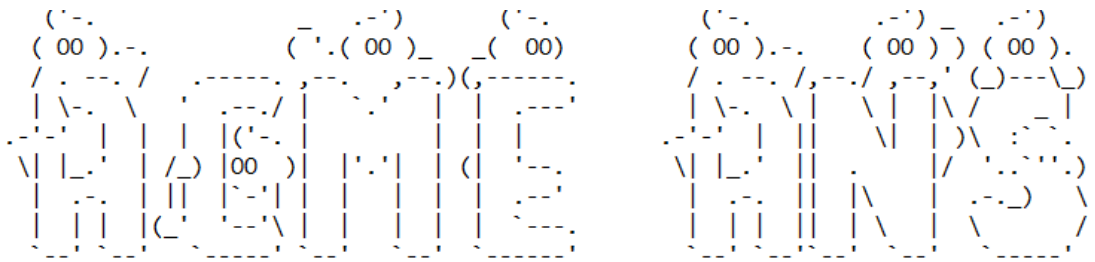
REPORTE DE TESTING

Acme-ANS-D04



Repositorio: https://github.com/FranciscoFernandezN/Acme-ANS

Creado por el grupo C1.022, del G1

Participantes	
Nombres	Correos
Varo Vera, Juan	juavarver@alum.us.es

_

Índice

Resumen ejecutivo

Este es el proyecto del grupo C1.022 sobre Acme AirNav Solutions, S.A. (Acme ANS, S.A. abreviado), la cual es una compañía ficticia especializada en ayudar a aeropuertos a organizar y coordinar sus operaciones a partir de soluciones desarrolladas en software. La logística de los vuelos (la programación de los vuelos, la organización de reservas y de tripulación, etc.) se gestionan mediante el desarrollo de un WIS.

Con esto, para el correcto funcionamiento de la aplicación, se deberá escribir un reporte de testing donde se describan los valores de cobertura de código junto con los valores correspondientes al rendimiento de la aplicación.

Tabla de revisiones

Número	Fecha	Descripción
V1.0.0	25/05/2025	Versión inicial terminada del reporte

Introducción:

En este reporte quedarán expuestos los resultados de los tests realizados durante el entregable D04 sobre las funcionalidades desarrolladas por el estudiante 3, Juan Varo Vera, sobre los requisitos 8 y 9.

Se reportarán 2 tipos de resultados con el testeo funcional y los relacionados con el testeo del rendimiento.

Este reporte está organizado de la siguiente forma:

1. **Resumen ejecutivo**: Introducción breve sobre el reporte.
2. **Tabla de revisiones**: Historial de revisiones del documento.
3. **Introducción**: Contextualización de la planificación y progreso además de su importancia.
4. **Funcional testing**: Resultados obtenidos de la fase de testing y posibles bugs encontrados.
5. **Performance testing**: Resultados obtenidos de analizar el rendimiento de los tests en 2 ordenadores.
6. **Conclusiones**: Resumen de los hallazgos y la importancia de este reporte.
7. **Bibliografía**: Fuentes consultadas durante la investigación.

Functional testing

- Resultados sobre el testeo en las asignaciones de vuelo:

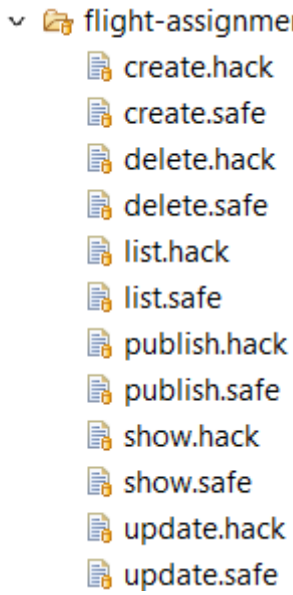


Figura 1: Archivos de testing para los flight assignments.

