# 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:

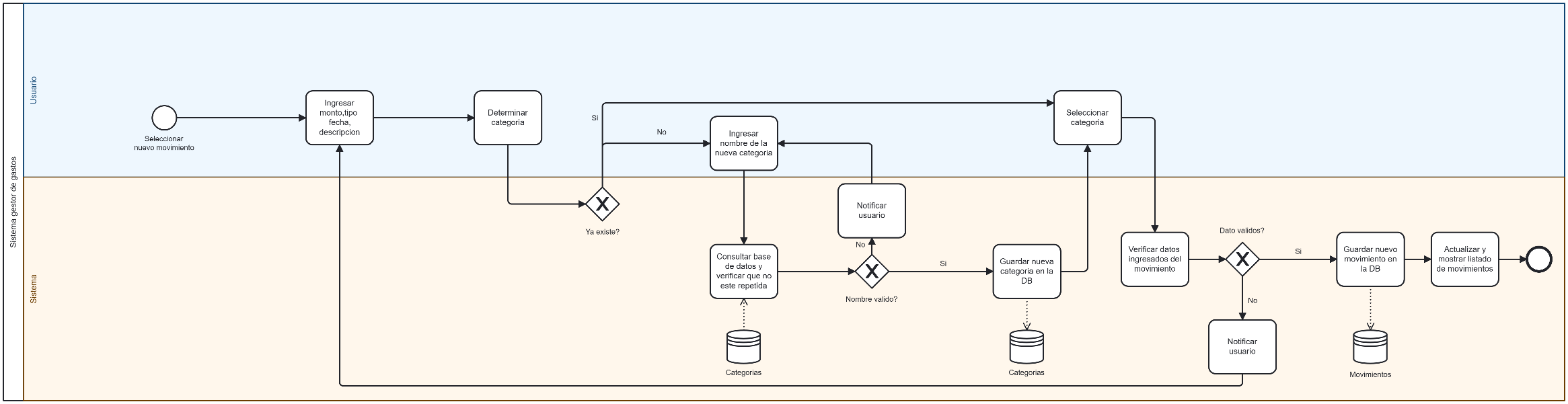
|  |  |
| --- | --- |
| Tecnología | ¿Para que se usa? |
| Visual Studio Code | Entorno de desarrollo principal, para escribir y organizar el código fuente del proyecto. |
| JavaScript | Lenguaje de programación principal que se va a usar para el desarrollo de la lógica del sistema. |
| JavaScrip con 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:**

