PLAN DE PRUEBAS - EASYBANK

Información General del Plan

Objetivo

Verificar que el sistema EasyBank cumpla con todos los requisitos funcionales y no funcionales especificados en las historias de usuario, garantizando la calidad del software antes de su implementación.

Alcance

Este plan cubre pruebas funcionales y no funcionales para las historias de usuario definidas en el documento de requisitos.

Estrategia de Pruebas

Tipo de Prueba	Descripción
Pruebas Funcionales	Verificación de criterios de aceptación
Pruebas No Funcionales	Seguridad, rendimiento y usabilidad

Niveles de Prueba

- Unitarias
- Integración
- Sistema
- Aceptación

Casos de Prueba por Historia de Usuario

💡 HU - 1 - Reducción del Consumo de Recursos del Navegador

CP-001: Prueba Funcional – Medición de Consumo de Memoria Bajo el 70%

Tipo: Funcional – Rendimiento

Prioridad: 🔴Alta

Precondiciones

- La aplicación está desplegada en un entorno estable o ejecutada de manera local.
- El navegador está actualizado.
- No existen extensiones que interfieran con la medición de rendimiento.
- El inspector del navegador está disponible para monitorear memoria en tiempo real.

Pasos de Ejecución

1. Abrir la aplicación en una pestaña nueva.
2. Abrir el panel de rendimiento del navegador ya sea Performance o Memory.
3. Navegar entre las secciones principales de la aplicación por 3 minutos.
4. Observar el consumo de memoria y registrar el promedio.
5. Repetir la prueba 3 veces.

Resultado Esperado

- El uso de memoria debe mantenerse **por debajo del 70% del promedio actual** documentado antes de la optimización.

💡 HU - 2 - Optimización de Carga Inicial del Dashboard

CP-002: Prueba Funcional – Carga Inicial del Dashboard en < 2 Segundos

Tipo: Funcional – Rendimiento

Prioridad: 🔴Alta

Precondiciones

- El usuario debe tener una cuenta activa y con datos asociados.
- La aplicación debe estar desplegada o en un entorno local accesible.
- El navegador debe estar con cache limpio para medir el tiempo real.
- Herramientas de medición disponibles Performance de Chrome o Lighthouse.

Pasos de Ejecución

1. Abrir la pantalla de login del sistema.
2. Iniciar sesión con un usuario válido.
3. Iniciar la medición del tiempo de carga desde el momento en que se envía el formulario.
4. Registrar el tiempo que tarda en aparecer el Dashboard completamente interactivo.
5. Verificar el orden de carga de los elementos.
6. Realizar una acción inmediata para validar que la UI no esté bloqueada.

Resultado Esperado

- El Dashboard debe cargar **en menos de 2 segundos** en condiciones normales.
- Los elementos principales deben aparecer **primero**.
- El usuario debe poder **interactuar sin bloqueos** mientras se cargan elementos secundarios.

💡 HU - 3 - Optimización del Rendimiento del Módulo de Gestión

CP-003: Prueba Funcional – Carga Rápida y Fluida del Módulo de Gestión

Tipo: Funcional – Rendimiento

Prioridad: 🔴Alta

Precondiciones

- El usuario debe tener datos existentes de categorías, balance y gastos.
- La aplicación debe estar funcionando en un entorno estable.
- La cache del navegador debe estar limpia para medir el tiempo real.
- Herramientas de medición disponibles ya sea Performance o Lighthouse.

Pasos de Ejecución

1. Iniciar sesión con un usuario válido.
2. Navegar hacia el módulo de gestión.
3. Iniciar la medición del tiempo desde el clic hasta que el módulo sea visible.
4. Verificar que el tiempo total de carga no supere los 2 segundos.
5. Revisar que los datos principales como balance y categorías aparezcan de manera progresiva sin bloquear la UI.
6. Probar la acción **Agregar Gasto** inmediatamente y comprobar que el sistema responde sin retraso perceptible.

Resultado Esperado

- El módulo de gestión debe cargar **en menos de 2 segundos**.
- Los datos deben mostrarse de forma **progresiva**, sin congelar la interfaz.
- La acción de **agregar gasto** debe responder **de forma inmediata**, sin lag ni bloqueos.

