Introducción

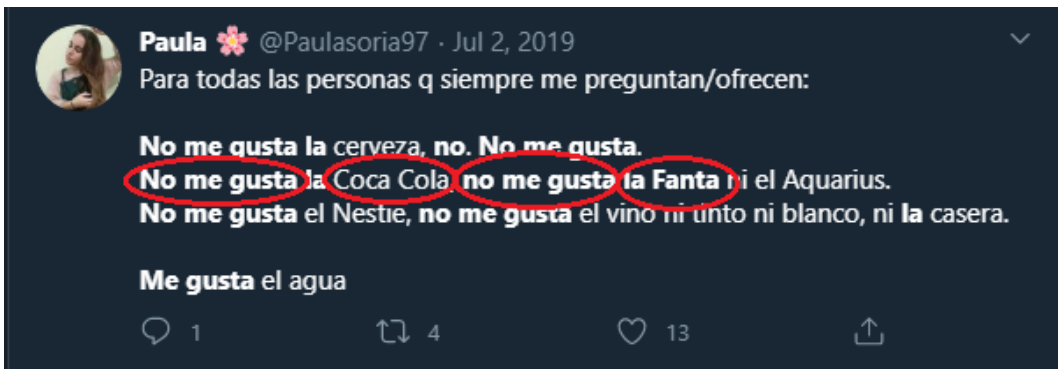
Está de moda el análisis de sentimiento ("*Sentiment Analysis")* para detectar cómo las personas se expresan respecto a marcas, personas públicas, productos, entre otros.

Como parte del análisis de sentimiento se procesan los textos que las personas publican en las redes (por ejemplo, Twitter) para determinar si el sentimiento hacia una de la entidades es positivo o negativo.

Para esto, se analiza distintas palabras clave que pueda contener ese texto, y la cantidad de comentarios como para entender si el texto es positivo, o negativo, y hacia quién puede estar dirigido.

Estas herramientas suelen usarse frecuentemente luego de campañas publicitarias para entender el impacto que tienen y la reacción de clientes frente a la misma, o luego del lanzamiento de nuevos productos, etc.

Por ejemplo:



Tomando esto en cuenta, se pide que diseñe un sistema de análisis de sentimiento básico, que permita definir ciertas palabras (o combinaciones) que indican sentimiento positivo (ejemplo: “me gusta”, “amo”, “me encanta”, “quiero”), otras que indican sentimiento negativo (ejemplo: “no me gusta”, “me aburre”) y otras palabras que representan a las entidades (ejemplo: “El presidente”, “Coca-cola”, “Panadería del Museo”).

Además el sistema deberá permitir ingresar distintas frases, para ser procesadas, y se deberá asociar una categoría a cada frase, relacionándola con la entidad.

Por ejemplo:

“**Me encanta** tomar un **Pomelo Paso de los Toros** después de hacer ejercicio”.

- Positivo
- Pomelo Paso de los Toros

Por último, el sistema deberá permitir gestionar alertas.

Por ejemplo:

Si hay X comentarios negativos en los últimos Y días respecto a una entidad, entonces se deberá generar una alerta.

Para resumir, se pide:

1 - La aplicación deberá mantener tres registros de palabras o frases: positivas, negativas, entidades (Alta, baja, modificación)

2 - La aplicación deberá mantener un registro de frases (poder dar de alta)

3 - La aplicación deberá procesar las frases y asignarle a cada frase una categoría relacionado a la entidad

4 - La aplicación deberá poder gestionar Alarmas, permitiendo definir un lapso de tiempo y una cantidad necesaria de comentarios de un tipo u otro para ser activadas.

Requerimientos

Dado que solo se desea implementar una prueba de concepto, los requerimientos de implementación para el obligatorio son los siguientes:

a) Construir una solución con al menos dos componentes separados, llamados BusinessLogic y UserInterface. El primero debe contener la implementación de las reglas del negocio y el segundo la interfaz de usuario, según se describe más adelante.

b) Ambos componentes deben ser desarrollados usando C# en Visual Studio 2017, y utilizando github como repositorio de código.

c) El componente BusinessLogic debe ser diseñado y construído 100% utilizando TDD y contener la implementación de todas las reglas de negocio que se especifican en la letra.

d) El componente UserInterface debe contener una ventana principal con un menú que permita acceder a las siguientes opciones:
   i) **Registrar sentimientos positivos:** Permite registrar palabras o combinaciones que indiquen un sentimiento positivo. Además debe mostrar una lista de los sentimientos positivos y permitir eliminarlos.

   ii) **Registrar sentimientos negativos:** Permite registrar palabras o combinaciones que indiquen un sentimiento negativo. Además debe mostrar una lista de los sentimientos negativos y permitir eliminarlos.

   iii) **Registrar entidades:** Permite ingresar entidades. Además debe mostrar una lista de los sentimientos negativos y permitir eliminarlos.

   iv) **Ingresar frase:** Permite ingresar una frase al sistema (se debe registrar la fecha de la frase, pudiendo elegir la misma con un control de fechas ya existente)

   v) **Crear configuración de alarma:** Permite configurar una alarma a la que se le puede definir: Entidad, Tipo de alarma (positiva, o negativa), cantidad de posts necesarios para activar, y plazo de tiempo (en días u horas)

   vi) **Reporte de análisis:** una grilla que muestra una frase, la entidad relacionada a esa frase y si es positiva, negativa o neutra (es neutra cuando no se detecta un sentimiento positivo ni negativo).
   vii) **Reporte de Alarmas:** un listado de las alarmas generadas.

