UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

Departamento de Ciencias de la Computación

Análisis y Diseño de Software

Informe de Pruebas - PagoSeguro Backend V3.0.0

Sistema de Gestión de Créditos y Pagos

Pruebas Unitarias (Jest) y Pruebas de Carga (K6)

Fecha de Ejecución:	4 de febrero de 2026
Versión del Sistema:	3.0.0
Ambiente:	Desarrollo local con Docker

Sangolquí, Ecuador

Índice

Índice de figuras

Índice de cuadros

1 Introducción

El presente informe documenta el proceso de diseño, ejecución y análisis de las pruebas implementadas en el sistema **PagoSeguro Backend V3.0.0**, una plataforma financiera orientada a la gestión de créditos, pagos electrónicos y generación de reportes para el sector agrícola.

Las pruebas implementadas incluyen:

- **Pruebas Unitarias con Jest:** Validación de la lógica de negocio en controladores y servicios.
- **Pruebas de Carga con K6:** Evaluación del rendimiento bajo múltiples usuarios concurrentes.

El backend está desarrollado con **Node.js**, **Express** y **TypeScript**, siguiendo una arquitectura limpia con separación de responsabilidades.

2 Objetivo de las Pruebas

2.1 Objetivos Generales

- Verificar que cada función o método del sistema cumpla con los requisitos definidos.
- Detectar errores de forma temprana en la lógica del negocio.
- Asegurar la correcta validación de datos de entrada y salida.
- Evaluar el rendimiento del sistema bajo carga concurrente.
- Garantizar la seguridad en los endpoints de autenticación.

2.2 Objetivos Específicos

- Validar el flujo completo de autenticación (login, registro, logout, recuperación de contraseña).
- Verificar las operaciones CRUD del servicio de créditos.
- Medir tiempos de respuesta bajo carga de 10 usuarios virtuales.
- Asegurar que el sistema responde correctamente a credenciales inválidas.

3 Alcance

Las pruebas cubren los siguientes módulos del sistema:

Cuadro 1: Módulos cubiertos por las pruebas

Módulo	Tipo de Prueba	Cobertura
AuthController	Jest (Unitaria)	100 %
CreditService	Jest (Unitaria)	100 %
Endpoint /api/v1/auth/login	K6 (Carga)	Rendimiento
Endpoint /api/v1/credits	K6 (Carga)	Rendimiento

4 Herramientas Utilizadas

4.1 Jest - Framework de Pruebas Unitarias

Jest es un framework de pruebas para JavaScript/TypeScript desarrollado por Facebook. Se utilizó para:

- Pruebas unitarias de controladores y servicios
- Mocking de dependencias (repositorios, servicios externos)
- Validación de respuestas HTTP

Configuración: `jest.config.cjs` con soporte para TypeScript mediante `ts-jest`.

4.2 K6 - Herramienta de Pruebas de Carga

K6 es una herramienta moderna de pruebas de carga desarrollada por Grafana Labs. Se utilizó para:

- Simulación de usuarios virtuales concurrentes
- Medición de tiempos de respuesta (P90, P95)
- Validación de thresholds de rendimiento

Ejecución: Mediante contenedor Docker `grafana/k6`.

5 Estrategia de Pruebas

5.1 Principio AAA (Arrange-Act-Assert)

Cada prueba unitaria sigue el principio **AAA**:

- **Arrange:** Preparación del contexto, mocks y datos de prueba.
- **Act:** Ejecución del método o función bajo prueba.
- **Assert:** Comparación del resultado obtenido con el esperado.

5.2 Aislamiento de Dependencias

Las dependencias externas son simuladas mediante *mocks*:

- `IUserRepository` - Repositorio de usuarios
- `IRefreshTokenRepository` - Repositorio de tokens
- `ITokenService` - Servicio de generación de JWT
- `ICreditRepository` - Repositorio de créditos
- `AuditService` - Servicio de auditoría

6 Pruebas Unitarias Implementadas (Jest)

6.1 AuthController.test.ts

Este archivo contiene las pruebas del controlador de autenticación, validando los endpoints de login, registro, logout y recuperación de contraseña.

