Pruebas de Software Servicio Backend SmishGuard

1. Introducción

1.1 Propósito

Este documento tiene como objetivo documentar las pruebas realizadas para el servicio Flask de **SmishGuard**, que permite realizar consultas a modelos de análisis, almacenar y gestionar mensajes en una base de datos **MongoDB** y consumir varios microservicios para detección de spam y amenazas. Este sistema también permite la gestión de comentarios de soporte, historial de análisis y números bloqueados, y provee estadísticas de mensajes y su estado de publicación.

1.2 Alcance de las Pruebas

El documento cubre los siguientes tipos de pruebas:

1. **Pruebas Unitarias**: Se verifican individualmente las funcionalidades de cada ruta y sus métodos asociados.
2. **Pruebas de Integración**: Se evalúa la interacción entre distintos módulos y la correcta colaboración entre microservicios externos y MongoDB.
3. **Pruebas de API**: Se prueban los endpoints de la API para verificar su correcto funcionamiento y las respuestas esperadas.
4. **Pruebas de Carga**: Se somete a la API a condiciones de alta carga para medir su rendimiento, capacidad de respuesta y estabilidad bajo distintos niveles de tráfico.

1.3 Objetivos

Los objetivos específicos de las pruebas incluyen:

- Asegurar que cada endpoint responda de acuerdo a las especificaciones funcionales.
- Verificar que el sistema maneje adecuadamente los errores y tiempos de espera en los microservicios externos.
- Evaluar la eficiencia de la base de datos MongoDB al manejar múltiples peticiones concurrentes.
- Validar el rendimiento del sistema y su capacidad para manejar grandes volúmenes de datos sin comprometer su estabilidad.

2. Metodología de Pruebas

2.1 Tipos de Pruebas

2.1.1 Pruebas Unitarias

- **Descripción**: Prueban individualmente las funcionalidades de cada ruta y endpoint, verificando respuestas HTTP, mensajes y estatus.
- **Objetivo**: Validar el comportamiento correcto de cada función o método de manera aislada.
- **Alcance**: Incluyen pruebas de los endpoints principales (`/`, `/ping`, `/consultar-modelo`, entre otros) y las funcionalidades de guardar, actualizar y eliminar mensajes.

2.1.2 Pruebas de Integración

- **Descripción**: Verifican la interacción entre diferentes módulos y microservicios externos, y el almacenamiento en MongoDB.
- **Objetivo**: Asegurar que las dependencias externas y la base de datos interactúan correctamente con la aplicación y que el flujo de datos entre ellos es exitoso.
- **Alcance**: Se simulan flujos completos como el guardado y consulta de mensajes, eliminación de datos en la base de datos y la gestión de comentarios y números bloqueados.

2.1.3 Pruebas de API

- **Descripción**: Valida el correcto funcionamiento de los endpoints expuestos, incluyendo la verificación de respuestas, estructura de JSON y contenido de las respuestas.
- **Objetivo**: Asegurar que cada endpoint responde según lo esperado, con respuestas en el formato correcto y que se devuelven los datos esperados.
- **Alcance**: Se probaron endpoints como `/ping`, `/guardar-mensaje-para-publicar`, `/eliminar-mensaje-para-publicar`, `/mensaje-aleatorio` y `/estadisticas`.

2.1.4 Pruebas de Carga

- **Descripción**: Prueban la aplicación bajo condiciones de carga utilizando **Locust** para simular múltiples usuarios accediendo al servicio simultáneamente.
- **Objetivo**: Determinar la capacidad del sistema para manejar un alto volumen de solicitudes, midiendo tiempos de respuesta, latencia y estabilidad.
- **Alcance**: Se testearon varios endpoints con diferentes frecuencias, incluyendo `/ping`, `/consultar-modelo`, `/guardar-mensaje-para-publicar`, `/mensajes-para-publicar` y `/mensaje-aleatorio`.

2.2 Herramientas de Prueba

- **pytest**: Utilizado para la ejecución de pruebas unitarias e integraciones, y validar la funcionalidad de los endpoints de la API.
- **mongomock**: Simula MongoDB en memoria para aislar las pruebas de la base de datos real.
- **unittest.mock**: Utilizado para simular y controlar la conexión a MongoDB.
- **Locust**: Herramienta de pruebas de carga para simular múltiples usuarios concurrentes interactuando con el servicio.

