistema debe garantizar que toda solicitud a rutas protegidas del backend sea accesible únicamente por usuarios autenticados mediante un token JWT válido.  
En caso de que un usuario no autenticado (o con un token inválido) intente acceder a una de estas rutas, el sistema debe **rechazar la solicitud**, devolver un **mensaje de error apropiado** y **evitar el acceso a cualquier dato sensible**.

Este requisito no funcional tiene como objetivo **asegurar la confidencialidad de los datos del usuario**, evitando accesos no autorizados a información privada.  
Corresponde a la **subcaracterística de Confidencialidad**, incluida dentro de la **característica de Seguridad**, según lo establecido en la norma **ISO 25010**.

**Escenario de calidad:**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Implementación:**

El usuario inicia sesión → el backend genera un **token JWT**.

El frontend guarda el token y lo envía en cada solicitud protegida (header Authorization).

El backend usa un **middleware verificarToken**:

* Si el token es válido → permite el acceso.
* Si falta o es inválido → responde con **401 Unauthorized**.

Se protegen rutas sensibles como /api/dashboard, evitando accesos no autorizados.

**Objetivo**: verificar que el backend rechace toda petición a una ruta protegida si no se incluye el token JWT.

**Comprobación:**

Usando Postman:

Metodo:GET

URL:<http://localhost:3001/api/dashboard>

Respuesta recibida:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

El manejo se lleva a cabo en esta línea de la implementación:



**RNF2: El sistema debe adaptar su contenido para mostrarse con claridad en dispositivos móviles.**

**Descripción:**

El sistema debe garantizar que la interfaz de usuario se adapte correctamente a diferentes tamaños de pantalla, especialmente a dispositivos móviles con anchos menores a 600px. Esto implica que los elementos visuales, textos, botones y formularios sean legibles, accesibles y que no provoquen scroll horizontal ni desbordes. Además, la navegación debe ser intuitiva y cómoda para usuarios que utilicen smartphones.

Este requerimiento corresponde a la característica de **usabilidad** de la norma ISO 25010, cubriendo las subcaracteristicas de **operatividad, estética de interfaz de usuario** y **accesibilidad**.

**Implementación:**

Para cumplir con este requisito se implementaron media queries en los archivos CSS principales, ajustes en tamaños de fuente, márgenes y paddings, y la inclusión de la meta etiqueta viewport en el HTML base.

**Escenario de calidad :**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

A continuación, se adjunta como se presentaría la web en dispositivos móviles:

Gráfico

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**RNF3: Se debe utilizar un servicio externo que es brindado por SendGrid para el envío de correos a los usuarios.**

Descripción: se necesita este servicio externo para realizar el envío de correos a los usuarios para la recuperación de contraseñas o validación de emails.

## Análisis de seguridad

1. **Fuerza bruta en el login**

Si al momento de ingresar contraseñas no existe un límite de intentos fallidos, ni se aplican bloqueos o retrasos, un atacante puede adivinar una contraseña aplicando fuerza bruta probando con muchas combinaciones automáticamente en el endpoint de login /api/auth/login, pudiendo descubrir las cuentas de usuarios con contraseñas débiles.

**Contramedidas efectivas:**

1. **Rate limiting:** lo que se hace es limitar la cantidad de login por IP o por usuario en un tiempo determinado. Para ello se puede implementar utilizando la tecnología “express-rate-limit” la cual limita la cantidad de peticiones de un endpoint por IP.

**express-rate-limit**: librería de Node.js para Express que sirve para limitar la cantidad de solicitudes que un cliente puede hacer a tu API en un período de tiempo determinado.

Ejemplo de implementación de express-rate-limit:

Texto

El contenido generado por IA puede ser incorrecto.

En este ejemplo luego de 5 intentos fallidos, se aplica un bloqueo y se deben esperar 15 minutos para poder volver a ingresar una contraseña.

1. **Bloqueo temporal del usuario:** después de una N cantidad de intentos fallidos, se bloquea la cuenta por un tiempo.

**Implementación:**

Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Se guarda un campo failed\_attemps y LockUntil en la base de datos y a través de la lógica del backend se incrementa failed\_attemps en cada login fallido, y si se supera el límite se establece lock\_until = NOW() + INTERVAL '30 minutes', y se rechaza el login hasta que expire el bloqueo.

1. **Falta de expiración y revocación de sesiones activas**

Actualmente, cuando un usuario cierra sesión, el token JWT se borra del frontend, pero **sigue siendo válido en el backend hasta que expire** (2 horas). Esto significa que, si un token es robado, un atacante podría usarlo para acceder a la cuenta durante todo el tiempo de vida del token. No existe un mecanismo para revocar o invalidar tokens antes de su expiración, lo que representa un riesgo significativo de acceso no autorizado.