6.1.1 Estructura del Test

```
import { AuthService } from "../../src/services/AuthService";
import { AuthController } from "../../src/controllers/AuthController";
import { beforeEach, describe, it, expect, jest } from "@jest/globals";
import {
  IUserRepository,
  IRefreshTokenRepository,
  ITokenService,
  IAuditLogRepository,
} from "../../src/lib/interfaces";
```

```
import { UserDomainService } from "../../src/services/UserDomainService
    ";
import { AuditService } from "../../src/services/AuditService";
import { UserRole } from "../../src/models/User";

jest.mock("../../src/services/UserDomainService");
jest.mock("../../src/services/PasswordHashingService");

describe("AuthController", () => {
  let authService: AuthService;
  let authController: AuthController;
  let mockUserRepository: jest.Mocked<IUserRepository>;
  let mockRefreshTokenRepository: jest.Mocked<IRefreshTokenRepository>;
  let mockTokenService: jest.Mocked<ITokenService>;
  let mockUserDomainService: jest.Mocked<UserDomainService>;
  let mockAuditService: jest.Mocked<AuditService>;

  beforeEach(() => {
    // Configuracion de mocks...
  });
});
```

Listing 1: AuthController.test.ts - Configuración de Mocks

6.1.2 Casos de Prueba - Login

```
it("should return 401 for invalid credentials when user not found",
    async () => {
  const mockRequest = {
    body: {
      email: "testuser@example.com",
      password: "wrongpassword",
    },
    ip: "127.0.0.1",
    socket: { remoteAddress: "127.0.0.1" },
    headers: { "user-agent": "test-agent" },
  } as any;

  const mockResponse = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
    cookie: jest.fn(),
  } as any;

  mockUserRepository.findByEmail.mockResolvedValue(null);
```

```
  mockAuditService.logLoginFailure.mockResolvedValue(undefined);

  await authController.login(mockRequest, mockResponse);

  expect(mockResponse.status).toHaveBeenCalledWith(401);
  expect(mockResponse.json).toHaveBeenCalledWith({
    success: false,
    message: "Credenciales incorrectas",
  });
});
```

Listing 2: Prueba de login con credenciales inválidas

6.1.3 Casos de Prueba - Logout

```
it("should logout user successfully", async () => {
  const mockRequest = {
    cookies: { refreshToken: "validRefreshToken" },
    user: { userId: "user1" },
    ip: "127.0.0.1",
    socket: { remoteAddress: "127.0.0.1" },
    headers: { "user-agent": "test-agent" },
  } as any;

  const mockResponse = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
    clearCookie: jest.fn(),
  } as any;

  mockRefreshTokenRepository.deleteByToken.mockResolvedValue(undefined);
  mockAuditService.logLogout.mockResolvedValue(undefined);

  await authController.logout(mockRequest, mockResponse);

  expect(mockResponse.clearCookie).toHaveBeenCalledWith("refreshToken");
  expect(mockResponse.status).toHaveBeenCalledWith(200);
});
```

Listing 3: Prueba de logout exitoso

6.1.4 Resultados de AuthController.test.ts

Cuadro 2: Resultados de pruebas AuthController

Caso de Prueba	Estado	Tiempo
should authenticate a user successfully	PASÓ	2ms
should return 401 for invalid credentials	PASÓ	5ms
should return 400 for invalid email format	PASÓ	3ms
should register a new user successfully	PASÓ	1ms
should return 400 for missing required fields	PASÓ	2ms
should logout user successfully	PASÓ	2ms
should return 400 when user not authenticated	PASÓ	2ms
should send recovery email for existing user	PASÓ	8ms
should return success for non-existing user (security)	PASÓ	2ms
should handle user role correctly	PASÓ	1ms
should have all expected roles	PASÓ	1ms
TOTAL	**11/11**	**4.174s**

6.2 CreditService.test.ts

Este archivo contiene las pruebas del servicio de créditos, validando todas las operaciones CRUD.

6.2.1 Estructura del Test

```
import { beforeEach, describe, it, expect, jest } from "@jest/globals";
import { CreditService } from "../../src/services/CreditService";
import { ICreditRepository } from "../../src/lib/interfaces/
   creditRepository";
import { Credit, CreditStatus } from "../../src/models/Credit";

describe("CreditService", () => {
  let creditService: CreditService;
  let mockCreditRepository: jest.Mocked<ICreditRepository>;

  beforeEach(() => {
    mockCreditRepository = {
      findByUserId: jest.fn(),
      findAll: jest.fn(),
      findById: jest.fn(),
      save: jest.fn(),
      update: jest.fn(),
      delete: jest.fn(),
    };
```

```
    creditService = new CreditService(mockCreditRepository);
  });
});
```

Listing 4: CreditService.test.ts - Configuración

6.2.2 Casos de Prueba - Creación de Créditos

```
it("should create a new credit with default values", async () => {
  mockCreditRepository.save.mockResolvedValue();

  const result = await creditService.createCredit({
    userId: "user1",
    amount: 1000,
  });

  expect(mockCreditRepository.save).toHaveBeenCalled();
  expect(result.userId).toBe("user1");
  expect(result.amount).toBe(1000);
  expect(result.status).toBe(CreditStatus.ACTIVE);
  expect(result.term).toBe(12);
  expect(result.interestRate).toBe(12);
});
```