IMPORTANTE: Contemplar que en el futuro se puede solicitar nuevas reglas para procesar las frases, y además que se pueden configurar nuevos tipos de alarmas. Es importante diseñar el sistema para minimizar el impacto de estos cambios.

IMPORTANTE 2: Para simplificar el dominio, se puede asumir que a lo sumo existirá una entidad y un sentimiento por frase. Si se detectaran dos entidades o más de un sentimiento, se puede asumir que la frase es neutra y corresponde a la primer entidad encontrada.

NOTA: La totalidad y detalle de los requisitos serán relevados a partir de consultas en el foro correspondiente en aulas, las que deben tener un título descriptivo de su contenido. Para evitar complejidades innecesarias se realizaron simplificaciones al dominio del problema real.

IMPORTANTE FUNCIONALIDAD EXTRA GRUPOS DE A 3:

- Los grupos de a 3 deberán además analizar el grado de “positivismo” o “negatividad” en una frase. Si en la misma encuentran más de un sentimiento positivo, entonces deberán agregar una cuarta columna al reporte de análisis que indique el grado. El mismo puede ser Bajo, Medio o Alto si encuentra uno, dos, o más de dos respectivamente.

Entrega del Obligatorio

A continuación se describen los requisitos esperados para la entrega del obligatorio:

Diseño e Implementación

Se debe entregar una aplicación que contemple toda la funcionalidad descrita en este documento. Para esta primera instancia la solución guardará toda la información en memoria, es decir, no es necesario el uso de base de datos (esto se implementará en la segunda entrega).

El desarrollo de todo el obligatorio debe cumplir:

- Estar en un repositorio **Git**.
- Haber sido desarrollado utilizando **TDD** (desarrollo guiado por pruebas) lo que involucra otras dos prácticas: escribir las pruebas primero (Test First Development) y refactoring. De esta forma se utilizan las pruebas unitarias para dirigir el diseño.

- Cumplir los lineamientos de **Clean Code** (capítulos 1 al 10, y el 12), utilizando las técnicas y metodologías ágiles presentadas para crear código limpio.
- Se requiere escribir los casos de prueba automatizados con el framework de unit tests visto en el curso (MSTest), **documentando** y **justificando** las pruebas realizadas (ver documentación).
- Como parte de la evaluación se revisará el **nivel de cobertura** de los test sobre el código entregado. Por lo que se debe entregar un reporte y un **análisis de la cobertura** de las pruebas justificando cada uno de los valores obtenidos de acuerdo al diseño elaborado para la solución.

Documentación

La documentación entregada debe ser en **un solo** documento digital (**límite 15 páginas)**, que contenga la siguiente información ordenada e indexada:

1. Descripción general del trabajo y del sistema: si alguna funcionalidad no fue implementada o si hay algún error conocido (bug) deben ser descritos aquí.
2. Descripción y justificación de diseño. Para ello se debe incluir:
    a. Diagrama(s) de paquetes mostrando la organización general de la aplicación (no es necesario incluir el paquete de pruebas).
    b. Diagramas de clases: al menos uno por paquete. Los diagramas deberán ser lo más completos y formales posible respecto a lo que se desea comunicar.
    c. Explicación de los mecanismos generales y descripción de las principales decisiones de diseño tomadas. A modo de ejemplo, se debería describir la forma en que la interfaz de usuario interactúa con el dominio, como se almacenan los datos, el manejo de errores y excepciones, o la utilización de polimorfismo.
    d. Breve análisis de los criterios seguidos para asignar las responsabilidades.
3. Cobertura de pruebas unitarias con su debido análisis y justificaciones.
    a. En caso que la cobertura de líneas de código no supere el 90% para alguna funcionalidad, se deberá incluir un análisis sobre porque, explicando que faltó en los test para que no se ejercitara la parte de código no cubierto
    b. Para la funcionalidad análisis de frases y generación de alertas se debe documentar los casos de prueba elaborados, incluyendo: valores inválidos, valores límites, ingreso de tipos de datos erróneos, datos vacíos, datos nulos, omisión de campos obligatorios, formato inválidos, pruebas de las reglas del negocio, etc. Se debe entregar un reporte que muestre evidencia del resultado de ejecutar estos casos de prueba, que puede ser registrada mediante capturas de pantalla (como anexo en el documento) o realizando uno o más videos cortos publicados en Youtube, en los cuales se pueda apreciar claramente la ejecución de las pruebas. Los links a estos videos deben incluirse en este documento y se debe verificar que se hayan compartido correctamente y que un tercero pueda verlos.

Importante: se espera que la descripción y justificaciones sigan un orden lógico, **intercalando diagramas y explicaciones según sea necesario**. Las condiciones de entrega serán evaluadas como si se le estuviese entregando a un cliente real: **prolijidad, claridad, profesionalismo**, etc.