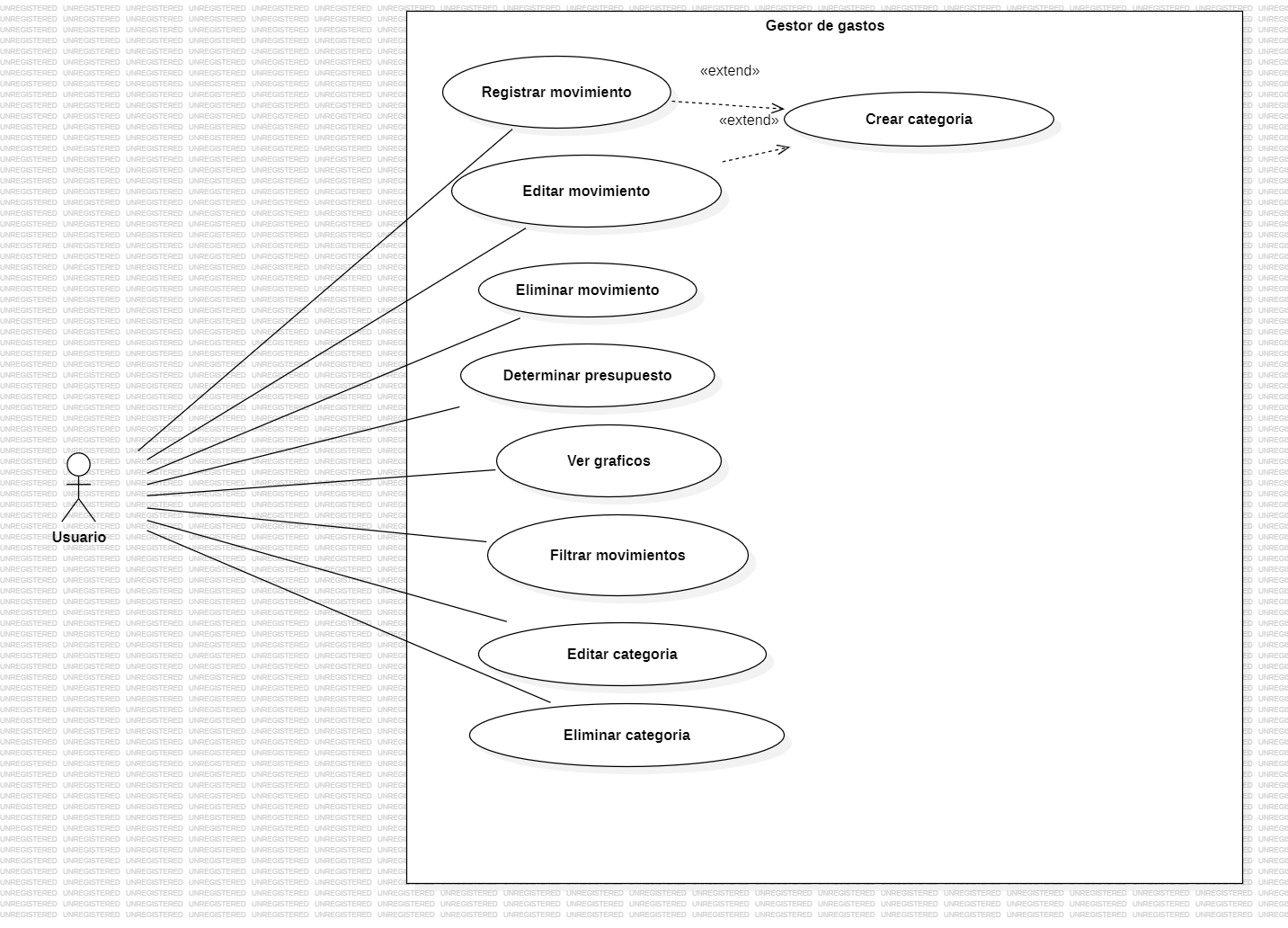
**Diagrama

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.

