**Informe de Avance – Fase 3**

**Implementación y Evaluación**

**Proyecto:** Sistema de Gestión de Gastos  
**Arquitectura:** Microservicios

**Información General del Proyecto**

* **Nombre del Proyecto:** Gestión de Gastos
* **Sector:** Finanzas
* **Fecha del Informe:** Octubre de 2025
* **Fase Actual:** Fase 3 – Implementación y Evaluación
* **Porcentaje de Avance:** 55 %

**Equipo de Trabajo**

| **Rol** | **Integrante** |
| --- | --- |
| Product Owner | Miguel Ángel Beltrán Bohórquez |
| Desarrollador Backend | Nicolás Rivera Aroca |
| Desarrollador Frontend | Jesús Candelo |

**1. Resumen Ejecutivo**

Este informe documenta el progreso alcanzado en la **Fase 3: Implementación y Evaluación** del proyecto “Sistema de Gestión de Gastos”. Se trata de una aplicación web diseñada para digitalizar y optimizar la gestión financiera personal, reemplazando métodos manuales o físicos. La arquitectura se basa en microservicios, empleando Java con Spring Boot para el backend, Angular para el frontend y PostgreSQL como sistema de base de datos.

Hasta la fecha, se ha completado un **55 % del desarrollo**. Dos microservicios principales (**ms\_expense** y **ms\_income**) están en producción, mientras que otros tres (**ms\_categorias**, **ms\_usuario** y **ms\_login**) se encuentran en etapa de desarrollo.

**2. Descripción del Problema**

**2.1 Contexto**

El uso de herramientas manuales —como libreta, hojas de cálculo o registros en papel— para gestionar gastos personales acarrea diversos problemas:

* Ausencia de centralización de la información
* Dificultad para análisis cronológicos
* Falta de categorización eficiente
* Imposibilidad de acceso remoto
* Riesgo de pérdida física o deterioro de registros

**2.2 Solución Propuesta**

Se propone desarrollar una aplicación web que permita:

* Registrar y visualizar gastos e ingresos digitalmente
* Realizar consultas por fecha, monto o categoría
* Mantener la información centralizada y accesible desde cualquier dispositivo
* Servir de base para futuras funciones de análisis financiero

**3. Arquitectura de la Solución**

**3.1 Arquitectura de Alto Nivel**

El sistema adopta una **arquitectura de microservicios**, lo cual ofrece:

* Escalabilidad independiente por servicio
* Despliegue modular y mantenimiento eficiente
* Desarrollo paralelo por distintos miembros del equipo
* Aislamiento de errores en servicios individuales

**3.2 Stack Tecnológico**

|  |  |  |
| --- | --- | --- |
| **Componente** | **Tecnología** | **Versión** |
| Backend | Java + Spring Boot | 3.0.2 |
| Frontend | Angular | 15.2.1 |
| Base de datos | PostgreSQL | 14 |
| Contenedores | Docker | 24.0 |
| CI/CD | Jenkins | 2.387 |
| Control de versiones | Git / GitHub | — |

*Nota:* Ajusta las versiones reales según lo que uses en tu proyecto.

**3.3 Comunicación entre Servicios**

* **Protocolo:** HTTP REST
* **Formato de datos:** JSON
* **Autenticación:** En desarrollo (ms\_login)

**4. Estado Actual de Implementación**

**4.1 Microservicios Implementados**

**4.1.1 ms\_expense (Microservicio de Gastos) — En producción**

**Estado:** Desplegado y funcional en entorno de pruebas/producción local.

**Funcionalidades ya operativas:**

* Registro de nuevos gastos
* Visualización de gastos registrados
* Consulta por fecha, monto y categoría
* Validación de datos de entrada
* Operaciones CRUD (crear, leer, actualizar, eliminar)

**Endpoints principales (ejemplo):**

POST /api/gastos

GET /api/gastos

GET /api/gastos/{id}

GET /api/gastos/fecha

GET /api/gastos/monto

GET /api/gastos/categoria

PUT /api/gastos/{id}

DELETE /api/gastos/{id}

**Base de datos:**

* Tabla: expenses
* Campos (ejemplo): id, usuario\_id, categoria\_id, monto, fecha, descripcion