💡 HU - 4 - Optimización de carga inicial

CP-004: Prueba de Validación – Carga de datos en tiempo solicitado

Tipo: Rendimiento / No funcional

Prioridad: 🔴 Alta

Precondiciones:

- Credenciales de ADMIN válidas.
- Existen datos de clientes y transacciones en la base de datos para poblar la tabla.
- Conexión a red estable

Pasos de Ejecución:

1. Acceder con credenciales válidas y rol ADMIN
2. Verificar que los datos del panel principal carguen de manera correcta en el tiempo esperado
3. Dar al botón iniciar sesión y esperar a ser redirigido al panel de administrador

Resultado Esperado:

- Tablas con los usuarios cargadas con los datos correctamente

💡 HU - 5 - Optimización del sistema de búsqueda de cliente

CP-005: Prueba de Validación – Búsqueda de clientes

Tipo: Rendimiento / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Credenciales de ADMIN válidas.

- Existen datos de clientes y transacciones en la base de datos para poblar la tabla.
- Conexión a red estable

Pasos de Ejecución:

1. En el panel de principal de administrador, colocar un id de un cliente
2. Verificar que los datos del cliente solicitado sean los correctos

Resultado Esperado:

- Datos coincidentes con la busqueda solicitada

💡 HU - 6 - Carga eficiente del historial de transacciones

CP-006: Prueba de Validación – Carga de transacciones eficiente

Tipo: Rendimiento / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Credenciales de ADMIN válidas.
- El sistema debe contar con un volumen significativo de transacciones para probar la paginación.
- Conexión a red estable

Pasos de Ejecución:

1. En el panel de administrador, navegar al módulo de Transacciones.
2. Verificar que la tabla de transacciones cargue y mida la cantidad de filas mostradas.

Resultado Esperado:

- Los datos deben aparecer en las tablas con un tiempo no superior a 4 segundos
- Los datos deben mostrarse paginados

💡 HU-7 – Mejora en la Confirmación Visual de Transacciones

CP-007: Prueba de Validación – Confirmación sobre una transacción

Tipo: Rendimiento / No funcional

Prioridad: 🟡 Media

Precondiciones:

- Credenciales de ADMIN válidas.
- Usuario ubicado en el modulo de transacciones
- Datos listos para una transacción valida.

Pasos de Ejecución:

1. Completar el formulario para realizar una transacción o deposito.
2. Hacer click en el botón de registrar el deposito o transacción.
3. Medir el tiempo entre el click y la aparición del mensaje.

Resultado Esperado:

- Confirmación visual de la realización de la transacción o deposito.

🧪 HU-8 – Incorporación de historial detallado de transacciones

CP-008: Prueba de Validación – Carga de datos en tiempo solicitado

Tipo: Rendimiento / No funcional

Prioridad: 🔴 Alta

Precondiciones:

- Credenciales de ADMIN válidas.
- Existen datos de clientes y transacciones en la base de datos para poblar la tabla.

- Conexión a red estable

Pasos de Ejecución:

1. Acceder con credenciales válidas y rol ADMIN
2. Verificar que los datos del panel principal carguen de manera correcta en el tiempo esperado
3. Dar al botón iniciar sesión y esperar a ser redirigido al panel de administrador

Resultado Esperado:

- Tablas con los usuarios cargadas con los datos correctamente

CP-009: Filtro de transacciones por fecha

Tipo: Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Usuario autenticado.
- Existen transacciones registradas en diferentes fechas.

Pasos de Ejecución:

1. Enviar GET `/api/transactions?from=2024-01-01&to=2024-01-31`.

Resultado Esperado:

- `200 OK`.
- Se muestran **solo** las transacciones dentro del rango.
- Cada registro muestra: **fecha**, **descripción**, **monto**, **tipo**.

CP-010: Filtro de transacciones por tipo

Tipo: Funcional

Prioridad: 🟡 Media

Precondiciones:

- Usuario autenticado.
- Existen transacciones de distintos tipos (ej. `"INGRESO"`, `"GASTO"`, `"TRANSFERENCIA"`).

Pasos de Ejecución:

1. Enviar GET `/api/transactions?type=GASTO`.

Resultado Esperado:

- `200 OK`.
- Solo aparecen transacciones cuyo tipo es `"GASTO"`.
- Cada registro incluye: fecha, descripción, monto y tipo.

CP-011: Filtros combinados (fecha + tipo)

Tipo: Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Usuario autenticado.
- BD con suficiente variedad en fechas y tipos.

Pasos de Ejecución:

1. Enviar GET: `/api/transactions?from=2024-01-01&to=2024-01-31&type=INGRESO`

Resultado Esperado:

- `200 OK`.
- Se muestran únicamente las transacciones del tipo INDICADO dentro del rango.
- Se incluyen los campos completos por transacción.