Se reportaron varios bugs durante la fase de testing. El más importante fue al realizar distintas pruebas de hacking, que permitían cambiar valores usando f12, y su posterior actualización, lo que resultaba en un panic y una brecha de seguridad. También fueron detectadas validaciones innecesarias y otras necesarias.

acme.features.flightCrewMembe	94,7 %	2.240	125	2.365
FlightCrewMemberFlightAssig	91,1 %	530	52	582
FlightCrewMemberFlightAssig	92,6 %	566	45	611
FlightCrewMemberFlightAssig	95,1 %	469	24	493
FlightCrewMemberFlightAssig	99,2 %	261	2	263
FlightCrewMemberFlightAssig	99,0 %	207	2	209
FlightCrewMemberFlighAssig	100,0 %	42	0	42
FlightCrewMemberFlightAssig	100,0 %	80	0	80
FlightCrewMemberFlightAssig	100,0 %	85	0	85

Figura 2: Cobertura de los tests en los flight assignments.

Se ha logrado alcanzar una cobertura de un 94,7%, cubriendo la gran mayoría de las líneas de código, pero hay alguna rama condicional que no se ejecuta debido a que posiblemente falten algunos datos de ejemplo más para el testeo.

```
// AbstractGuiService interface ----------------------------------------
@Override
public void authorise() {
    int id = super.getRequest().getPrincipal().getRealmOfType(FlightCrewMember.class).getId();
    Boolean isAvailable = this.repository.findFlightCrewMemberById(id).getAvailabilityStatus().equals(AvailabilityStatus.AVAILABLE);
    super.getResponse().setAuthorised(super.getRequest().getPrincipal().hasRealmOfType(FlightCrewMember.class) && isAvailable);
}
```

Figura 3: Condición sin cubrir al completo.

- Resultado sobre el testeo en los activity logs:

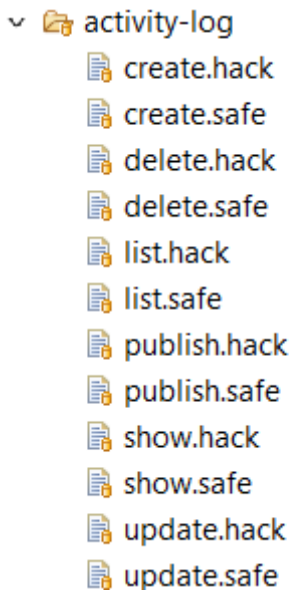


Figura 4: Archivos de testing para los tracking logs.

Fueron detectados varios bugs durante la fase de testing, El más importante fue que se podían modificar campos con f12, de forma que se podía hacking. El resto fueron problemas de validaciones.

Se ha logrado alcanzar una cobertura del 96,2% en la feature. No se ha alcanzado más por la posible falta de algunos datos de ejemplo más para el testeo.

acme.features.flightCrewMembe	96,2 %	1.278	51	1.329
FlightCrewMemberActivityLo	91,3 %	251	24	275
FlightCrewMemberActivityLo	95,6 %	260	12	272
FlightCrewMemberActivityLo	96,5 %	218	8	226
FlightCrewMemberActivityLo	98,1 %	258	5	263
FlightCrewMemberActivityLo	98,9 %	173	2	175
FlightCrewMemberActivityLo	100,0 %	35	0	35
FlightCrewMemberActivityLo	100,0 %	83	0	83

Figura 5: Cobertura de los tests en los activity logs

```java
@Override
public void authorise() {
    int activityLogId = super.getRequest().getData("id", int.class);
    ActivityLog activityLog = this.repository.findActivityLogById(activityLogId);

    boolean authorised = false;

    if (activityLog != null && activityLog.getFlightAssignment() != null) {
        FlightAssignment assignment = activityLog.getFlightAssignment();
        int userId = super.getRequest().getPrincipal().getActiveRealm().getId();

        boolean isOwner = assignment.getFlightCrewMember().getId() == userId;
        boolean isDraft = Boolean.TRUE.equals(activityLog.getIsDraftMode());

        Leg leg = assignment.getLeg();
        Date now = MomentHelper.getCurrentMoment();
        boolean legHasOccurred = leg != null && leg.getScheduledDeparture() != null && leg.getScheduledDeparture().befor

        authorised = isOwner && isDraft && legHasOccurred;
    }
```

Figura 6: Condición sin cubrir al completo en el delete.