2.3 Criterios de Aceptación

2.3.1 Pruebas Unitarias

- Los endpoints deben devolver el código de estado HTTP correcto.
- Los mensajes de respuesta deben coincidir con los valores esperados según la funcionalidad de cada endpoint.

2.3.2 Pruebas de Integración

- Los datos deben almacenarse y recuperarse correctamente en MongoDB.
- Los servicios deben responder dentro del tiempo de espera especificado, y los flujos de datos entre la aplicación y los microservicios deben completarse sin errores.

2.3.3 Pruebas de API

- Cada endpoint debe responder con los datos esperados en el formato JSON correcto.
- Las solicitudes HTTP deben recibir respuestas dentro de tiempos razonables (menos de 2 segundos para la mayoría de los endpoints).

2.3.4 Pruebas de Carga

- La aplicación debe manejar hasta 100 usuarios concurrentes sin errores.
- Los tiempos de respuesta deben mantenerse estables, con picos de latencia permitidos solo en períodos breves.

3. Pruebas Unitarias

3.1 Propósito

El propósito de las pruebas unitarias es validar que cada una de las funcionalidades individuales de los endpoints de la API y otros métodos críticos del sistema funcionen correctamente de manera aislada. Estas pruebas aseguran que los componentes básicos operen según lo esperado antes de integrarlos y probar su interacción con otros sistemas o servicios externos.

3.2 Herramientas

Para las pruebas unitarias se utilizó:

- **pytest**: Framework de pruebas en Python que permite la ejecución de pruebas unitarias y ofrece un reporte detallado de los resultados.
- **mongomock**: Librería para simular una base de datos MongoDB en memoria, lo cual permite aislar las pruebas de la base de datos real.
- **unittest.mock**: Utilizado para simular la conexión a MongoDB y facilitar la creación de pruebas aisladas.

3.3 Estrategia de Pruebas

Las pruebas unitarias cubrieron los principales endpoints y funcionalidades, tales como la validación de respuestas correctas y la gestión de errores. Cada prueba se ejecutó de manera independiente, comprobando las salidas esperadas y el manejo de errores en cada ruta.

3.4 Casos de Prueba

Los casos de prueba incluyen, entre otros:

1. **`test_empty_text_error`**: Verifica el manejo de errores cuando el texto está vacío.
2. **`test_home_route`**: Comprueba la accesibilidad de la ruta principal.
3. **`test_invalid_json_format`**: Valida el manejo de solicitudes con un formato JSON incorrecto.
4. **`test_missing_text_key_error`**: Prueba el caso en que falta la clave `texto` en el JSON.
5. **`test_model_not_loaded_error`**: Valida el comportamiento cuando el modelo no está cargado.
6. **`test_predict_ham_message`**: Verifica la respuesta del sistema cuando se envía un mensaje no sospechoso.
7. **`test_predict_spam_message`**: Prueba la respuesta cuando se envía un mensaje identificado como spam.
8. **`test_translation_error_handling`**: Comprueba el manejo de errores en la traducción.

3.5 Resultados de Ejecución

- **Número de pruebas ejecutadas**: 15
- **Pruebas exitosas**: 15 (100%)
- **Tiempo de ejecución total**: 5.12 segundos

Cada una de las pruebas fue exitosa, como lo indica el reporte de `pytest` con la etiqueta "**15 passed**" y un progreso de ejecución del 100%. Esto sugiere que todas las funcionalidades probadas están operativas y cumplen con los requisitos definidos para cada caso de prueba.

3.6 Observaciones

- **Advertencia**: Se presentó una advertencia de `PytestDeprecationWarning` relacionada con la configuración de `asyncio_default_fixture_loop_scope`. Esta advertencia no afecta los resultados de las pruebas actuales, pero podría considerarse la actualización o revisión de la configuración en futuras versiones de `pytest`.

4. Pruebas de Carga

4.1 Propósito

La prueba de carga tiene como objetivo evaluar el rendimiento del sistema bajo un alto número de solicitudes concurrentes y determinar si los tiempos de respuesta y la estabilidad se mantienen dentro de los límites aceptables. Este análisis es esencial para identificar cualquier problema de rendimiento que pueda surgir en condiciones de tráfico elevado.

4.2 Herramientas

Para la prueba de carga se utilizó:

- **Locust**: Herramienta de pruebas de carga que permite simular múltiples usuarios accediendo a los endpoints del sistema de manera concurrente. Locust proporciona métricas detalladas sobre la cantidad de solicitudes por segundo, tiempos de respuesta y tasas de error.