**Solución:**

**Implementar una lista de revocación de token:** Cada vez que un usuario cierra sesión, el frontend manda el token al backend y guarda el jti del token junto con la fecha de expiración en la tabla Tokens\_revocados. Si alguien intenta usar ese token el backend lo detecta en la tabla y no permite el acceso. Cada cierto tiempo se borran de la tabla los tokens que ya expiraron.

**Implementación:**

En la **base de datos** se necesitaría crear la tabla Tokens\_revocados con las columnas de token\_id, user\_id, y fecha de expiración.

En el **backend**:

* Generación de tokens: incluir el campo jti único por token al generarlo.
* Limpiar en la tabla los tokens expirados.
* Middleware de autenticación:
* Verificar la firma y expiración del jwt y consultar la tabla de tokens revocados antes de permitir el acceso.

**Endpoints:**

* Logout: guarda el token en la base de datos al cerrar sesión.

1. **Contraseñas sin requisitos de complejidad**

**Descripción del riesgo**

Actualmente en el sistema en la función register, no se valida el nivel de seguridad de la contraseña creada, ni se exigen requisitos mínimos para la creación de estas como el uso de mayúsculas, símbolos, números o una cantidad de caracteres mínima. Esto le permite a los usuarios utilizar contraseñas comunes y poco seguras como 1234, qwery, contraseña entre otras. Esto debilita la autenticación permitiendo a los atacantes por medio de fuerza bruta o diccionarios, adivinar las contraseñas de los usuarios tener acceso a la información personal y financiera de los usuarios.

**Solución propuesta:**

Implementar políticas de complejidad de contraseñas que obliguen a los usuarios a definir credenciales seguras:

* Longitud mínima de 8 caracteres.
* Inclusión de mayúsculas, minúsculas números y símbolos especiales.
* Rechazo de contraseñas muy comunes.

**Implementación:**

* En el backend utilizar una librería como *validator* para validar el formato de la contraseña al registrar un usuario y al querer cambiar una contraseña.
* Rechazar cualquier contraseña que no cumpla con las políticas definidas.
* Crear una tabla en la base de datos con contraseñas comunes y siempre antes de guardar una nueva contraseña consultar esta base de datos y rechazarla si esta en la lista.

1. **No hay verificación de email**

**Descripción del riesgo:**

Actualmente en el sistema permite que cualquier persona se registre con cualquier dirección de correo electrónico, lo cual conlleva riesgos como suplantación de identidad, en el caso de el usuario olvide la contraseña no pueda recuperarla y la creación de cuentas spam que ocupen espacio en la base de datos.

**Solución propuesta:**

Implementar un proceso de verificación de email al registrarse un nuevo usuario:

* Se envía un correo electrónico de confirmación con un enlace único y temporal.
* La cuenta queda en estado pendiente hasta que el usuario haga click en el enlace
* Solo después de confirmar el email la cuenta se activa y puede iniciar sesión.

**Implementación técnica:**

En la base de datos se agrega un campo email\_verificado en la tabla usuarios.

**Registro**: Al registrarse, el backend genera un **token de verificación único** utilizando la librería nativa de Node.js crypto. Este token se convierte a cadena hexadecimal y se asocia al usuario en la base de datos, junto con una fecha de expiración. Luego, se envía un correo electrónico al usuario utilizando el servicio **SendGrid**, incluyendo un enlace con el token de verificación.

Texto

El contenido generado por IA puede ser incorrecto.

**Endpoint de verificación:** cuando el usuario hace clic en el enlace, el backend valida el token y marca el campo *email\_verificado* como *true*.

En autenticación no permitir el login si el email no ha sido verificado.

1. **Falta de validación y sanitización de datos de entrada en la función register**

**Descripción del riesgo:**

En la función register, los campos nombre, email y contraseña se reciben directamente del cuerpo de la petición (req.body) y se almacenan en la base de datos sin ninguna validación ni limpieza. Esto permite que usuarios malintencionados introduzcan datos peligrosos o inconsistentes.

Esto puede tener como consecuencia un ataque de tipo XSS almacenado, en el momento que un atacante ingrese por ejemplo en el campo nombre un script malicioso y puede ejecutar el código.

**Aclaración:**  
Aunque el frontend (React) suele escapar los datos al renderizarlos, guardar datos maliciosos en la base de datos es una mala práctica y puede ser peligroso si en el futuro se cambia el framework, se exportan datos, o se usan en otros contextos.

