Trabajo Practico Diseño de Sistemas de la información

Alumno: Agustín Perez

**Introducción**

**Problema**

En la actualidad, muchas personas enfrentan dificultades para llevar un control eficiente de sus gastos. El uso de métodos tradicionales como libretas o planillas de cálculo puede generar desorganización, errores y falta de seguimiento. Esto impide por ejemplo poder saber bien en qué categorías se podría hacer una reducción de los gastos.

**Objetivos**

* Desarrollar una aplicación web que permita registrar, clasificar y visualizar gastos de forma simple e intuitiva.
* Facilitar al usuario el acceso a sus registros desde cualquier dispositivo con conexión a internet.
* Implementar mecanismos de validación y retroalimentación para asegurar la integridad de los datos ingresados.
* Aplicar buenas prácticas de modelado y desarrollo con herramientas modernas.

**Forma en la que se va a llevar a cabo el proyecto**

He decidido desarrollar este proyecto utilizando el modelo de desarrollo en cascada, ya que proporciona una estructura secuencial y bien definida, que se adapta adecuadamente al tipo de proyecto que estamos realizando.

Este enfoque permite avanzar de manera ordenada por cada fase del desarrollo: análisis de requisitos, diseño, implementación, pruebas y, finalmente, despliegue. Cada fase debe completarse y validarse antes de pasar a la siguiente, lo que garantiza un control estricto del avance y una mayor claridad en la planificación del proyecto.

**Desarrollo**

**Primera iteración:**

Requisitos:

Funcionales:

* El sistema debe permitir al usuario iniciar sesión con credenciales válidas.
* El usuario debe poder agregar, editar, eliminar y visualizar gastos.
* Debe ofrecer un historial de gastos filtrado por año, mes y categorías.
* Debe permitir cerrar sesión de forma segura.

Procesos identificados en el gestor de gastos:

* Iniciar sesión
* Registrar Usuario
* Agregar gastos
* Editar gastos
* Eliminar gastos
* Cerrar sesión
* Mostrar lista mensual de gastos filtrados
* Generar lista mensual de gastos filtrados
* Validar usuario
* Guardar gasto en base de datos
* Editar gasto en base de datos
* Eliminar gasto en base de datos

Esquematización de los procesos “Iniciar Sesión, Registrar Usuario y Agregar Gasto”:

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama

El contenido generado por IA puede ser incorrecto.

Diagrama de casos de uso:

Diagrama

El contenido generado por IA puede ser incorrecto.Diagrama

El contenido generado por IA puede ser incorrecto.

Descripción de casos de uso Iniciar Sesión, Registrar usuario y Agregar Gasto:

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Diagramas de secuencia de los casos de uso Iniciar Sesión, Registrar usuario y Agregar Gasto:

Los siguientes diagramas de secuencia nos permiten visualizar como interactúan los componentes del sistema a lo largo del caso de uso.

Tabla

El contenido generado por IA puede ser incorrecto.

Diagrama

El contenido generado por IA puede ser incorrecto.

Para el prototipo del sistema se utilizó una herramienta de IA. Durante el proceso de creación se tuvo en cuenta realizar un sistema sencillo y muy intuitivo para usuarios inexpertos donde la disposición de los botones, información y campos de relleno de información sea clara y no haya confusiones. La paleta de colores se eligió como para que no sea molesta a la vista.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Teams

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

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

Se realizaron 5 pruebas para descubrir posibles fallas en la implementación, en siguientes iteraciones se realizarán más:

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Segunda iteración:**

Arquitectura:

Se utiliza una arquitectura cliente – servidor donde:

* Cliente: Interfaz desarrollada en React que corre en el navegador.
* Servidor: API RESTful que procesa solicitudes y accede a la base de datos.
* Base de datos: Almacena usuarios, gastos y categorías.

Diagrama

El contenido generado por IA puede ser incorrecto.