**4.1.2 ms\_income (Microservicio de Ingresos) — En producción parcial**

**Estado:** Backend desplegado y funcional, pero aún no integrado al frontend.

**Funcionalidades ya implementadas (backend):**

* Registro de ingresos
* Consultas simples (listar ingresos, obtener por ID)

**Pendiente:** integración con interfaz de usuario (frontend Angular)

**4.2 Microservicios en Desarrollo**

**4.2.1 ms\_categorias (Microservicio de Categorías)**

**Objetivo:** administración centralizada de categorías para gastos e ingresos.

**Funcionalidades planificadas:**

* Crear, listar, actualizar y eliminar categorías
* Asignar iconos o colores a cada categoría

**Estado actual:** ~ 20 % (por ejemplo)

**4.2.2 ms\_usuario (Microservicio de Usuarios)**

**Objetivo:** gestión del perfil y datos de los usuarios.

**Funcionalidades planificadas:**

* Registro de nuevos usuarios
* Actualización de perfil
* Recuperación de contraseña
* Preferencias del usuario

**Estado actual:** ~ 15 % (por ejemplo)

**4.2.3 ms\_login (Microservicio de Autenticación)**

**Objetivo:** manejar la autenticación/autorización segura.

**Funcionalidades planificadas:**

* Login de usuarios
* Emisión y validación de tokens JWT
* Recuperación de contraseña
* Gestión de roles/permisos

**Estado actual:** ~ 10 % (por ejemplo)

**5. Desarrollo del Backend**

**5.1 Estándares de Codificación**

Se adoptaron las siguientes convenciones:

* Clases: PascalCase (por ejemplo: ExpenseController)
* Métodos: camelCase (por ejemplo: createExpense())
* Constantes: UPPER\_SNAKE\_CASE (por ejemplo: MAX\_AMOUNT)
* Paquetes: todo en minúsculas (por ejemplo: com.gestiongastos.expense)

**Estructura de paquetes (ejemplo):**

src/main/java/com/gestiongastos/

├── controller/

├── service/

├── repository/

├── model/

├── dto/

├── exception/

└── config/

**5.2 Modularidad y Reutilización**

* Separación clara de responsabilidades (controller, service, repository)
* Uso de inyección de dependencias con Spring
* DTOs para desacoplar las entidades de la capa de persistencia
* Servicios reutilizables para lógica de negocio común

**5.3 Manejo de Excepciones**

Se ha implementado una capa global de manejo de excepciones:

@RestControllerAdvice

public class GlobalExceptionHandler {

// Métodos para capturar excepciones personalizadas y estándar

// Convertirlas en respuestas HTTP con códigos apropiados

}

**Tipos de excepciones manejadas:**

* Validación de datos de entrada
* Recursos no encontrados (404)
* Errores de base de datos
* Errores de negocio

**5.4 Validación de Datos**

Se utilizan anotaciones de Bean Validation:

* @NotNull, @NotBlank
* @Email, @Pattern
* @Min, @Max
* Validaciones personalizadas si se requiere

**5.5 Documentación y Pruebas**

**Pruebas unitarias:**

* Framework: JUnit + Mockito
* Cobertura actual: ~ 65 % (estimado)
* Módulos testeados: servicios, repositorios, controladores principales

**Documentación del código:**

* JavaDoc en las clases y métodos más relevantes
* Comentarios cuando la lógica es compleja
* README general en el repositorio principal

**6. Desarrollo del Frontend**

**6.1 Diseño de Interfaz y Experiencia de Usuario**

**Tecnología:** Angular, versión 15.x (por ejemplo)

**Principios aplicados:**

* Interfaz limpia e intuitiva
* Diseño adaptable (responsive)
* Feedback visual en acciones del usuario
* Navegación consistente

**Estructura de componentes (ejemplo):**

src/app/

├── components/

│ ├── expense-list/

│ ├── expense-form/

│ ├── expense-filter/

│ └── dashboard/

├── services/

│ ├── expense.service.ts

│ └── api.service.ts

└── models/

└── expense.model.ts

**6.2 Funcionalidades Implementadas**

**Módulo de gastos:**

* Formulario de ingreso de gastos
* Tabla de visualización
* Filtros por fecha, monto, categoría
* Validaciones en cliente