**Solución:**

**Validación de entrada**: en el backend se debe verificar que los campos cumplan con el formato esperado donde en el nombre solo se permitan letras y el email tenga un formato valido.

**Sanitización de datos:** Antes de guardar en la base de datos se deben eliminar caracteres peligrosos (<,>,&,”), evitando que se ejecute código malicioso si los datos se muestran en el frontend.

**Uso de librerías de sanitización y validación:** Librerias como validator.js para validadr emails (validator.isEmail) y escapar de caracteres peligrosos (validator.escape).

# Plan de implementación y puesta en marcha

**Estado actual del proyecto al día de la fecha 23 de septiembre de 2025.**

Ya se encuentra implementado y probado en entorno local:

* **Gestión de movimientos**
  + RF1: Ingresar movimiento.
  + RF2: Detallar monto, tipo, categoría, fecha y descripción.
  + RF5: Editar movimiento.
  + RF6: Eliminar movimiento.
  + RF7: Listar movimientos en orden cronológico.
  + RF13: Actualización automática de la lista de movimientos.
* **Gestión de categorías**
  + RF3: Crear categoría.
* **Dashboard básico**
  + RF9: Mostrar ingresos, gastos y balance mensual.
* **Gestión de usuarios (autenticación)**
  + RF16: Crear cuenta.
  + RF17: Iniciar sesión.
  + RF18: Validar email único.
* **Presupuestos**
  + RF10: Crear presupuestos y notificar excedentes.

**Recursos necesarios para llevar a cabo la implementación del proyecto:**

**Técnicos:**

* **Backend:** 
  + Node.js: entorno de ejecución de JavaScript en el servidor.
  + Express.js: Framework para crear API REST y gestionar rutas.
  + PostgreSQL : base de datos.
  + Bcryp: encriptación de contraseñas.
  + JWT: autenticación y autorización mediante tokens.
  + Dotenv: manejo seguro de variables de entorno.
* **Frontend:** 
  + React: biblioteca para la construcción de la UI.
  + Vite
  + React Router: para la navegación entre paginas y protección de rutas privadas.
  + Fecth API: comunicación con la API backend desde el frontend.
* **Infraestructura:** GitHub (repositorio), Render (backend/DB), Vercel (frontend).
* **Colaboración:**, StarUML (diagramas), Postman (API).
* **Servicios externos:** sendgrid (utilizado para el envio de emails)

**Iteraciones del proyecto:**

## 5 iteración (Realizada) :

Fecha de inicio: 23 de agosto del 2025

Fecha final 23 de septiembre de 2025

Duración estimada: 4 semanas

Objetivo:

Tener una primera versión **completamente funcional** de la aplicación, con la **base de datos diseñada e implementada**, backend operativo y frontend conectado, además de los **requisitos funcionales centrales** ya listos y probados.

**Requisitos funcionales cubiertos**

* **RF1:** Ingresar un nuevo movimiento.
* **RF2:** Detallar monto, tipo (ingreso/gasto), categoría, fecha y descripción.
* **RF3:** Crear nueva categoría.
* **RF5:** Editar un movimiento.
* **RF6:** Eliminar un movimiento.
* **RF7:** Listar movimientos en orden cronológico.
* **RF9:** Mostrar total de ingresos, gastos y balance mensual.
* **RF10:** Crear presupuestos mensuales y notificar sobre su estado.
* **RF13:** Actualización automática de lista de movimientos.
* **RF16:** Registro de usuario (email + contraseña).
* **RF17:** Inicio de sesión de usuario.
* **RF18:** Verificación de emails únicos.

**Requisitos no funcionales cubiertos:**

* RNF1: EL Sistema debe estar protegido a accesos no autorizados a rutas privadas.

**Tareas realizadas:**

**Base de datos:**

Creación de tablas de usuarios, categorías, movimientos y presupuestos.

**Backend** (Express + PostgreSQL):

Se implementarios los endpoints REST principales.

Middleware de autenticación JWT activo en rutas protegidas.

**Testing:**

Realizar las pruebas unitarias para testear los requerimientos funcionales implementados.

**Documentación:**

* Descripción de casos de uso.
* Diagramas BPMN de procesos claves.
* Diagramas de secuencia de casos de uso.
* Diagrama de casos de usos.
* Descripción de endpoints y procesos.

**Entregables:**

* Código fuente backend y frontend.
* Base de datos implementada y conectada.
* Conjunto de pruebas unitarias.
* Aplicación funcional en entorno local.
* Documentación completa.