Listing 5: Prueba de creación de crédito con valores por defecto

6.2.3 Casos de Prueba - Actualización de Créditos

```
it("should throw an error when updating a non-existent credit", async ()
    => {
  mockCreditRepository.findById.mockResolvedValue(null);

  await expect(
    creditService.updateCredit("invalid-id", { amount: 500 })
  ).rejects.toThrow("Credito no encontrado");
});
```

Listing 6: Prueba de error al actualizar crédito inexistente

6.2.4 Resultados de CreditService.test.ts

Cuadro 3: Resultados de pruebas CreditService

Caso de Prueba	Estado	Tiempo
should fetch credits by user ID	PASÓ	8ms
should return empty array when user has no credits	PASÓ	1ms
should return all credits	PASÓ	2ms
should return credit when found	PASÓ	1ms
should return null when credit not found	PASÓ	1ms
should create a new credit with default values	PASÓ	2ms
should create a new credit with custom values	PASÓ	1ms
should generate a unique ID for new credit	PASÓ	2ms
should update an existing credit	PASÓ	2ms
should update credit status	PASÓ	1ms
should throw error when updating non-existent credit	PASÓ	17ms
should update the updatedAt timestamp	PASÓ	1ms
should delete a credit	PASÓ	1ms
should create Credit instance with all properties	PASÓ	1ms
should have all expected CreditStatus values	PASÓ	1ms
TOTAL	**15/15**	**3.499s**

7 Pruebas de Carga Implementadas (K6)

7.1 auth.test.js - Prueba de Autenticación

7.1.1 Configuración del Test

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  vus: 10,
  duration: '10s',
  thresholds: {
    http_req_duration: ['p(95)<500'],
    http_req_failed: ['rate<0.01'],
  },
};

const BASE_URL = __ENV.BASE_URL || 'http://host.docker.internal:4000';

export default function () {
```

```
  const loginUrl = `${BASE_URL}/api/v1/auth/login`;

  const payload = JSON.stringify({
    email: 'test@example.com',
    password: 'TestPassword123!',
  });

  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  const res = http.post(loginUrl, payload, params);

  check(res, {
    'status is 200 or 401': (r) => r.status === 200 || r.status === 401,
    'response has success field': () => res.json().success !== undefined
    ,
  });

  sleep(1);
}
```

Listing 7: auth.test.js - Configuración K6

7.1.2 Resultados de auth.test.js

Cuadro 4: Métricas de rendimiento - Autenticación

Métrica	Valor
Usuarios Virtuales (VUs)	10
Duración	10 segundos
Total de Iteraciones	100
Requests por segundo	9.78 req/s
Duración Promedio	**35.5ms**
Duración Mínima	6.65ms
Duración Mediana	17.64ms
Duración Máxima	180.83ms
P90	**68.29ms**
P95	**173.07ms**
Datos Recibidos	72 KB (7.0 KB/s)
Datos Enviados	21 KB (2.0 KB/s)

Cuadro 5: Validación de Thresholds - Autenticación

Threshold	Criterio	Resultado
http_req_duration	p(95) ¡500ms	PASÓ (173.07ms)

Cuadro 6: Checks de validación - Autenticación

Check	Resultado
status is 200 or 401	100 % (100/100)
response has success field	100 % (100/100)
response has accessToken on success	100 % (100/100)
response has user on success	100 % (100/100)

7.2 credits.test.js - Prueba de Créditos

7.2.1 Configuración del Test

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  vus: 10,
  duration: '30s',
  thresholds: {
    http_req_duration: ['p(95)<1000'],
    http_req_failed: ['rate<0.05'],
  },
};

const BASE_URL = __ENV.BASE_URL || 'http://host.docker.internal:4000';

export default function runCreditsTest() {
  // Primero autenticarse
  const loginUrl = `${BASE_URL}/api/v1/auth/login`;

  const loginPayload = JSON.stringify({
    email: 'test@example.com',
    password: 'TestPassword123!',
  });

  const loginRes = http.post(loginUrl, loginPayload, {
    headers: { 'Content-Type': 'application/json' },
  });

  if (loginRes.status !== 200) {
```

```
    console.log('Login failed, skipping credits test');
    sleep(1);
    return;
  }

  const token = loginRes.json().accessToken;

  // Obtener creditos del usuario
  const creditsUrl = `${BASE_URL}/api/v1/credits/my`;
  const creditsRes = http.get(creditsUrl, {
    headers: { Authorization: `Bearer ${token}` },
  });

  check(creditsRes, {
    'credits status is 200': (r) => r.status === 200,
  });

  sleep(1);
}
```