4.3 Estrategia de Pruebas

La prueba de carga se diseñó para evaluar los principales endpoints del sistema con un número creciente de usuarios. Se definieron las siguientes tareas para simular el comportamiento de los usuarios:

1. **`ping`**: Endpoint básico para verificar la disponibilidad.
2. **`consultar_modelo`**: Solicita el análisis de un mensaje.
3. **`guardar_mensaje_para_publicar`**: Guarda un mensaje nuevo en el sistema.
4. **`obtener_mensajes_para_publicar`**: Consulta los mensajes en cola para ser publicados.
5. **`mensaje_aleatorio`**: Obtiene un mensaje aleatorio del sistema.
6. **`obtener_estadisticas`**: Consulta estadísticas generales del sistema.

Cada tarea tiene una ponderación asignada, lo que permite simular una carga más realista al darle prioridad a ciertas operaciones.

4.4 Resultados de Ejecución

- **Total de Solicitudes por Segundo**: El sistema mantuvo un promedio de aproximadamente 50 solicitudes por segundo con una tasa de fallos nula, indicando que el sistema puede manejar la carga solicitada sin errores.
- **Tiempos de Respuesta**:
    - **Percentil 50 (mediana)**: Se observó una respuesta estable alrededor de los **200-300 ms**.
    - **Percentil 95**: Al inicio, algunos tiempos alcanzaron picos de hasta **1200 ms**, pero se estabilizaron cerca de los **700 ms**, lo cual indica un ajuste en el manejo de carga.
- **Número de Usuarios Concurrentes**: La prueba se ejecutó inicialmente con **100 usuarios concurrentes**, manteniendo este valor antes de la desaceleración progresiva hasta 0. Esto permitió evaluar el sistema tanto en condiciones de carga constante como en reducción.

4.5 Observaciones

- **Estabilidad**: No se presentaron errores de sistema bajo una carga sostenida de 100 usuarios concurrentes, lo cual es un buen indicador de estabilidad.
- **Optimización de Respuesta**: Los tiempos de respuesta iniciales más altos en el percentil 95 sugieren que podría considerarse una optimización en el manejo de las solicitudes iniciales para reducir los picos de tiempo de respuesta.

4.6 Conclusión

El sistema demostró un buen rendimiento bajo carga con capacidad para manejar solicitudes concurrentes sin errores. No obstante, se recomienda monitorear y optimizar los tiempos de respuesta en los momentos de arranque o aumento súbito de carga.

5. Pruebas de API

5.1 Propósito

El propósito de las pruebas de API es validar que los endpoints del sistema funcionen correctamente, devuelvan las respuestas esperadas y manejen adecuadamente los errores. Estas pruebas se centran en verificar la integridad de los datos, los códigos de estado HTTP y la estructura de las respuestas JSON.

5.2 Herramientas

Para la ejecución de las pruebas de API se utilizó:

- **pytest**: Framework de pruebas para validar las respuestas de los endpoints.

- **mongomock**: Utilizado para simular la base de datos MongoDB en memoria, lo cual permite que las pruebas sean aisladas sin afectar la base de datos real.

5.3 Estrategia de Pruebas

Las pruebas de API se diseñaron para evaluar los endpoints principales del sistema. Se definieron casos de prueba para validar las siguientes operaciones:

1. **`/ping`**: Verifica la disponibilidad básica del sistema.
2. **`/guardar-mensaje-para-publicar`**: Prueba la funcionalidad para guardar un nuevo mensaje.
3. **`/actualizar-publicado`**: Valida la actualización del estado de publicación de un mensaje.
4. **`/eliminar-mensaje-para-publicar`**: Verifica la eliminación de un mensaje del sistema.
5. **`/mensaje-aleatorio`**: Obtiene un mensaje aleatorio y valida su contenido.
6. **`/estadisticas`**: Consulta las estadísticas de los mensajes en el sistema.

Cada prueba comprueba el código de estado HTTP, el contenido de la respuesta y la estructura de los datos devueltos para asegurarse de que los endpoints cumplen con los requisitos definidos.

5.4 Resultados de Ejecución

- **Número de pruebas ejecutadas**: 6
- **Pruebas exitosas**: 6 (100%)
- **Tiempo de ejecución total**: 4.64 segundos

Las pruebas fueron ejecutadas satisfactoriamente, con todas las verificaciones superadas y el mensaje "**6 passed in 4.64s**" en el reporte. Esto indica que cada endpoint probó correctamente su funcionalidad principal, incluyendo la creación, consulta, actualización y eliminación de datos, así como la obtención de estadísticas.