Functional testing

- Resultados en ordenador 1 con índices:

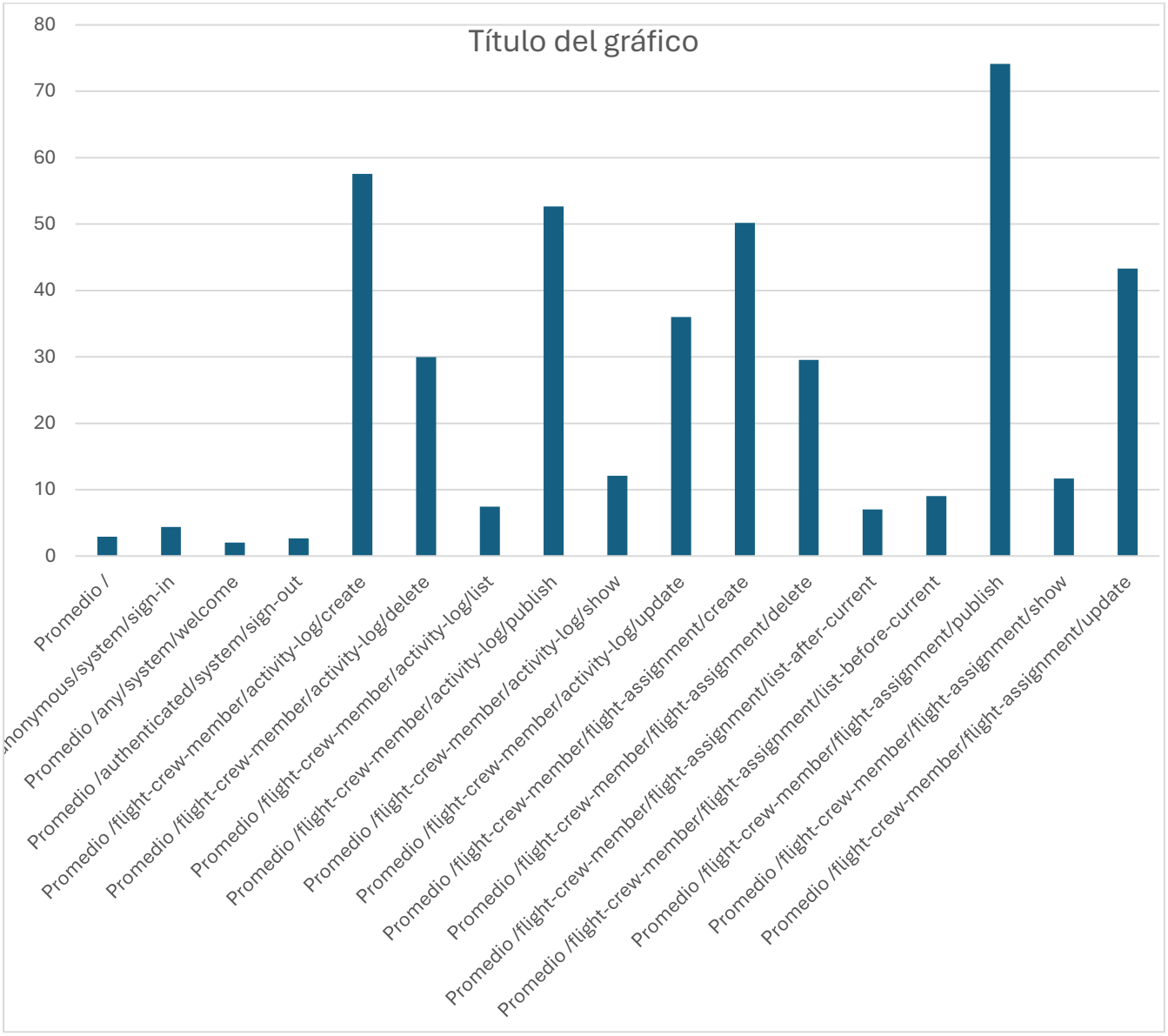


Figura 1: Tiempos para Ordenador 1 con índices

En esta tabla podemos observar los tiempos promedios de las distintas peticiones realizadas tanto en los flight assignments como en los activity logs con los índices calculados por el ordenador 1.