Listing 8: credits.test.js - Configuración K6

7.2.2 Resultados de credits.test.js

Cuadro 7: Métricas de rendimiento - Créditos

Métrica	Valor
Usuarios Virtuales (VUs)	10
Duración	30 segundos
Total de Iteraciones	300
Requests por segundo	9.99 req/s
Duración Promedio	**16.87ms**
Duración Mínima	0.15ms
Duración Mediana	10.46ms
Duración Máxima	115.19ms
P90	**29.03ms**
P95	**51.67ms**
Datos Recibidos	215 KB (7.1 KB/s)
Datos Enviados	63 KB (2.1 KB/s)

Cuadro 8: Validación de Thresholds - Créditos

Threshold	Criterio	Resultado
http_req_duration	p(95) ¡1000ms	PASÓ (51.67ms)

8 Resumen de Resultados

Cuadro 9: Resumen ejecutivo de pruebas

Tipo de Prueba	Total	Pasados	Fallidos	Tasa
Jest - AuthController	11	11	0	100 %
Jest - CreditService	15	15	0	100 %
K6 - Auth (iteraciones)	100	100	0	100 %
K6 - Credits (iteraciones)	300	300	0	100 %
TOTAL	**426**	**426**	**0**	**100 %**

9 Métricas de Calidad

9.1 Cobertura Funcional

Cuadro 10: Cobertura funcional por módulo

Módulo	Funciones Probadas	Cobertura
AuthController	login, register, logout, recoverPassword	100 %
CreditService	getCreditsByUserId, listCredits, getCreditById, createCredit, updateCredit, deleteCredit	100 %

9.2 Tiempos de Respuesta (SLA)

Cuadro 11: Cumplimiento de SLA de tiempos de respuesta

Endpoint	P95 Actual	SLA	Estado
POST /api/v1/auth/login	173ms	¡500ms	Cumple
GET /api/v1/credits	51ms	¡1000ms	Cumple

10 Comandos de Ejecución

10.1 Pruebas Unitarias (Jest)

```
# Navegar al directorio del backend
cd pagoseguro-backend

# Ejecutar todas las pruebas
npm run test

# Ejecutar pruebas de AuthController
npm run test AuthController.test.ts
```

```
# Ejecutar pruebas de CreditService
npm run test CreditService.test.ts
```

Listing 9: Comandos para ejecutar pruebas Jest

10.2 Pruebas de Carga (K6 con Docker)

```
# Navegar al directorio de pruebas K6
cd pagoseguro-backend/tests/k6

# Ejecutar prueba de autenticacion
docker run --rm -i grafana/k6 run \
  -e BASE_URL=http://host.docker.internal:4000 \
  - < auth.test.js

# Ejecutar prueba de creditos
docker run --rm -i grafana/k6 run \
  -e BASE_URL=http://host.docker.internal:4000 \
  - < credits.test.js

# Ejecutar con mas usuarios virtuales
docker run --rm -i grafana/k6 run \
  --vus 50 --duration 30s \
  -e BASE_URL=http://host.docker.internal:4000 \
  - < auth.test.js
```

Listing 10: Comandos para ejecutar pruebas K6

11 Conclusiones

- La implementación de pruebas automatizadas con **Jest** permitió validar de manera sistemática la lógica interna del backend, alcanzando una tasa de éxito del **100 %** en los 26 casos de prueba ejecutados.
- Las pruebas de carga realizadas con **K6** demostraron que el sistema mantiene tiempos de respuesta excelentes bajo carga concurrente, con un P95 de **173ms** para autenticación y **51ms** para consulta de créditos, muy por debajo de los límites establecidos.
- La arquitectura del backend, basada en principios **SOLID** y patrones de inyección de dependencias, facilita el testing mediante mocks, permitiendo aislar cada componente de forma efectiva.

- El sistema demuestra un comportamiento seguro al no revelar información sobre la existencia de usuarios en los endpoints de recuperación de contraseña, siguiendo las mejores prácticas de seguridad.

12 Recomendaciones

- **Crear usuario de prueba en base de datos:** Para ejecutar pruebas K6 completas que incluyan el flujo de créditos autenticados, se recomienda crear un usuario de prueba persistente.
- **Aumentar VUs para pruebas de estrés:** Se recomienda ejecutar pruebas con 50-100 usuarios virtuales para evaluar el comportamiento bajo alta carga.
- **Implementar pruebas de integración:** Agregar pruebas E2E que validen flujos completos desde el frontend hasta la base de datos.
- **Integrar en pipeline CI/CD:** Automatizar la ejecución de pruebas en cada push al repositorio mediante GitHub Actions o similar.
- **Monitoreo continuo:** Implementar dashboards de Grafana para visualizar métricas de rendimiento en tiempo real.