5.5 Casos de Prueba Detallados

Endpoint	Prueba	Descripción	Resultado
`/ping`	`test_ping`	Verifica que el endpoint de ping responda con un código de estado 200 y el mensaje esperado.	**Pasó**
`/guardar-mensaje-para-publicar`	`test_guardar_mensaje_para_publicar`	Prueba la funcionalidad de guardar un	**Pasó**

Endpoint	Prueba	Descripción	Resultado
		nuevo mensaje y verifica el mensaje de éxito en la respuesta.	
`/actualizar-publicado`	`test_actualizar_publicado`	Valida la actualización del estado de un mensaje a "publicado".	**Pasó**
`/eliminar-mensaje-para-publicar`	`test_eliminar_mensaje_para_publicar`	Verifica que el sistema permita eliminar un mensaje correctamente y confirme la eliminación.	**Pasó**
`/mensaje-aleatorio`	`test_mensaje_aleatorio`	Obtiene un mensaje aleatorio y confirma que el contenido devuelto coincide con el mensaje esperado.	**Pasó**
`/estadisticas`	`test_obtener_estadisticas`	Consulta estadísticas del sistema y valida los valores de mensajes seguros y peligrosos.	**Pasó**

5.6 Observaciones

- **Advertencia**: Similar a las pruebas anteriores, se presenta una advertencia `PytestDeprecationWarning` sobre la configuración `asyncio_default_fixture_loop_scope`. Esto no afecta los resultados actuales, pero se recomienda una revisión para futuras versiones de `pytest`.

5.7 Conclusión

Las pruebas de API confirmaron que los endpoints del sistema funcionan correctamente bajo las condiciones de prueba definidas. Todos los endpoints probados respondieron con los datos correctos y manejaron las solicitudes según lo esperado.

6. Pruebas de Integración

6.1 Propósito

Las pruebas de integración tienen como objetivo verificar la correcta interacción entre los diferentes módulos y componentes del sistema. En particular, se valida cómo los endpoints de la API interactúan con la base de datos simulada (MongoDB) y se asegura que el flujo de datos entre la aplicación y sus dependencias funcione correctamente.

6.2 Herramientas

Para la ejecución de las pruebas de integración se utilizaron:

- **pytest**: Para la definición y ejecución de los casos de prueba.
- **mongomock**: Para simular MongoDB en memoria, permitiendo aislar las pruebas y evitar alteraciones en la base de datos real.
- **unittest.mock**: Para simular conexiones y gestionar dependencias de manera controlada.

6.3 Estrategia de Pruebas

Se diseñaron pruebas para validar flujos completos en los endpoints del sistema. Cada prueba de integración se enfoca en una secuencia de operaciones, verificando que las interacciones con la base de datos se realicen correctamente y que los endpoints devuelvan los resultados esperados. Los principales flujos de prueba incluyen:

1. **Guardar y Consultar un Mensaje**: Verificar que un mensaje se pueda guardar y consultar exitosamente.
2. **Eliminar un Mensaje**: Validar que un mensaje puede ser eliminado de la base de datos.
3. **Consultar un Modelo**: Verificar el análisis de un mensaje existente.
4. **Gestión de Comentarios de Soporte**: Probar el almacenamiento y consulta de comentarios de soporte.
5. **Gestión de Números Bloqueados**: Validar el almacenamiento y consulta de números bloqueados por un usuario.
6. **Mensaje Aleatorio**: Comprobar que el sistema devuelve un mensaje aleatorio.

7. **Consulta de Estadísticas**: Validar la consulta de estadísticas de los mensajes y su estado de publicación.

6.4 Resultados de Ejecución

- **Número de pruebas ejecutadas**: 7
- **Pruebas exitosas**: 7 (100%)
- **Tiempo de ejecución total**: 5.10 segundos

Todas las pruebas de integración se ejecutaron exitosamente, como lo indica el reporte con el mensaje "**7 passed in 5.10s**" y una cobertura del 100%. Esto confirma que los flujos de interacción entre los componentes del sistema funcionan correctamente en las condiciones de prueba.