Entrega Online

La entrega debe estar compuesta por:

- Una carpeta con la aplicación compilada en release.
- Código fuente de la aplicación, incluyendo el proyecto que permita probar y ejecutar.
- Documentación digital (incluyendo modelado UML).

La entrega de los obligatorios será en formato digital online. Los principales aspectos a destacar sobre la entrega online de obligatorios son:

1. La entrega se realizará desde gestion.ort.edu.uy
2. Previo a la conformación de grupos cada estudiante deberá estar inscripto a la evaluación. Sugerimos realizarlo con anticipación.
3. Uno de los integrantes del grupo de obligatorio será el administrador del mismo y es quien formará el equipo y subirá la entrega
4. Cada equipo debe entregar un único archivo en formato zip o rar (los documentos de texto deben ser pdf, y deben ir dentro del zip o rar)
5. El archivo a subir debe tener un tamaño máximo de 40mb
6. Les sugerimos realicen una 'prueba de subida' al menos un día antes, donde conformarán el 'grupo de obligatorio'.
7. La hora tope para subir el archivo será las 21:00 del día fijado para la entrega.
8. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc)
9. Aquellos que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor pasar por la oficina del Coordinador o por Coordinación adjunta antes de las 20:00hs. del día de la entrega

En caso de tener una situación particular de fuerza mayor, es necesario dirigirse con suficiente antelación al plazo de entrega, al Coordinador de Cursos o Secretario Docente.

La entrega final del obligatorio debe además estar claramente identificada (mediante un tag con el texto "**entrega_1**") en la rama **master** en el repositorio del grupo de **GitHub** dentro de la organización **ORT-DA1**. Debe incluir:

- **Código fuente** de la aplicación, incluyendo el proyecto que permita compilar, probar, ejecutar y ejecutar **pruebas unitarias**.
- Carpeta con la aplicación compilada en **release**.
- Documentación digital en formato **PDF** (incluyendo modelado **UML**).
- **Importante: La documentación debe incluir la url del repositorio del grupo, de forma que los docentes puedan asociar al grupo con su trabajo en github. Este es un requerimiento mandatorio, y no se revisará el código si no se cumple.**

- No se aceptará que a master se suba un archivo comprimido con la entrega final (ej: Zip, Rar, otros), el contenido de master deberá ser la **integración** de la rama **develop** a la **master**.

Evaluación (15 pts.)

	Evaluación de	**Pts.**	**Criterios de corrección**
Funcionalidad	Implementación de la funcionalidad pedida y calidad de la interfaz de usuario.	5	**Excelente**: Se implementa toda la funcionalidad, tiene buena usabilidad y no hay errores importantes de funcionamiento. **Correcto**: Falta algún punto no central de la funcionalidad, existen detalles no bloqueantes de funcionamiento o la aplicación no es fácil de usar. **No suficiente**: Faltan partes importantes (por alcance o dificultad) de la funcionalidad. Existe gran cantidad de errores o funcionalidades no usables.
Diseño y documento	Diagramas de paquetes Diagramas de clases Justificación del diseño Calidad del diseño Informe de cobertura de los casos de prueba Evidencia de pruebas Claridad de la documentación Organización de la documentación (debe tener un orden lógico y un índice) Completitud de la documentación Prolijidad de la documentación	5	**Excelente**: Se presenta un documento completo en función de lo que se pide, ordenado y prolijo, que explica la solución entregada en todos su aspectos. Se utiliza la notación vista en clase, de manera formal y correcta, para presentar los aspectos importantes del diseño. Se intercalan diagramas y explicaciones, que permiten comprender el diseño de la solución, las decisiones tomadas y la motivación detrás de las mismas. **Correcto:** Se presenta un documento prolijo que describe el diseño de la solución. Se describen los aspectos más importantes, pero hay errores en la nomenclatura. Falta describir aspectos importantes. El orden de la documentación no permite seguir un hilo descriptivo. **No suficiente**: Falta detallar parte importante de la solución. No se usa la notación vista en clase, o se usa en forma equivocada o incompleta en repetidas ocasiones. Se presenta la documentación como una sucesión de diagramas sin explicación.

Código	Desarrollo guiado por las pruebas (TDD) y técnicas de refactoreo de código Buenas prácticas de estilo y codificación y su impacto en la mantenibilidad (Clean Code) Correcto uso de las tecnologías Claridad del código Concordancia con el diseño	5	**Excelente**: El código es prolijo, fácil de entender y cumple con los puntos cubiertos en el curso sobre Clean Code. Lo construido coincide con lo detallado en la documentación de diseño. Se evidencia el uso de TDD mediante los commits en el repositorio. Hay buena cobertura de casos de prueba. **Correcto**: El código es prolijo en general, aunque presenta detalles a mejorar. Lo construido se corresponde en términos generales con lo presentado en el documento. Hay pruebas unitarias, aunque no se evidencie el uso de TDD. **No suficiente**: El código es desprolijo o difícil de entender. No cumple en repetidos casos lo propuesto por Clean Code. No hay pruebas unitarias.