CP-012: Validación – Rango de fechas inválido

Tipo: Validación

Prioridad: 🔴 Alta

Precondiciones:

- Usuario autenticado.

Pasos de Ejecución:

1. Enviar GET `/api/transactions?from=2024-02-01&to=2024-01-01`.

Resultado Esperado:

- `400 Bad Request`.
- Mensaje indicando que la fecha inicial no puede ser mayor que la final.
- Manejado por el GlobalExceptionHandler.

CP-013: Sin transacciones registradas

Tipo: Funcional

Prioridad: 🟢 Baja

Precondiciones:

- Usuario autenticado.
- Usuario sin transacciones.

Pasos de Ejecución:

1. Enviar GET `/api/transactions`.

Resultado Esperado:

- `200 OK`.
- Lista vacía (`data: []` si está paginado).
- Metadatos de paginación correctos (si aplica).
- Sin errores.

💡 HU - 11 - Incorporar validaciones en los DTOs

CP-014: Prueba de Validación – DepositRequestDTO inválido

Tipo: Validación / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Usuario ADMIN logueado.
- Cuenta de usuario existe.

Pasos de Ejecución:

1. Enviar POST `/api/admin/userlist/{id}/deposit` con payload:

```
{
  "accountId": "uuid-correcto",
  "amount": -100,
  "description": "Test deposit"
}
```

Resultado Esperado:

- Código `400 Bad Request`.
- Mensaje de error.

🧪 HU - 12 - Implementar paginación en respuestas de listas

CP-015: Prueba de Paginación – Parámetros válidos

Tipo: Funcional / Validación

Prioridad: 🟡 Media

Precondiciones:

- Usuario ADMIN o USER logueado.
- Existen al menos 20 registros en la base de datos.

Pasos de Ejecución:

1. Enviar GET `/api/resource?page=0&size=10&sort=id,asc`.

Resultado Esperado:

- Código `200 OK`.
- `content` retorna máximo 10 elementos.
- `page = 0`, `size = 10`.
- `totalElements`, `totalPages` presentes.
- Respuesta envuelta en `ApiResponse`.

CP-016: Prueba de Paginación – Parámetros omitidos

Tipo: Funcional

Prioridad: 🟡 Media

Precondiciones:

- Usuario logueado.
- Existen registros en BD.

Pasos de Ejecución:

1. Enviar GET `/api/resource` sin parámetros.

Resultado Esperado:

- Código `200 OK`.
- Se aplican valores por defecto (ej: `page = 0`, `size = 20`).
- Respuesta incluye metadatos de paginación.

CP-017: Prueba de Paginación – Valores fuera de rango

Tipo: Validación

Prioridad: 🔴 Alta

Precondiciones:

- Usuario logueado.

Pasos de Ejecución:

1. Enviar GET `/api/resource?page=-1&size=-5`.

Resultado Esperado:

- `400 Bad Request`.
- Mensaje indicando parámetros inválidos.
- Manejados por el GlobalExceptionHandler.
- Estructura consistente de error.

CP-018: Prueba de Paginación – Página mayor al total

Tipo: Funcional

Prioridad: 🟡 Media

Precondiciones:

- BD con pocos registros (ej. totalPages = 3).

Pasos de Ejecución:

1. Enviar GET `/api/resource?page=99&size=10`.

Resultado Esperado:

- `200 OK`.
- Listado vacío: `content: []`.
- Metadatos correctos: `page=99`, `totalPages=3`.
- No se produce error.

💡 HU - 13 - Mejorar manejo de excepciones en el backend

CP-019: Prueba de Excepción – Usuario no encontrado

Tipo: Negativa / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Token JWT válido de ADMIN.

Pasos de Ejecución:

1. Llamar GET `/api/admin/userlist/{id}` con un UUID que no existe.

Resultado Esperado:

- Código `404 Not Found`.
- Mensaje de error: `"User not found with id: {uuid}"`.

💡 HU - 14 - Cambio a Soft Delete de Usuarios

CP-020: Eliminación Lógica Exitosa

Tipo: Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Usuario ADMIN logueado con token JWT válido.
- Usuario objetivo existe en la base de datos y `active = true`.
- Sistema de base de datos operativo.

Pasos de Ejecución:

1. Llamar endpoint `DELETE /api/admin/userlist/{id}` con el UUID del usuario objetivo.
2. Verificar la respuesta del endpoint.
3. Consultar nuevamente la lista de usuarios: `GET /api/admin/userlist`.
4. Consultar directamente la base de datos el registro del usuario eliminado.