6.5 Casos de Prueba Detallados

Caso de Prueba	Descripción	Resultado
Guardar y Consultar un Mensaje	Guarda un mensaje en la base de datos y luego lo consulta para verificar que se almacenó correctamente.	**Pasó**
Eliminar un Mensaje	Elimina un mensaje de la base de datos y confirma que fue removido exitosamente.	**Pasó**
Consultar un Modelo	Realiza un análisis sobre un mensaje existente en la base de datos y verifica la respuesta correcta del modelo.	**Pasó**
Comentario de Soporte	Almacena un comentario de soporte y luego lo consulta para validar el correcto almacenamiento en la base de datos.	**Pasó**
Números Bloqueados	Bloquea un número de celular y lo consulta para verificar su correcta inserción y almacenamiento.	**Pasó**
Mensaje Aleatorio	Obtiene un mensaje aleatorio de la base de datos y valida que el contenido es correcto.	**Pasó**
Consulta de Estadísticas	Consulta las estadísticas de los mensajes para verificar los conteos de mensajes seguros, sospechosos y publicados.	**Pasó**

6.6 Observaciones

- **Advertencia**: Se presenta una advertencia `PytestDeprecationWarning` relacionada con la configuración `asyncio_default_fixture_loop_scope`. Aunque no afecta los resultados actuales, se sugiere revisar esta configuración para mantener la compatibilidad en futuras versiones de `pytest`.

6.7 Conclusión

Las pruebas de integración confirman que los módulos del sistema interactúan correctamente, y los flujos de trabajo principales cumplen con las expectativas. Los endpoints y sus interacciones con la base de datos funcionan adecuadamente, permitiendo al sistema soportar los flujos de datos requeridos sin problemas.

7. Reporte de Resultados de Pruebas

7.1 Resumen de Ejecución de Pruebas

Tipo de Prueba	Pruebas Totales	Pruebas Exitosas	Pruebas Fallidas	Tiempo Total de Ejecución
Pruebas Unitarias	15	15 (100%)	0	5.12 segundos
Pruebas de API	6	6 (100%)	0	4.64 segundos
Pruebas de Integración	7	7 (100%)	0	5.10 segundos
Pruebas de Carga	N/A	N/A	N/A	Aproximadamente 5 minutos

Total de Pruebas Ejecutadas: **28**
Total de Pruebas Exitosas: **28 (100%)**
Total de Pruebas Fallidas: **0**

7.2 Análisis de Resultados

- **Rendimiento del Sistema**: Las pruebas de carga mostraron que el sistema mantiene un buen rendimiento bajo una carga de hasta 100 usuarios concurrentes, con un tiempo de respuesta promedio de 200-300 ms para el percentil 50 y picos iniciales en el percentil 95 de hasta 1200 ms.
- **Estabilidad**: Durante las pruebas de API, unitarias e integración, todos los endpoints funcionaron correctamente y cumplieron con los tiempos de respuesta esperados, mostrando un comportamiento estable y sin errores bajo las condiciones de prueba.
- **Cumplimiento de Objetivos**: Se cumplieron todos los objetivos específicos de las pruebas, ya que el sistema respondió conforme a las especificaciones y mostró estabilidad en sus funcionalidades principales.

7.3 Observaciones y Recomendaciones

- **Optimización en Tiempos de Respuesta Iniciales**: Se sugiere revisar el manejo de solicitudes iniciales bajo alta carga para reducir los tiempos de respuesta del percentil 95 en los primeros segundos.
- **Deprecación de `asyncio_default_fixture_loop_scope` en pytest**: Revisar la advertencia `PytestDeprecationWarning` y actualizar la configuración de `pytest` para evitar problemas de compatibilidad en futuras versiones.
- **Revisión de Escalabilidad**: Considerar pruebas de carga con un número mayor de usuarios concurrentes para evaluar la escalabilidad en futuros desarrollos.

8. Anexos

8.1 Archivos de Configuración de Pruebas

Incluye archivos y configuraciones relevantes como:

- `unit_test.py`, `test_api.py`, `test_integration.py`: Scripts de pruebas unitarias, de API y de integración.
- `locustfile.py`: Script para pruebas de carga con Locust.

8.2 Logs de Ejecución

Capturas de pantalla de resultados de ejecución (incluyendo logs de `pytest` y métricas de **Locust**):

- Resultados de cada tipo de prueba con información detallada del paso de cada caso.
- Logs de pruebas de carga que evidencian el rendimiento y la estabilidad del sistema.

8.3 Documentación de Herramientas

Documentación de las herramientas usadas para referencia futura:

- Manuales de uso de `pytest`, `mongomock`, `unittest.mock` y `Locust`.
- Instrucciones para la ejecución de los scripts de pruebas y configuración del entorno de pruebas.