**Pendientes:**

* Integración con ms\_income
* Dashboard con gráficos y estadísticas
* Mecanismo de login/autenticación

**6.3 Modularidad del Frontend**

* Componentes reutilizables
* Servicios HTTP compartidos
* Guards para rutas protegidas (pendiente)
* Interceptors para adjuntar tokens (pendiente)

**6.4 Usabilidad y Accesibilidad**

**Implementado:**

* Mensajes de error/confirmación claros
* Validaciones reactivas
* Indicadores de carga en peticiones asíncronas

**Pendiente:**

* Soporte para lectores de pantalla
* Navegación por teclado
* Modo claro/oscuro

**6.5 Funcionamiento de Elementos Interactivos**

Se han probado:

* Botones acción (guardar, cancelar, eliminar)
* Formularios con validación
* Filtros de búsqueda
* Mensajes de error y confirmación

**7. Despliegue del Sistema**

**7.1 Infraestructura**

**Entorno actual:** local con Docker

**Servicios desplegados:**

* ms\_expense
* ms\_income
* PostgreSQL
* Frontend Angular

**7.2 Pipeline CI/CD con Jenkins**

Flujo configurado:

1. Detección de cambios en GitHub
2. Compilación del código
3. Ejecución de pruebas
4. Construcción de imágenes Docker
5. Despliegue automático (local)

**Automatización:**

* Despliegue tras git push
* Notificaciones en caso de éxito/fallo
* Registro de logs

**7.3 Gestión de Contenedores Docker**

**Servicios definidos en Docker Compose (ejemplo):**

services:

expense-service:

build: ./ms-expense

ports: [...]

income-service:

build: ./ms-income

ports: [...]

db:

image: postgres:14

environment: [...]

frontend:

build: ./frontend

ports: [...]

**Ventajas del uso de Docker:**

* Entornos consistentes
* Escalabilidad sencilla
* Aislamiento entre servicios
* Portabilidad

**7.4 Control de Versiones**

**Repositorio:** GitHub  
**Estrategia de branching sugerida:** Git Flow o desarrollo con feature branches

Branches propuestos:

* main (producción)
* develop (desarrollo integrado)
* feature/\* (funcionalidades)

**Versionado:** Semántico (v0.5.0, etc.)

**7.5 Gestión de Credenciales**

Buenas prácticas:

* Uso de variables de entorno
* .env o archivos de configuración excluidos del repositorio
* Gestión segura de contraseñas y claves

**Credenciales manejadas:**

* Conexión a PostgreSQL
* (En el futuro) claves de servicios externos

**7.6 Validación en Múltiples Entornos**

**Entornos definidos:**

* Desarrollo (funcional)
* Staging (pruebas intermedias)
* Producción real

**8. Pruebas y Evaluación**

**8.1 Estrategia de Pruebas**

**8.1.1 Pruebas funcionales con Postman**

Se probaron los endpoints de ms\_expense y ms\_income:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpoint** | **Método** | **Caso de prueba** | **Estado** | **Resultado** |
| /api/gastos (POST) | Crear gasto válido | ✅ | 201 Created |  |
| /api/gastos (POST) | Falta de campo obligatorio | ✅ | 400 Bad Request |  |
| /api/gastos (GET) | Listar gastos | ✅ | 200 OK |  |
| /api/gastos/{id} (GET) | Obtener gasto existente | ✅ | 200 OK |  |
| /api/gastos/{id} (GET) | Gasto no existente | ✅ | 404 Not Found |  |
| /api/gastos/fecha | Filtrar por rango fechas | ✅ | 200 OK |  |
| /api/gastos/monto | Filtrar por rango montos | ✅ | 200 OK |  |
| /api/gastos/categoria | Filtrar por categoría | ✅ | 200 OK |  |
| /api/gastos/{id} (PUT) | Actualizar gasto | ⏳ | En progreso |  |
| /api/gastos/{id} (DELETE) | Eliminar gasto | ⏳ | En progreso |  |

Para ms\_income se realizaron pruebas similares en backend: creación y obtención de ingresos.