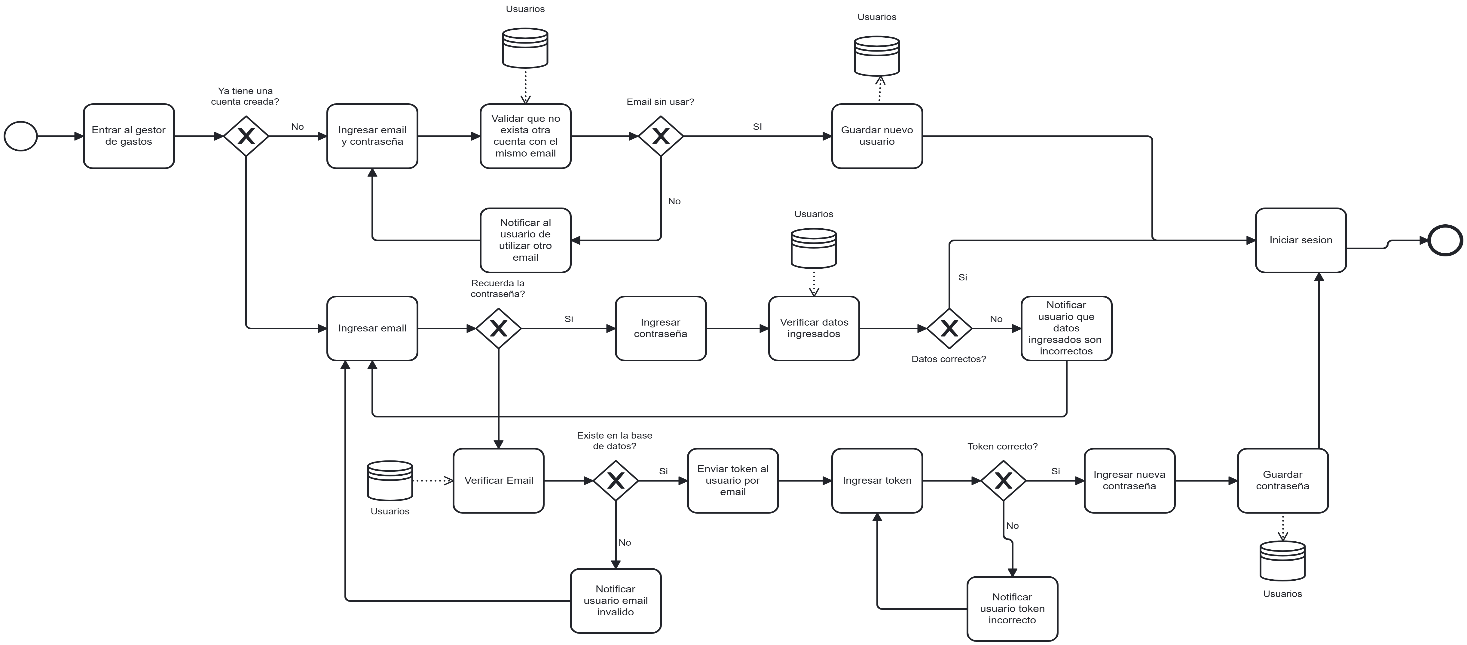
## Segunda Iteracion

### Requerimientos:

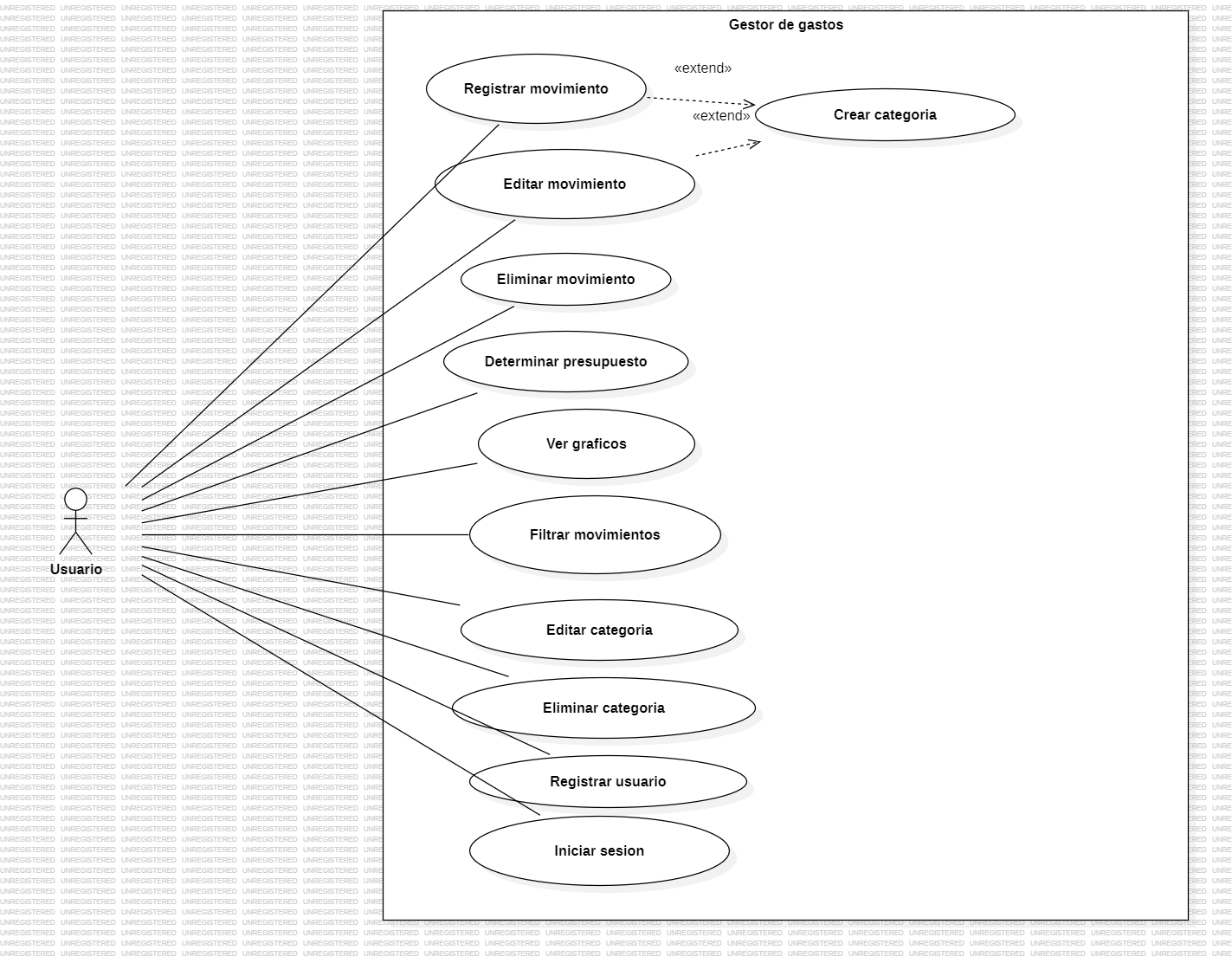
Nuevos requerimientos funcionales:

* 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:

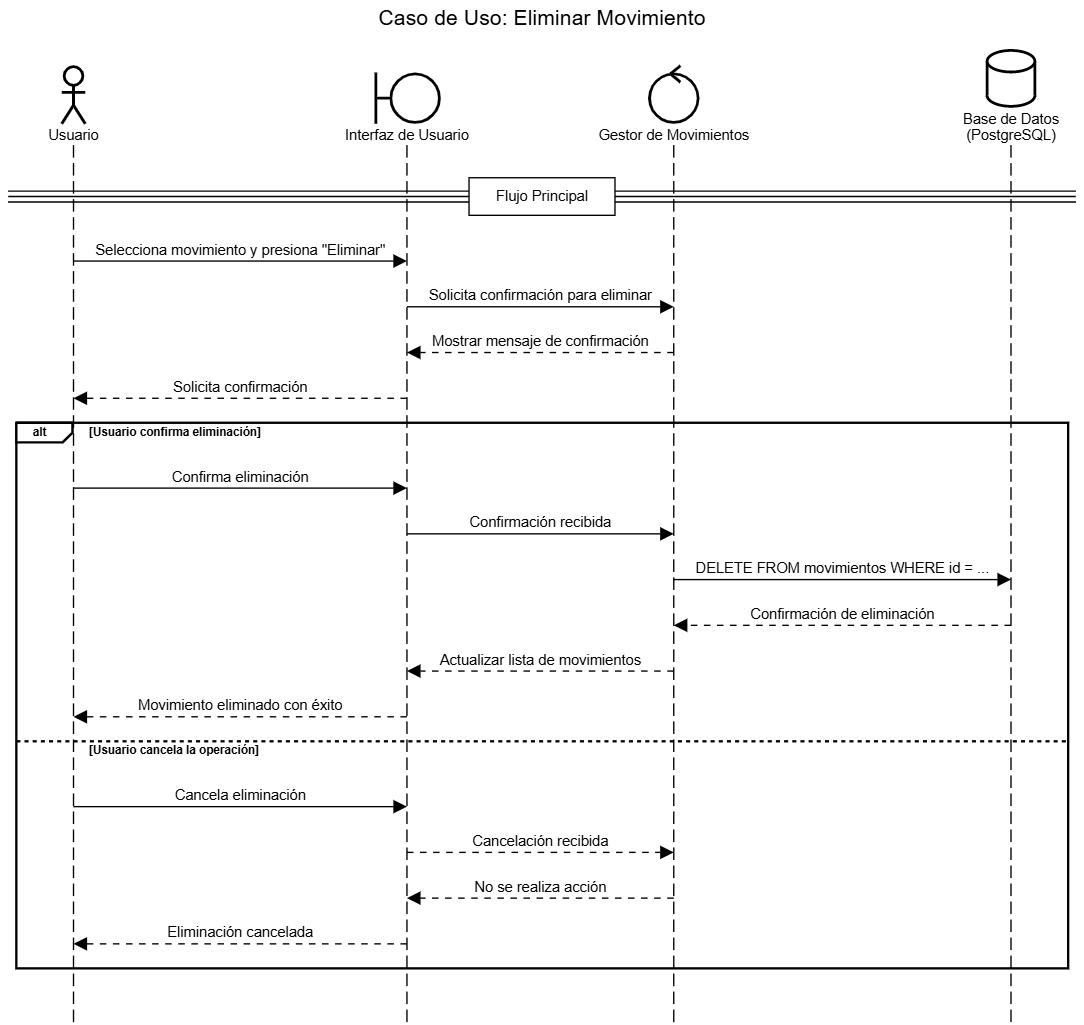
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de Uso**: Ver gráficos | | | | |
| **ID: 1** | | | **Fecha:** | |
| **Descripción:** Se le muestran los gráficos al usuario en la sección de gráficos | | | | |
| **Actores Principales**: Usuario | | **Actores Secundarios**: | | |
| **Observaciones:** | | | | |
| **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:

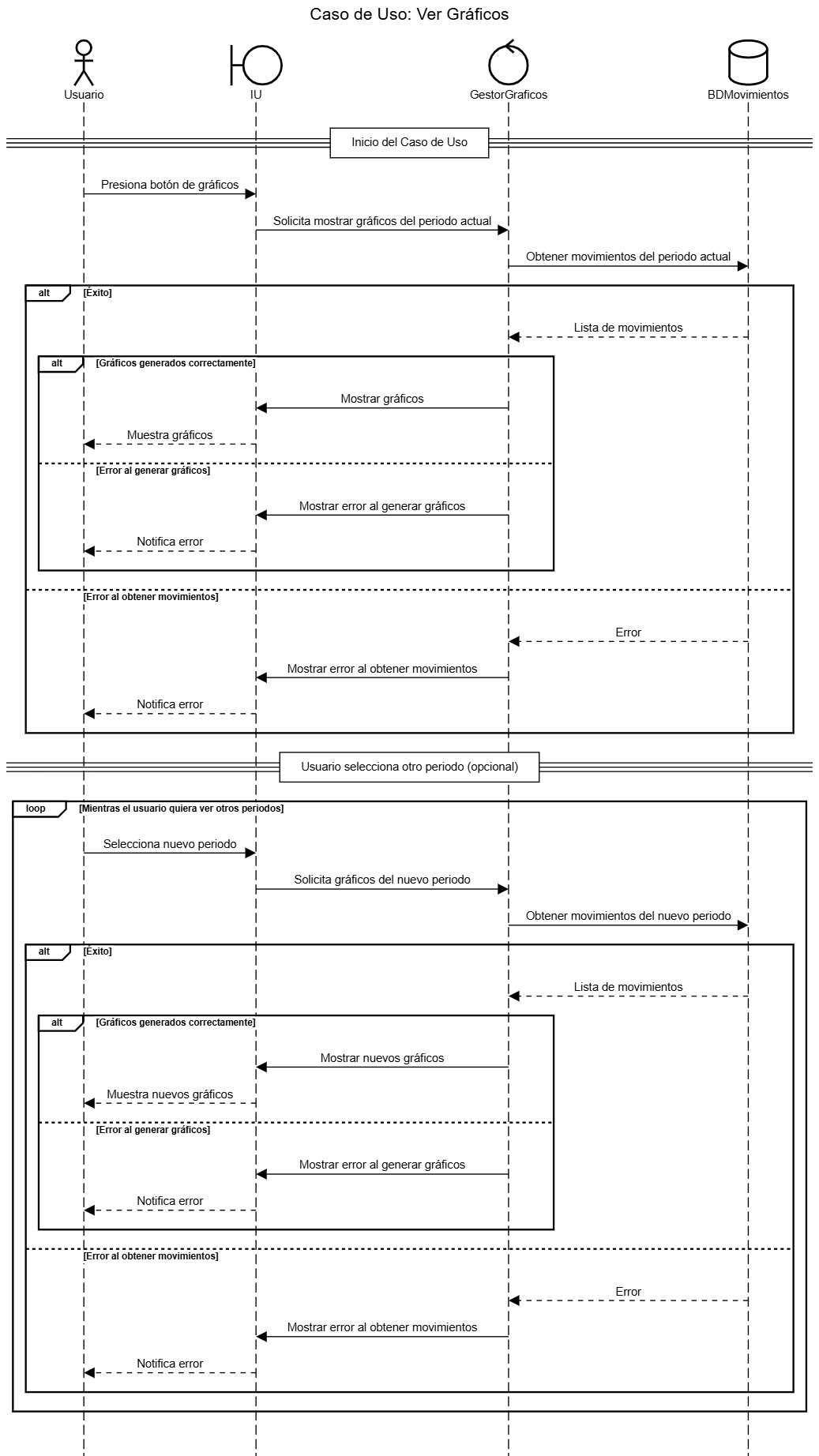
Diagrama

El contenido generado por IA puede ser incorrecto.



Tabla, Calendario

El contenido generado por IA puede ser incorrecto.



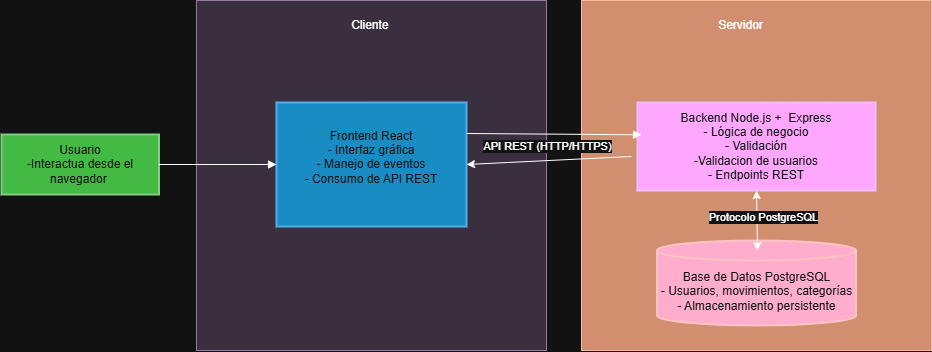
Diagrama

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.

### Requerimientos no funcionales:

**RNF1:** Autenticación basada en JWT, para garantizar que solo usuarios autenticados accedan a recursos privados.

**RNF2:** Control de acceso por usuario, para evitar que un usuario pueda modificar lo de otro.

**RNF3:** Validación de datos: El sistema debe validar todos los datos recibidos desde el cliente en el backend, asegurando que cumplen con las reglas de formato, tipo y contenido esperadas. Esto incluye validaciones de campos obligatorios, tipos de dato correctos, valores dentro de rangos válidos, y control de longitud de texto, con el fin de garantizar la integridad de los datos y prevenir ataques como inyecciones o fallos lógicos.

### 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.

Y también se implementaron los requerimientos no funcionales:

**RNF1:** Autenticación basada en JWT, para garantizar que solo usuarios autenticados accedan a recursos privados.

**RNF2:** Control de acceso por usuario, para evitar que un usuario pueda modificar lo de otro.

**RNF3:** Validación de datos: El sistema debe validar todos los datos recibidos desde el cliente en el backend, asegurando que cumplen con las reglas de formato, tipo y contenido esperadas. Esto incluye validaciones de campos obligatorios, tipos de dato correctos, valores dentro de rangos válidos, y control de longitud de texto, con el fin de garantizar la integridad de los datos y prevenir ataques como inyecciones o fallos lógicos.

El hosting de la página se realizó en Versel para el frontend y en Render para el backend y la base de datos.