Resultado Esperado:

- Código HTTP `202 Accepted` en la respuesta.
- Mensaje: `"Successfully deleted user"`.
- El usuario ya no aparece en la lista de usuarios activos (`GET /api/admin/userlist`).
- El campo `active` del usuario está marcado como `false` .
- Transacciones, cuentas y bills relacionados permanecen intactos.

🧪 HU - 15 - Mejorar manejo de errores en el registro de usuario y cuenta

CP-021: Registro falla al crear cuenta → rollback

Tipo: Transaccional / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Endpoint `/api/auth/register` activo.
- Mock o simulación de error al crear la cuenta.

Pasos de Ejecución:

1. Enviar POST `/api/auth/register` con payload válido.

2. Forzar fallo en la creación de cuenta (ej: excepción simulada).

Resultado Esperado:

- `500 Internal Server Error` o el código definido.
- Usuario **no queda registrado en la BD**.
- Error manejado por `GlobalExceptionHandler`.

CP-022: Validación de datos inválidos en registro

Tipo: Validación / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Endpoint de registro disponible.

Pasos de Ejecución:

1. Enviar POST `/api/auth/register` con payload inválido (email sin formato, password corta, etc.)

Resultado Esperado:

- `400 Bad Request`.
- Respuesta con errores de validación.
- GlobalExceptionHandler devuelve mensajes claros.
- No se guarda nada en la BD.

CP-023: Error por duplicado (email o username)

Tipo: Validación / Funcional

Prioridad: 🔴 Alta

Precondiciones:

- Ya existe un usuario registrado con ese email/username.

Pasos de Ejecución:

1. Enviar POST `/api/auth/register` con email o username duplicado.

Resultado Esperado:

- `409 Conflict` (o el código definido).

- Mensaje claro indicando duplicado.
- Capturado y formateado por GlobalExceptionHandler.
- No se crea usuario duplicado.

💡 HU - 16 - Implementar control de acceso con @PreAuthorize

CP-024: Prueba Funcional – Acceso autorizado para ADMIN

Tipo: Funcional

Prioridad: 🔴 Alta

Precondiciones:

- El usuario está registrado como ADMIN.
- El token JWT es válido.
- Sistema operativo y servidor corriendo.

Pasos de Ejecución:

1. Realizar login como usuario ADMIN.
2. Obtener token JWT.
3. Llamar al endpoint `/api/admin/userlist` con token.

Resultado Esperado:

- El endpoint retorna código `200 OK`.
- Se devuelve la lista de usuarios.
- El sistema no lanza errores de autorización.

🧪 HU - 17 - Dividir la lógica de registro en submétodos

CP-025: Validar que el registro usa submétodos

Tipo: Técnico / Refactor

Prioridad: 🟡 Media

Precondiciones:

- Código refactorizado.

Pasos de Ejecución:

1. Revisar estructura del método `register()`.
2. Confirmar separación en submétodos:
    - validarDatos()
    - crearUsuario()
    - crearCuenta()
    - asignarRol()
    - guardar()

Resultado Esperado:

- Método `register()` reducido y legible.
- Código duplicado eliminado.
- Flujo modular y desacoplado.

CP-026: Comportamiento del registro después del refactor

Tipo: Funcional

Prioridad: 🟡 Media

Precondiciones:

- Refactor completado.

Pasos de Ejecución:

1. Enviar POST `/api/auth/register` con payload válido.

Resultado Esperado:

- `201 Created`.
- Usuario y cuenta creados correctamente.
- No rompe la funcionalidad original.

Matriz de Trazabilidad

Historia de Usuario	Casos de Prueba Asociados	Tipo de Prueba	Prioridad
HU-1 – Reducción del Consumo de Recursos del Navegador	CP-001	Funcional – Rendimiento	🔴 Alta
HU-2 – Optimización de Carga Inicial del Dashboard	CP-002	Funcional – Rendimiento	🔴 Alta
HU-3 – Optimización del Módulo de Gestión	CP-003	Funcional – Rendimiento	🔴 Alta
HU-4 – Optimización de Carga Inicial (Admin Panel)	CP-004	Rendimiento / No Funcional	🔴 Alta
HU-5 – Optimización del Sistema de Búsqueda de Cliente	CP-005	Funcional / Rendimiento	🔴 Alta
HU-6 – Carga eficiente del historial de transacciones	CP-006	Funcional / Rendimiento	🔴 Alta
HU-7 – Confirmación Visual de Transacciones	CP-007	No funcional / Rendimiento	🟡 Media
HU-9 – Historial Detallado de Transacciones	CP-008, CP-009, CP-010, CP-011, CP-012, CP-013	Funcional / Validación / Rendimiento	🔴 Alta
HU-11 – Validaciones en los DTOs	CP-014	Validación / Funcional	🔴 Alta
HU-12 – Implementar Paginación	CP-015, CP-016, CP-017, CP-018	Funcional / Validación	🟡 Media – 🔴 Alta
HU-13 – Manejo Mejorado de Excepciones	CP-019	Funcional / Negativa	🔴 Alta
HU-14 – Cambio a Soft Delete	CP-020	Funcional	🔴 Alta