Se dispone de una colección Postman exportada y configurada con variables de entorno y scripts de prueba (cuando aplique).

**8.1.2 Pruebas de integración Frontend-Backend ✅**

Escenarios evaluados:

* Envío de gasto desde la interfaz
* Visualización de gastos en tabla
* Aplicación de filtros
* Manejo de respuestas HTTP y errores

La integración con ms\_expense funciona correctamente. La conexión con ms\_income está pendiente.

**8.1.3 Pruebas unitarias (backend)**

* Cobertura estimada: ~ 65 %
* Pruebas en servicios, repositorios y controladores
* Uso de frameworks JUnit + Mockito

**8.2 Problemas encontrados y soluciones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Problema** | **Severidad** | **Solución aplicada** | **Estado** |
| Filtro por fecha con error en consulta | Media | Ajuste de la consulta SQL o JPQL | ✅ Resuelto |
| No integración de ms\_income con frontend | Media | Pendiente de desarrollo | ⏳ En progreso |
| CORS entre frontend y backend | Baja | Configuración de @CrossOrigin o proxy | ✅ Resuelto |
| Timeout en peticiones grandes | Baja | Ajuste de timeout y paginación | ✅ Resuelto |

**8.3 Evaluación de rendimiento**

Se midieron:

* Tiempo de respuesta de endpoints (promedio: ~200–300 ms)
* Tiempo de carga de interfaz (~1.5 segundos)
* Uso de memoria y CPU en ambiente local

Los resultados son adecuados para pruebas locales, aunque no simulando alta carga.

**8.4 Escalabilidad**

* Arquitectura lista para escalado horizontal
* Base de datos PostgreSQL apta para crecimiento
* Docker permite replicar servicios
* No se ha probado con alta concurrencia
* No hay balanceador de carga ni pruebas de resistencia

**8.5 Tolerancia a fallos**

Implementaciones actuales:

* Manejo centralizado de excepciones
* Mensajes de error claros
* (Pendiente) circuit breakers, reintentos y fallback entre microservicios

**9. Documentación**

**9.1 Documentación Técnica**

**Documentos generados:**

* Arquitectura del sistema (diagramas)
* Modelo de datos (ER y diccionario)
* Documentación de APIs (endpoints, ejemplos)

**Pendiente:**

* Guía de despliegue (Jenkins, Docker, configuración)

**9.2 Manual del Usuario**

* En desarrollo: guía de navegación, tutoriales, FAQ

**9.3 Documentación en el Código**

* JavaDoc y comentarios
* README principal

**9.4 Control de versiones de documentación**

Estructura recomendada:

docs/

├── arquitectura/

├── database/

├── api/

├── deployment/

└── user-manual/

**10. Planificación Scrum y Gestión del Proyecto**

**10.1 Roles y Responsabilidades**

(Tal como ya lo tienes, pero con mayor formalidad)

|  |  |  |
| --- | --- | --- |
| **Rol** | **Integrante** | **Responsabilidades** |
| Product Owner | Miguel Ángel Beltrán Bohórquez | Definir requisitos, priorización, validación, comunicación con stakeholders |
| Desarrollador Backend | Nicolás Rivera Aroca | Desarrollo de microservicios, diseño de APIs, lógica de negocio |
| Desarrollador Frontend | Jesús Candelo | Desarrollo de interfaz, integración con backend, validaciones cliente |

**10.2 Herramientas Scrum**

* [Jira / Trello / GitHub Projects]
* Tableros: backlog, en progreso, en revisión, completado

**10.3 Gestión del Alcance**

**Alcance original vs alcance actual:**

* Original: 5 microservicios, frontend completamente integrado, autenticación, dashboard
* Actual (55 %): ms\_expense y ms\_income operativos, frontend parcial, microservicios restantes en desarrollo

Desviación estimada: 10 %–15 %  
Acciones correctivas: re-priorizar entregables, enfocar en funcionalidades clave

**10.4 Sprints Realizados**

Puedes enumerar cada sprint indicando fechas, objetivos, entregables y lecciones aprendidas.