Es notable que las peticiones que más han tardado son las relacionadas con la creación, la publicación y la actualización, esto se debe a la gran cantidad de datos que deben de manejar en las numerosas peticiones que se realizan. El listado y el resto de peticiones no deben trabajar mucho debido a que no contamos con la suficiente cantidad de datos como para que el sistema se tenga que esforzar en gran medida.

- Resultados en ordenador 1 sin índices:

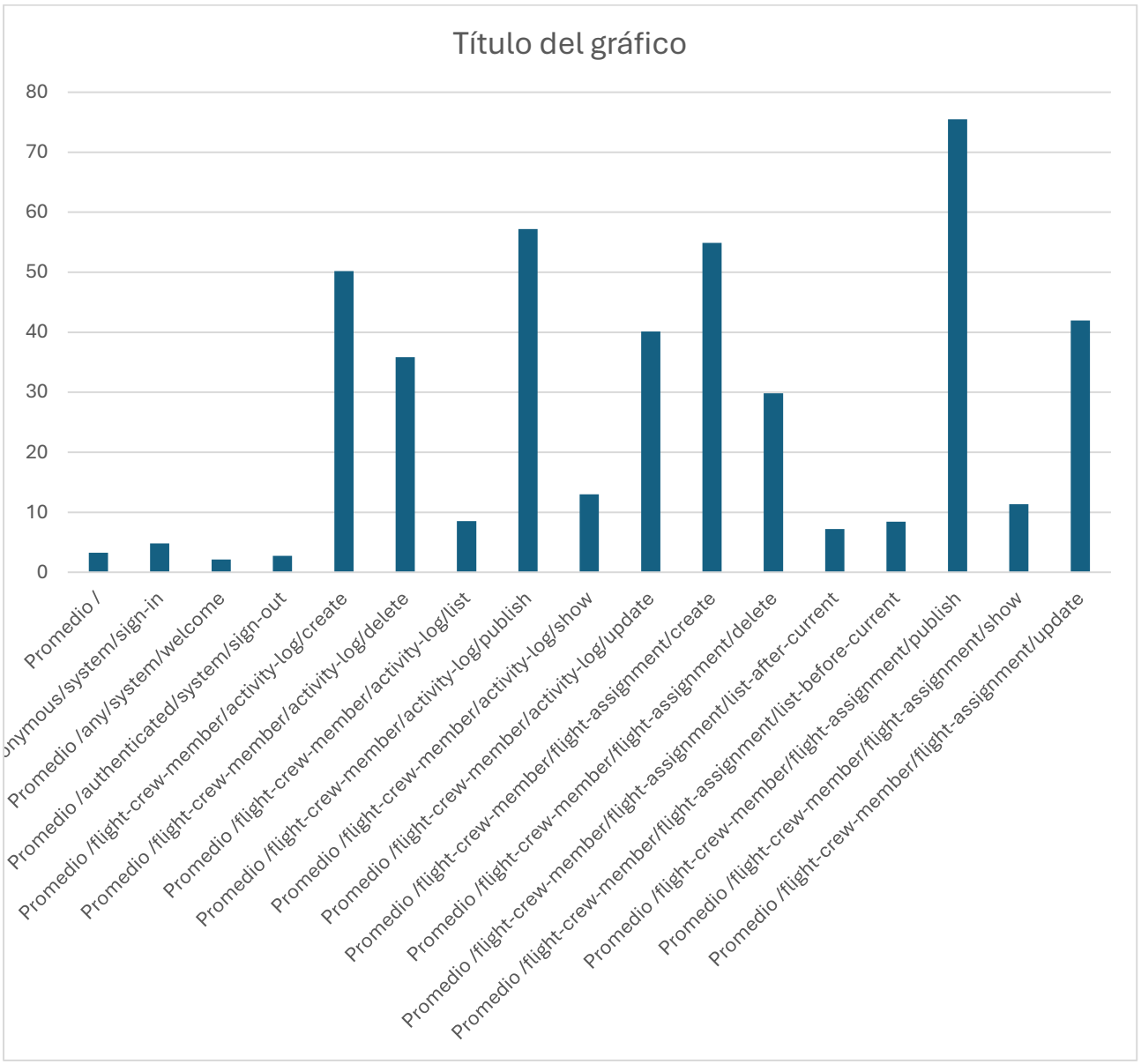


Figura 2: Tiempos para Ordenador 1 sin índices

Como se puede observar, los cambios entre usar o no índices no son muy claros, y esto es debido a que para que se refleje el rendimiento de los índices la cantidad de datos en la base de datos debe ser bastante más grande que los datos que actualmente contiene.