Endpoints de la Api que permite comunicar al cliente con el servidor y este con la base de datos.

| **Acción** | **Método HTTP** | **Endpoint** | **Descripción** |
| --- | --- | --- | --- |
| Iniciar sesión | POST | /api/login | Autenticar usuario |
| Registrar usuario | POST | /api/register | Crear nuevo usuario |
| Agregar gasto | POST | /api/gastos | Registrar gasto nuevo |
| Editar gasto | PUT | /api/gastos/:id | Modificar un gasto existente |
| Eliminar gasto | DELETE | /api/gastos/:id | Eliminar un gasto |
| Ver lista de gastos | GET | /api/gastos? | Obtener gastos filtrados |

Clasificación de errores según su prioridad:

* Crítico: Impide el uso básico del sistema o pone en riesgo la seguridad o datos.
* Alto: Afecta funcionalidades importantes, pero tiene alguna solución temporal.
* Medio: Problema molesto, pero no impide el uso del sistema.
* Bajo: Detalles de estética.

Clasificación de errores encontrados en las pruebas hechas en la Primera Iteración:

Prueba 2: Agregar gasto (no muestra mensaje). Prioridad Medio.

Prueba 3: Agregar gasto con monto negativo. Prioridad Medio.

Prueba 4: Agregar Gasto con descripción infinita. Prioridad Alto.

**Tercera Iteración:**

Se corrigieron requisitos funcionales y no funcionales, se agregaron procesos y se rehicieron diagramas en base a esos cambios.

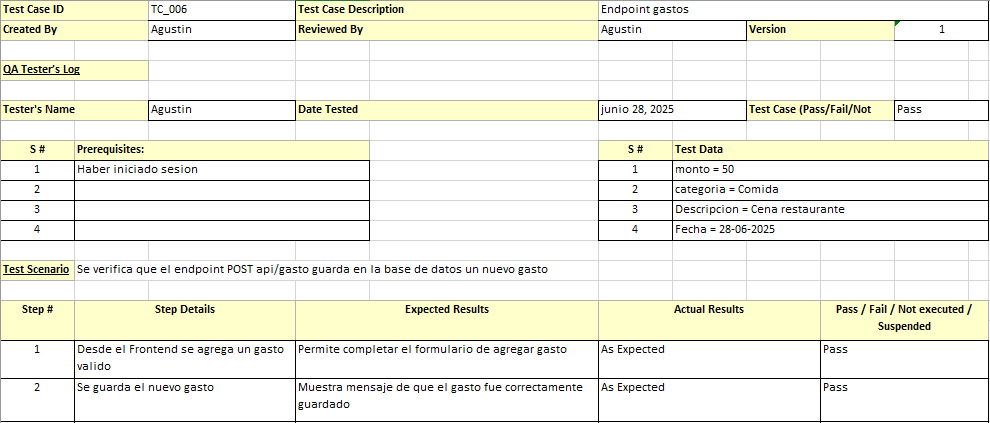
Se implementó una página web la cual consta de un Frontend implementado en React como biblioteca de JavaScript, un Backend utilizando Node.js con Express creando una API REST que permite manejar las operaciones del sistema conectado a una Base de Datos hecha con PostgreSQL, todo esto está en la plataforma de Render así cualquier persona puede probarlo y realizar test. No hay que tomar en cuenta los tiempos de respuesta del servidor como parte del rendimiento real de la aplicación ya que Render utiliza los suyos generando una latencia grande y limitaciones por el plan gratuito.