**10.5 Ceremonias Scrum**

* Standups diarios (15 minutos)
* Sprint planning
* Sprint review
* Sprint retrospective

**11. Análisis de Perspectivas**

**11.1 Perspectiva Técnica**

**Fortalezas:**

* Arquitectura modular
* Uso de tecnologías modernas
* Separación clara de responsabilidades
* Automatización del despliegue

**Áreas de mejora:**

* Mayor cobertura de pruebas
* Documentación con Swagger/OpenAPI
* Logging centralizado
* Gestión de errores entre servicios

**11.2 Perspectiva Económica**

* Costos actuales: prácticamente nulos (desarrollo en máquinas locales)
* Para producción: costos de infraestructura en nube, mantenimiento, escalamiento
* Posible modelo de monetización: subscripción, versiones premium

**11.3 Perspectiva Social**

* Facilita organización financiera personal
* Accesible desde dispositivos con navegador
* Ahorro de tiempo y esfuerzo

**11.4 Perspectiva Ética**

* Necesidad de protección de datos financieros
* Transparencia y control para el usuario
* Seguridad (encriptación, HTTPS, etc.)

**11.5 Perspectiva de Sostenibilidad**

* Código modular facilita mantenimiento
* Docker permite uso eficiente de recursos
* Escalabilidad futura con mínima re-arquitectura

**12. Lecciones Aprendidas**

**12.1 Técnicas**

* Diseño y construcción de microservicios
* Integración frontend-backend modernas
* Pipelines CI/CD con Jenkins y Docker
* Configuración de servicios distribuidos

**Dificultades:**

* Comunicación entre servicios (CORS, fallback)
* Estimación de tiempos
* Coordinación de tareas simultáneas

**12.2 Trabajo en Equipo**

**Fortalezas:**

* División clara de tareas
* Comunicación constante
* Apoyo mutuo

**Mejoras:**

* Sincronización más frecuente
* Revisión de tareas antes de implementarlas

**12.3 Gestión del Proyecto**

**Buenas prácticas:**

* Uso de control de versiones
* Reuniones regulares
* Documentación iterativa

**Mejoras para el futuro:**

* Estimaciones más realistas
* Retroalimentación continua
* Documentación desde etapas iniciales

**12.4 Conocimientos Adquiridos**

* Spring Boot, Angular, PostgreSQL
* Docker, Jenkins, microservicios
* Scrum, ABP, metodologías ágiles

**13. Próximos Pasos**

**13.1 Tareas Inmediatas (corto plazo)**

1. Integrar **ms\_income** con el frontend
2. Desarrollar **ms\_categorias**
3. Desarrollar **ms\_usuario**
4. Desarrollar **ms\_login**
5. Implementar el **dashboard con gráficos** (Chart.js, D3.js)
6. Elaborar el **manual de usuario**

**13.2 Funcionalidades Futuras (mediano plazo)**

* Exportar reportes en PDF
* Notificaciones de presupuesto excedido
* Análisis predictivo de gastos
* Sincronización con cuentas bancarias
* Aplicación móvil (iOS / Android)

**13.3 Mejoras Técnicas Pendientes**

* Seguridad: JWT, HTTPS, rate limiting
* Pruebas: integración automatizada, E2E
* DevOps: monitoreo (Prometheus / Grafana), alertas
* Documentación: Swagger / OpenAPI, guías completas

**13.4 Cronograma Estimado de Finalización**

|  |  |  |  |
| --- | --- | --- | --- |
| **Actividad** | **Inicio** | **Fecha estimada de fin** | **Estado** |
| Integración ms\_income | 15 oct 2025 | 25 oct 2025 | En progreso |
| Desarrollo ms\_categorias | 26 oct 2025 | 10 nov 2025 | Planeado |
| Desarrollo ms\_usuario | 26 oct 2025 | 10 nov 2025 | Planeado |
| Desarrollo ms\_login | 10 nov 2025 | 25 nov 2025 | Planeado |
| Dashboard / reportes | 26 nov 2025 | 10 dic 2025 | Planeado |
| Pruebas finales | 11 dic 2025 | 20 dic 2025 | Planeado |
| Documentación completa | 11 dic 2025 | 22 dic 2025 | Planeado |
| **Entrega final** | — | **24 dic 2025** | — |

**14. Conclusiones**

**14.1 Estado General**

El proyecto ha alcanzado un avance del **55 %**, con microservicios clave ya desplegados y funcionales. La arquitectura propuesta está demostrando su viabilidad técnica.