Historia de Usuario	Casos de Prueba Asociados	Tipo de Prueba	Prioridad
HU-15 – Manejo de Errores en Registro y Cuenta	CP-021, CP-022, CP-023	Validación / Transaccional	🔴 Alta
HU-16 – Control de Acceso con @PreAuthorize	CP-024	Funcional	🔴 Alta
HU-17 – Dividir Lógica del Registro en Submétodos	CP-025, CP-026	Técnico / Funcional	🟡 Media

Criterios de Aceptación del Plan

Criterios de Entrada

- Ambiente de pruebas configurado
- Datos de prueba cargados
- Casos de prueba revisados y aprobados
- Herramientas de prueba disponibles

Criterios de Salida

- 100% de casos de prueba ejecutados
- 0 defectos críticos sin resolver
- <5 defectos medios sin resolver
- Tasa de éxito >95%
- Documentación de pruebas completa

Criterios de Suspensión

- Más de 3 defectos críticos bloqueantes
- Ambiente de pruebas inestable
- Falta de datos de prueba críticos

Criterios de Reanudación

- Defectos bloqueantes resueltos
- Ambiente estabilizado
- Datos de prueba restaurados

Recursos y Herramientas

Herramientas de Prueba

Categoría	Herramientas
Funcionales	Postman
Rendimiento	Performance de Chrome
Gestión	JIRA

Ambiente de Pruebas

Componente	Especificación
Base de Datos	PostgreSQL
Sistema Operativo	Windows, Linux
Navegadores	Chrome, Edge

Equipo de Pruebas

Rol	Recursos	Responsabilidades
Líder de Pruebas	1 persona	Coordina y supervisa
Testers	4 personas	Ejecutan pruebas funcionales

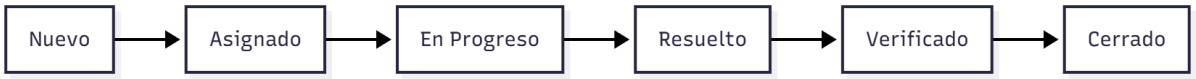
Gestión de Defectos

Clasificación de Severidad

Nivel	Descripción	Impacto
Crítico	Sistema inoperativo, pérdida de datos	🔴 Bloqueador
Alto	Funcionalidad principal afectada	🟠 Alto
Medio	Funcionalidad secundaria afectada	🟡 Moderado

Nivel	Descripción	Impacto
Bajo	Problemas cosméticos o menores	🟢 Menor

Ciclo de Vida del Defecto



Métricas de Calidad

- Densidad de defectos por módulo
- Tiempo promedio de resolución
- Tasa de reapertura de defectos
- Cobertura de pruebas

Riesgos y Mitigación

Riesgo	Impacto	Probabilidad	Mitigación
Datos de prueba insuficientes	🔴 Alto	🟡 Media	Generar datos sintéticos
Cambios en requisitos	🔴 Alto	🟡 Media	Pruebas iterativas
Ambiente inestable	🔴 Alto	🟢 Baja	Ambiente de respaldo
Falta de recursos	🟡 Medio	🟢 Baja	Priorización de pruebas

Conclusiones

Este plan de pruebas asegura cobertura completa de las nuevas modificaciones en el sistema EasyBank, validando cada historia de usuario. Su ejecución garantiza un sistema robusto, seguro y fácil de usar para usuarios y administradores.

💡 Recomendaciones

- Ejecutar pruebas de regresión tras cada corrección
- Mantener documentación actualizada

- Realizar pruebas de aceptación con usuarios reales
- Establecer métricas de calidad continuas

📋 Información del Documento

Campo	Información
Documento preparado para	EasyBank
Encargados	Axel Alvarado, Oscar Arguera, Juan Castellanos, Valeria Ortiz
Fecha	Noviembre 2025
Versión	1

📧 Para consultas sobre este plan de pruebas, contactar al equipo de QA