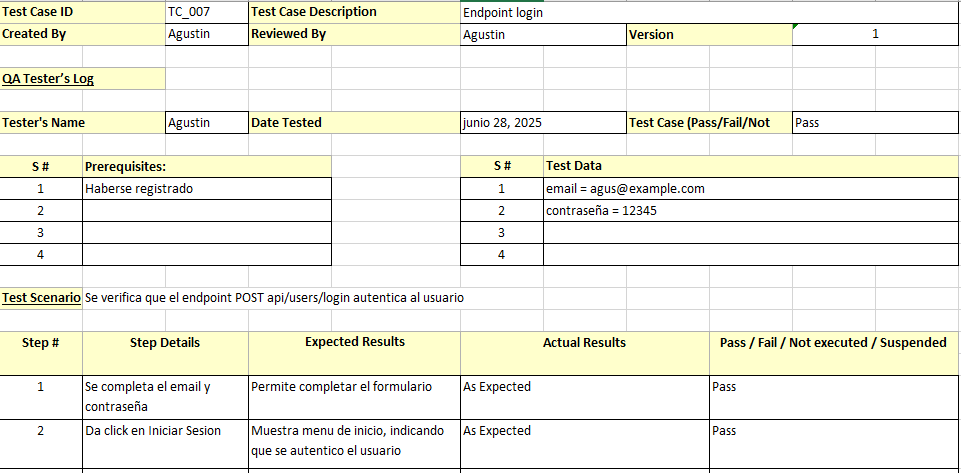
La implementación realizada no es definitiva ya que quedan cambios por hacer como las opciones visualizar gastos, editar y eliminar ya que estas no están esquematizadas como van a hacer los procesos. Estos se implementaron solo para probar la página web.

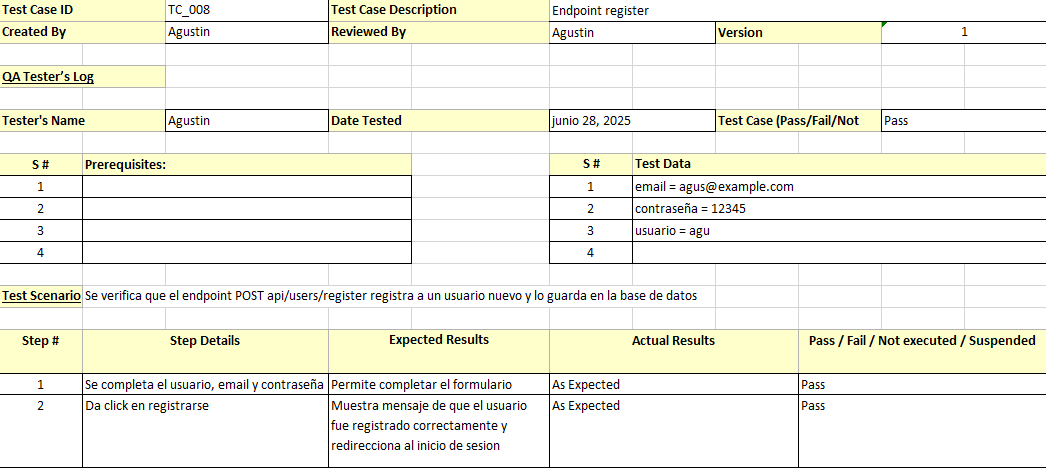
URL de la página web: <https://gestor-de-gastos-3alz.onrender.com>

Se realizarán nuevas PRUEBAS ya que los anteriores estaban hechos sobre una implementación antigua.

Las PRUEBAS realizados se hicieron buscando errores en los endpoints sobre las acciones de agregar gasto, iniciar sesión y registrar un nuevo usuario.







**Cuarta Iteración:**

Se borraron los Requerimientos no funcionales planteados al principio, para cambiarlos por unos nuevos mejor pensados.

Requerimientos no funcionales:

• Compatibilidad: La aplicación deberá garantizar un funcionamiento correcto en los principales navegadores.

* Navegadores web: Se debe asegurar compatibilidad funcional y visual completa en las dos últimas versiones estables de los siguientes navegadores:
  + Google Chrome
  + Microsoft Edge
* Condiciones de aceptación:
  + Las funcionalidades críticas (como login, navegación principal y formularios) deben funcionar correctamente en todos los navegadores.
  + Las pruebas deben cubrir tanto validación visual (sin errores de diseño) como validación funcional (que cada botón, enlace o componente interactivo funcione como se espera).
  + Se considera error grave si alguna funcionalidad queda inutilizable en cualquiera de los entornos definidos como compatibles.