Con índices		*Sin índices*	
Media	10,6583294	Media	10,953129
Error típico	0,63047683	Error típico	0,63499955
Mediana	5,075599	Mediana	5,2451
Moda	1,6272	Moda	2,1685
Desviación es	18,0871046	Desviación es	18,2168522
Varianza de la	327,143352	Varianza de la	331,853706
Curtosis	16,9565692	Curtosis	12,6053145
Coeficiente de	3,73487622	Coeficiente de	3,37204426
Rango	158,7444	Rango	141,756
Mínimo	1,0387	Mínimo	1,173
Máximo	159,7831	Máximo	142,929
Suma	8771,80507	Suma	9014,4252
Cuenta	823	Cuenta	823
Nivel de confia	1,23753406	Nivel de confia	1,2464115

	Sin índices	*Con índices*
Media	10,953129	10,6583294
Varianza (con	340339868	327143352
Observacione	823	823
Diferencia hip	0	
z	0,00032735	
P(Z<=z) una co	0,49986941	
Valor crítico d	1,64485363	
Valor crítico d	0,99973882	
Valor crítico d	1,95996398	

Figuras 3,4 y 5: Valores estadísticos entre Ordenador 1 con y sin índices

La tabla nos aporta que a pesar de que los índices mejoran el rendimiento, si lo comparamos sin ellos parece que los tiempos mejoran en este caso.

- Resultado del ordenador 2:

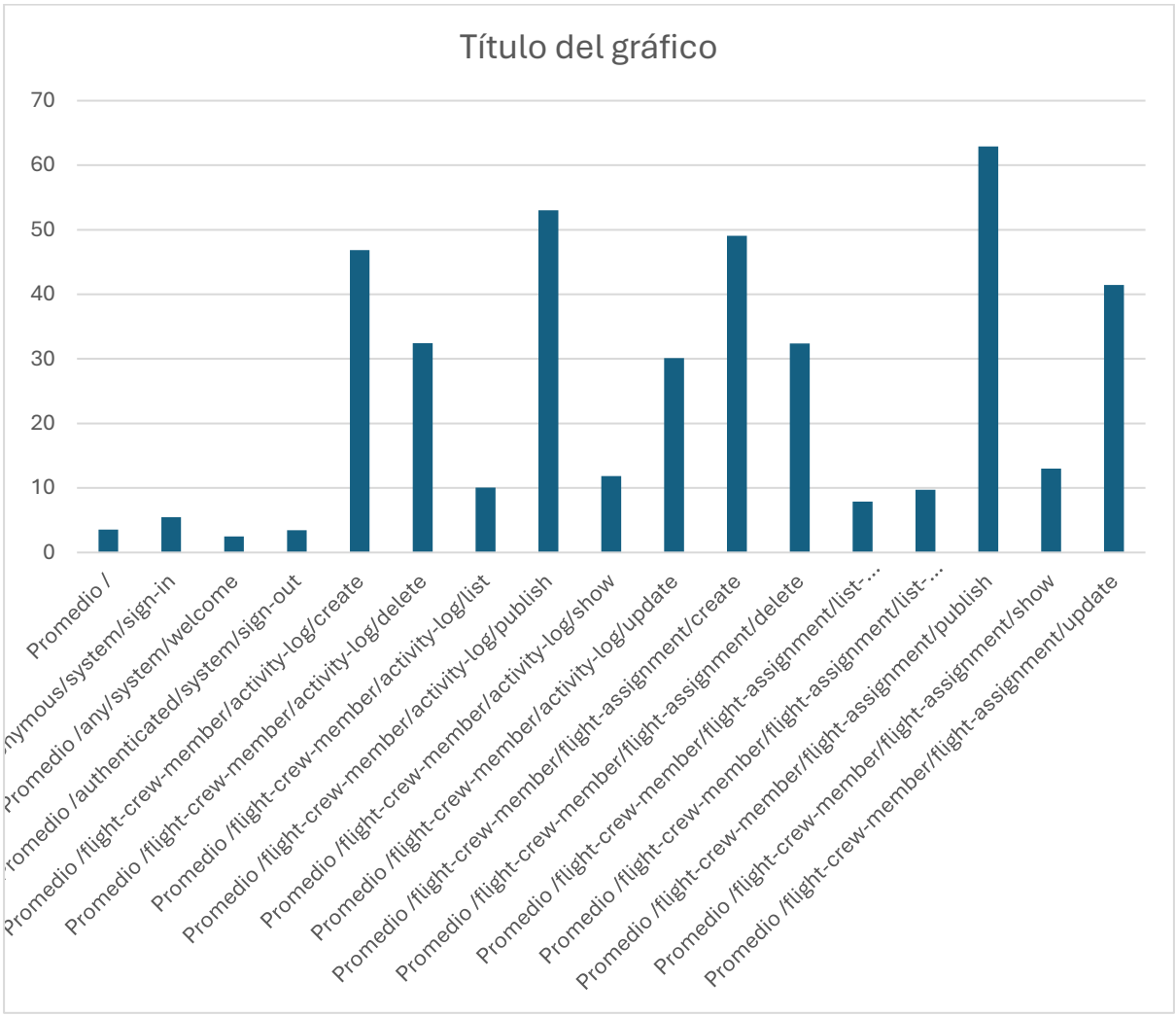


Figura 6: Tiempos para Ordenador 2 con índices

En esta tabla podemos observar los tiempos promedios de las distintas peticiones realizadas tanto en los flight assignments como en los activity logs con los índices calculados por el ordenador 2.

Estos resultados fueron obtenidos con un ordenador de similar potencia al ordenador 1, pero podemos ver que las proporciones se mantienen y create, publish y update destacan por encima del resto.

Ordenador 1		*Ordenador 2*	
Media	10,6583294	Media	11,0429018
Error típico	0,63047683	Error típico	0,5490093
Mediana	5,075599	Mediana	5,9898
Moda	1,6272	Moda	3,1315
Desviación es	18,0871046	Desviación es	15,7499659
Varianza de la	327,143352	Varianza de la	248,061427
Curtosis	16,9565692	Curtosis	9,83288738
Coeficiente de	3,73487622	Coeficiente de	3,00417693
Rango	158,7444	Rango	105,0658
Mínimo	1,0387	Mínimo	1,3862
Máximo	159,7831	Máximo	106,452
Suma	8771,80507	Suma	9088,3082
Cuenta	823	Cuenta	823
Nivel de confia	1,23753406	Nivel de confia	1,07762518

Prueba z para medias de dos muestras		
	Variable 1	*Variable 2*
Media	735,680165	744,475107
Varianza (con	327143352	248061427
Observacione	14	14
Diferencia hip	0	
z	-0,0013721	
P(Z<=z) una co	0,49945261	
Valor crítico d	1,64485363	
Valor crítico d	0,99890522	
Valor crítico d	1,95996398	

Figuras 7, 8 y 9: Valores estadísticos entre Ordenador 1 y Ordenador 2 con índices

Al haberse realizado la comparación con un ordenador ligeramente menos potente podemos ver que los tiempos han empeorado, pero esto es algo obvio debido a la diferencia de hardware, ya que el número de instrucciones ejecutadas es el mismo.

Conclusiones

Como conclusión de este reporte decir que gracias a los tests se han encontrado varias vulnerabilidades en el código y ciertas validaciones que no estaban haciendo lo correcto, con lo cuál la fase de testing ha sido un éxito y ha realizado su función de manera plena. También se ha podido observar que puntos de las operaciones realizadas por el sistema son los más costosos en cuanto a tiempo y recursos gracias al análisis de los datos.

Bibliografía

Intencionalmente en blanco.