## 6 Iteración:

Duración: 4 semanas

Fecha de inicio 24 de septiembre.

Fecha finalización: 24 de octubre

**Objetivo:**

Agregar al sistema funcionalidades avanzadas que mejoren la experiencia del usuario y la seguridad de la aplicación, incluyendo:

* Visualización de gráficos.
* Filtrado de movimientos.
* Notificaciones de presupuesto.
* Seguridad mejorada (validación de contraseñas, verificación de email, protección de rutas).
* Documentación completa de nuevas funcionalidades.

**Requisitos funcionales cubiertos**

* **RF8:** Filtrar la lista de movimientos por mes, tipo y categoría.
* **RF11a:** Gráfico de barras que muestre total de ingresos y gastos.
* **RF11b:** Gráfico de torta con distribución de gastos por categoría.
* **RF12:** Gráfico de línea que represente el balance mensual a lo largo del tiempo.
* **RF14:** Guardado automático de todos los cambios y registros en la base de datos.
* **RF15:** Restablecer gastos e ingresos al iniciar un nuevo mes.
* **RF19:** Permitir al usuario cambiar la contraseña si se la olvida.

**Requisitos no Funcionales cubiertos:**

* **RNF2:** El sistema debe adaptar su contenido para mostrarse con claridad en dispositivos móviles.
* **RNF3:** Integración con SendGrid para envío de correos (verificación de email y recuperación de contraseña).

**Tareas a realizar:**

**Frontend:**

* Implementar gráficos con Recharts.
* Filtros interactivos para movimientos.
* Notificaciones visuales al superar presupuesto.
* Formularios para cambio de contraseña y verificación de email.

**Backend:**

* Validación y sanitización de entradas usando librerías como validator.js.
* Lista de revocación de tokens y control de sesiones.

**Testing:**

* Unitarios para endpoints de estadísticas y seguridad.
* Integración frontend-backend para gráficos y filtros.
* Pruebas de envío de correos y validación de tokens.
* Realizar pruebas de usabilidad en móvil y escritorio.
* Pruebas de seguridad (ataques de fuerza bruta,revocación de tokens, sanitación de tokens)

**Documentación:**

* Realización de diagrama entidad relación de la base de datos.

**Entregables:**

* Código fuente actualizado backend y frontend.
* Base de datos con nuevas tablas o campos para recuperación de contraseña y revocación de tokens.
* Tests unitarios e integración completos.
* Aplicación funcional con gráficos, filtros y seguridad mejorada.

7 iteración (Refinamiento y despliegue)

Duración estimada=4 semanas

Fecha inicio: 25 de octubre

Fecha final: 25 de noviembre

**Objetivo:** refinar la interfaz de usuario y realizar el despliegue de la aplicación en producción.

**Requisitos funcionales cubiertos**

No se agregan nuevos RF, se mejora lo existente.

Mejoras en RF ya implementados:

* Refinamiento del dashboard y gráficos.
* Ajuste de notificaciones de presupuesto.

**Tareas a realizar :**

* Realizar el refinamiento del frontend.
* Deploy y configuración:
* Backend en **Render** con variables de entorno seguras.
* Frontend en **Vercel** configurado con dominio.
* Configuración de CORS y HTTPS.
* Migración de la base de datos PostgreSQL local a la nube (Render)

**Documentación final:**

Manual técnico: es una recopilación organizada de toda la documentación generada durante el proyecto, el cual incluye:

* Arquitectura del sistema: diagrama de arquitectura y descripción de los componentes.
* Base de datos: Modelo entidad-relación, configuración de la BD en local y Render.
* Backend: estructura de carpetas del proyecto, dependencias principales, descripción de endpoints, configuración de variables de entorno.
* Frontend: Estructura de carpetas del proyecto, principales componentes, contextos usados y manejo de rutas protegidas.
* Testing: estrategia de pruebas (unitarias).
* Despliegue: Desccripcion de pasos para correr en local, y despliegue en Versel y Render.
* Mantenimiento y futuras mejoras: Cómo agregar nuevos endpoints, cómo extender el modelo de datos, requisitos no funcionales a mejorar (seguridad, performance, escalabilidad).

**Testing:**

* Pruebas end-to-end en los principales flujos: login, registrar movimiento, editar movimiento, filtrar, dashboard.
* Pruebas unitarias y de integración completadas en todas las capas

**Entregables:**

* Aplicación desplegada en Render (backend) y Vercel (frontend).
* Base de datos en la nube.
* Documentación final (manual técnico + usuario + plan de mantenimiento).
* Resultados del testing.