• Seguridad: Todo componente del sistema que maneje información personal, sensible o crítica deberá cumplir con los siguientes criterios:

* Toda la comunicación entre cliente y servidor deberá realizarse mediante el protocolo HTTPS.
* Los formularios que gestionen credenciales u otros datos sensibles deben validar los datos tanto en frontend como en backend.
* En cada inicio de sesión:
  + Si se detectan 5 intentos fallidos consecutivos, la cuenta será bloqueada por 5 minutos.
  + Todos los accesos a recursos protegidos deben requerir autenticación previa mediante token válido o sesión activa, sin excepción.

**Quinta Iteración:**

Posibles problemas de seguridad:

1. Inyección SQL: Al tener una base de datos SQL, si las consultas del cliente hacia el servidor no están restringidas, alguien podría utilizar esto a su favor para manipular el inicio de sesión por ejemplo haciéndole creer al servidor que está entrando el admin.

Contramedidas técnicas para implementar:

* Usar consultas parametrizadas con librerías Node.js
* Evitar concatenar strings con variables

Herramientas para implementar estos cambios: Sequelize como Query

Cambios en código: Consulta hecha con Sequelize



1. Intentos ilimitados en el login: Este es un problema de seguridad en el cual un usuario podría intentar entrar a la página web, con fuerza bruta es decir que ingresa muchas combinaciones de mail y contraseña por segundo hasta que adivina alguna. Estos ataques se hacen de forma automática.

Contramedidas técnicas para implementar:

* Limitar la cantidad de pedidos que el usuario puede hacer en un periodo de tiempo

Herramientas: Biblioteca express-rate-limit, esto es un Middleware, es decir una función que se ejecuta antes o después de una consulta.

Cambios en código:

Texto

El contenido generado por IA puede ser incorrecto.

1. Token de inicio de sesión vulnerable a XSS, esto es un ataque donde se logra inyectar código JavaScript en la página web a través de inputs como lo son las descripciones. El token queda expuesto ya que se guarda en el localStorage y este es accesible desde JavaScript.

Contramedidas técnicas para implementar:

* Guardar el token en una cookie protegida
* Enviar la cookie por HTTPS
* Darle un tiempo de vencimiento

Herramientas para la implementación: Cookie HttpOnly es como el nombre lo dice una cookie del navegador así el usuario que intente ingresar JavaScript no tiene acceso a esta cookie.

Cambios en código:

Texto

El contenido generado por IA puede ser incorrecto.

1. CORS sin restricciones: Un CORS es un mecanismo de seguridad que permite o restringe solicitudes HTTP realizadas desde un origen distinto a donde este alojado el servidor, es decir el frontend. Esto significa que cualquier dominio puede hacer peticiones a la API, esto es perjudicial ya que los atacantes podrían acceder a datos de usuarios.

Contramedidas técnicas para implementar:

* Restringir el acceso solo al dominio donde esta alojada mi pagina web en este caso render

Herramientas para la implementación: mantenemos el CORS el cual es un middleware de expres pero tenemos que cambiarle la configuración un poco

Texto

El contenido generado por IA puede ser incorrecto.

1. Seguridad en cifrado SSH/SSL: estos son protocolos de cifrados para realizar una comunicación entre cliente y servidor.

Contramedidas técnicas para implementar:

* Cifrar la conexión con SSL y verificar el certificado utilizado

Herramientas para la implementación: Al tener la pagina web en render este ya genera los certificados para el dominio, lo único seria implementar en el código la verificación de estos, de la siguiente forma:

Imagen que contiene Logotipo

El contenido generado por IA puede ser incorrecto.

**Sexta Iteración:**

Se establece un plan a seguir para las siguientes iteraciones, detallando recursos necesarios, herramientas a utilizar y requerimientos para implementar detallando fechas.