**14.2 Logros Principales**

* Correcta implementación de arquitectura modular
* Automatización del despliegue con Jenkins y Docker
* Integración del frontend con ms\_expense
* Validación funcional de endpoints

**14.3 Desafíos Pendientes**

* Integrar ms\_income al frontend
* Completar desarrollo de los microservicios faltantes
* Implementar seguridad robusta
* Documentación de usuario y guía de despliegue

**14.4 Viabilidad**

El proyecto muestra alta viabilidad técnica. Con las tareas planificadas, se estima alcanzar entre 90 % y 95 % de la funcionalidad esperada para la presentación final.

**14.5 Aprendizajes**

El equipo adquirió experiencia práctica en microservicios, integración tecnológica, trabajo colaborativo y metodologías ágiles.

**15. Anexos**

**Anexo A: Estructura del Repositorio**

gestion-gastos/

├── backend/

│ ├── ms-expense/

│ │ ├── src/

│ │ ├── Dockerfile

│ │ └── pom.xml

│ ├── ms-income/

│ │ ├── src/

│ │ ├── Dockerfile

│ │ └── pom.xml

│ └── [otros microservicios]/

├── frontend/

│ ├── src/

│ ├── Dockerfile

│ └── package.json

├── database/

│ └── scripts/

├── docs/

│ ├── arquitectura/

│ ├── api/

│ └── database/

├── jenkins/

│ └── Jenkinsfile

└── docker-compose.yml

**Anexo B: Diagramas**

Incluye:

* Diagrama de arquitectura
* Diagrama entidad-relación (ER)
* Flujo de usuario
* Diagrama de despliegue

**Anexo C: Enlaces relevantes**

* Repositorio GitHub (URL)
* Dashboard Jenkins
* APIs desplegadas (local)
* Documentación de APIs

**Anexo D: Glosario**

| **Término** | **Definición** |
| --- | --- |
| Microservicio | Servicio independiente dentro de una arquitectura distribuida |
| API REST | Interfaz basada en HTTP para comunicación entre componentes |
| CI/CD | Integración Continua y Despliegue Continuo |
| Docker | Plataforma de contenedorización de aplicaciones |
| Jenkins | Herramienta de automatización para pipelines |
| Spring Boot | Framework Java para desarrollo de aplicaciones empresariales |
| Angular | Framework frontend de JavaScript/TypeScript |
| PostgreSQL | Sistema de gestión de bases de datos relacional |

**Formato APA**

Aquí algunas sugerencias para aplicar estilo APA al documento:

* **Portada:** título en mayúsculas centrado, nombre del autor(es), institución, curso, fecha.
* **Encabezados:** usa el estilo APA de encabezados (nivel 1,2,3…), siempre consistente.
* **Citas:** si llegas a incluir referencias bibliográficas, usa formato APA (Autor, año).
* **Formato:** márgenes de 2,54 cm, interlineado doble (2.0), fuente Times New Roman 12 pt.
* **Numeración de páginas:** incluir número de página en el encabezado derecho.
* **Tablas y figuras:** numeradas (Tabla 1, Figura 1) con título descriptivo arriba y fuente abajo.

**Portada sugerida (para pegar en Word)**

UNIVERSIDAD / INSTITUCIÓN

FACULTAD / PROGRAMA

INFORME DE AVANCE

FASE 3 – IMPLEMENTACIÓN Y EVALUACIÓN

“SISTEMA DE GESTIÓN DE GASTOS”

Arquitectura de Microservicios

Elaborado por:

Miguel Ángel Beltrán Bohórquez

Nicolás Rivera Aroca

Jesús Candelo

Neiva, Huila

Octubre de 2025