Recursos necesarios:

RRHH:

* Un desarrollador frontend que sepa utilizar React
* Un desarrollador backend que sepa utilizar Node.js, Express y PostgreSQL
* Un tester QA
* Un documentador que mantenga actualizado todos los documentos de requisitos, casos de usos, diagramas y entre otros.
* Un jefe de proyecto que coordine al equipo

Herramientas para utilizar:

* GitHub como repositorio
* Render para el hosting de la aplicación
* PostgreSQL para la base de datos
* Jest y Supertest para llevar a cabo las pruebas
* Figma para el diseño de la interfaz de usuario
* OWASP ZAP, proxi de seguridad para detectar todos los problemas vistos en la iteración 5 del proyecto.

**Iteración 7: Finalización de funcionalidades básicas**

Objetivo: Dejar implementadas funcionalidades básicas faltantes.

Duración estimada: 4 semanas

* Requisitos funcionales para implementar:
  + Registro de usuario, inicio y cierre de sesión.
  + Agregar, editar, eliminar y visualizar gastos.
  + Filtrado de gastos por categorías, mes y año.
* Requisitos no funcionales para implementar:
  + Compatibilidad en Chrome y Edge.
  + Validación de inputs en frontend y backend.
* Actividades clave:
  + Completar procesos de Editar y Eliminar gasto.
  + Migrar almacenamiento de token a cookies HttpOnly.
  + Implementar rate limiting en login (middleware express-rate-limit).
  + Configurar CORS restringido al dominio de producción.
  + Tests unitarios y de integración (Jest/Supertest).

**Iteración 8: Optimización de seguridad**

Objetivo: Fortalecer la aplicación contra ataques comunes y optimizar la respuesta.

Duración estimada: 2 semanas

* Requisitos funcionales: No se agregan nuevos.
* Requisitos no funcionales para implementar:
  + Seguridad contra inyección SQL.
  + Protección contra XSS.
  + Logs de auditoría para accesos y modificaciones.
  + Documentación de vulnerabilidades mitigadas.
  + Bloqueo de cuenta tras 5 intentos fallidos.
  + Conexión segura por HTTPS.
* Actividades clave:
  + Refactorizar consultas con Sequelize.
  + Validacion de entradas en backend.
  + Implementar middleware de login.
  + Pruebas de seguridad básicas (OWASP ZAP).

**Iteración 9: Funcionalidades extras**

Objetivo: Extender el alcance funcional del sistema.

Duración estimada: 4 semanas

* Requisitos funcionales para implementar:
  + Reportes gráficos de gastos por categoría/mes.
  + Soporte para múltiples monedas.
  + Categorías personalizadas por usuario.
* Requisitos no funcionales para implementar:
  + Rendimiento óptimo menos de 5s de respuesta en operaciones básicas.
  + Interfaz accesible.
* Actividades clave:
  + Implementar gráficos.
  + Pruebas de accesibilidad y usabilidad.

**Iteración 10: Despliegue**

Objetivo: Preparar la aplicación para su despliegue online.

Duración estimada: 1 semana

* Requisitos funcionales: no se agregan nuevas funcionalidades.
* Requisitos no funcionales para implementar:
  + Despliegue del frontend y backend en render
  + Copias de seguridad automáticas de la base de datos.
* Actividades clave:
  + Documentar procedimiento de despliegue.
  + Programar copias de seguridad diarias y semanales.

**Iteración 11: Mantenimiento**

Objetivo: Mantener la aplicación estable, corregir errores y optimizar su rendimiento.

Duración estimada: 1 semana

* Requisitos funcionales: no se agregan nuevas funcionalidades.
* Requisitos no funcionales para implementar:
  + Corrección de errores detectados en producción.
  + Actualización de librerías y dependencia.
* Actividades clave:
  + Corregir errores reportados por usuarios.
  + Actualizar dependencias de frontend y backend.
  + Realizar pruebas de usabilidad.