Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de Aplicaciones 1
Obligatorio 1

Felipe Najson (232863)
Santiago Topolansky (228360)

Entregado como requisito de la materia Diseño de Aplicaciones 1

18 de mayo de 2020

Declaraciones de autoría

Nosotros, Felipe Najson y Santiago Topolansky, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Diseño de Aplicaciones 1;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

1. Introducción

Análisis de Sentimientos es una aplicación para la detección de emociones implícitas en comentarios textuales realizados en redes sociales u otros medios de expresión. Esta clase de herramientas suele utilizarse en los estudios sobre marketing para interpretar la opinión pública sobre determinada marca, y la reacción de los clientes hacia distintas acciones de una empresa.

En esta oportunidad, el proyecto se orientó a desarrollar la tecnología de análisis de frases, permitiendo identificar los sentimientos asociados a las mismas. En cuanto a la entrada de datos, por ahora la aplicación recibe de forma manual las frases a analizar, así como todo el resto de la información pertinente (Entidades y Sentimientos).

Además, mediante la definición de alarmas orientadas a determinadas entidades, el cliente será notificado cuando se alcance un determinado número de comentarios de tipo positivo/negativo en un rango de tiempo establecido.

2. Descripción general del trabajo y del sistema

El sistema desarrollado cumple con la siguiente especificación funcional:

1. **Entidades**: El sistema dispone de una interfaz gráfica con una pantalla dedicada al ingreso de las marcas/entidades de las que interesa capturar información. Es posible consultar y eliminar las entidades desde la sección "ELEMENTOS DEL SISTEMA".

2. **Sentimientos**: El sistema permite el ingreso de sentimientos, solicitando como campo obligatorio su tipo (positivo/negativo).
   Es posible consultar y eliminar sentimientos desde la sección "ELEMENTOS DEL SISTEMA".

3. **Frases**: El sistema provee una sección dedicada a la "captura" de las frases que contienen información de relevancia de las entidades del sistema. Se ingresan de forma manual, y requieren el ingreso de la fecha en que se registró la frase.
   Es posible consultar el listado completo de frases desde la sección de "ELEMENTOS DEL SISTEMA".

4. **Análisis**: La funcionalidad central del sistema radica en el algoritmo de análisis de las frases que ingresan. Las frases/comentarios se analizan en búsqueda de alguna de las entidades y sentimientos del sistema, asociándoles una categoría (positiva, negativa o neutra).
   Según las reglas de negocio definidas, si la frase contiene más de una entidad, se asocia el resultado a la primer entidad identificada. Además, si contiene más de un sentimiento de cada tipo, o no contiene ningún sentimiento, inmediatamente se cataloga como comentario neutro.
   Es posible consultar el listado de todos los resultados de análisis del sistema en la sección de "ANÁLISIS".

5. **Alarmas**: El sistema cuenta con una sección dedicada a la configuración de alarmas, para notificar al cliente en caso de que se alcance un determinado número de posteos de un tipo (positivos/negativos) asociados a una entidad del sistema, en una determinada ventana de tiempo que es dinámica (X cantidad de días/horas hacia atrás en el momento de evaluación).
   Todas las alarmas del sistema se re-analizan cada vez que entra una nueva

frase al sistema, más allá de que esté o no asociada a la entidad de la alarma. El sistema notifica mediante el encendido del cartel de alarmas cuando alguna ha sido activada.
Es posible consultar el listado completo de alarmas configuradas en la sección de “ALARMAS DEL SISTEMA”.
NOTA: En esta versión no se incluyó la funcionalidad del borrado de alarmas.

Al momento de la entrega todos los bugs detectados han sido corregidos. No tenemos conocimiento de ninguna funcionalidad que no funcione correctamente según lo especificado.

3. Estructura del Sistema

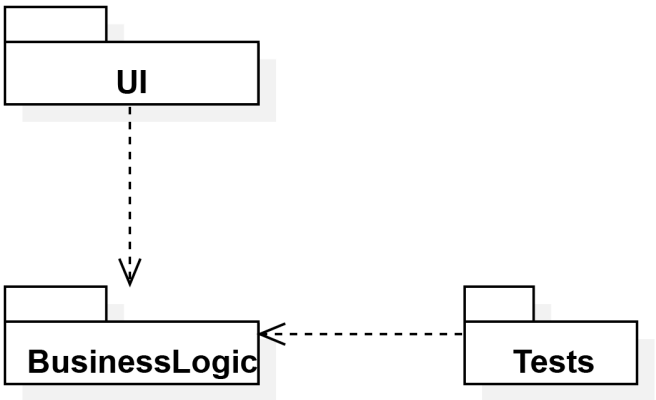
3.1. Organización del Proyecto

La solución esta dividida en 3 paquetes:

- **BusinessLogic**: Contiene las clases centrales que componen el sistema, y responden a las reglas de negocio que determinan cómo la información es creada, almacenada y modificada.
  Aquí se realizan la gestión de la información (entradas de datos, consultas y generación de informes) y todo el procesamiento que se realiza detrás de la aplicación visible para el usuario (Backend).

- **Tests**: Es la parte del sistema que contiene las pruebas unitarias de las clases del paquete BusinessLogic y controla el correcto funcionamiento de las mismas.

- **UI**: Es la parte que sirve de interacción con los usuarios. Es responsable de recolectar los datos de entrada del usuario, que pueden ser de variadas formas, y validarlos y transformarlos, ajustándolos a las especificaciones que demanda el backend para poder procesarlos. También se encarga de reportar errores de parte del usuario previo al ingreso de la información del sistema. Intenta ser llamativo y permitir un manejo ágil de las funcionalidades.



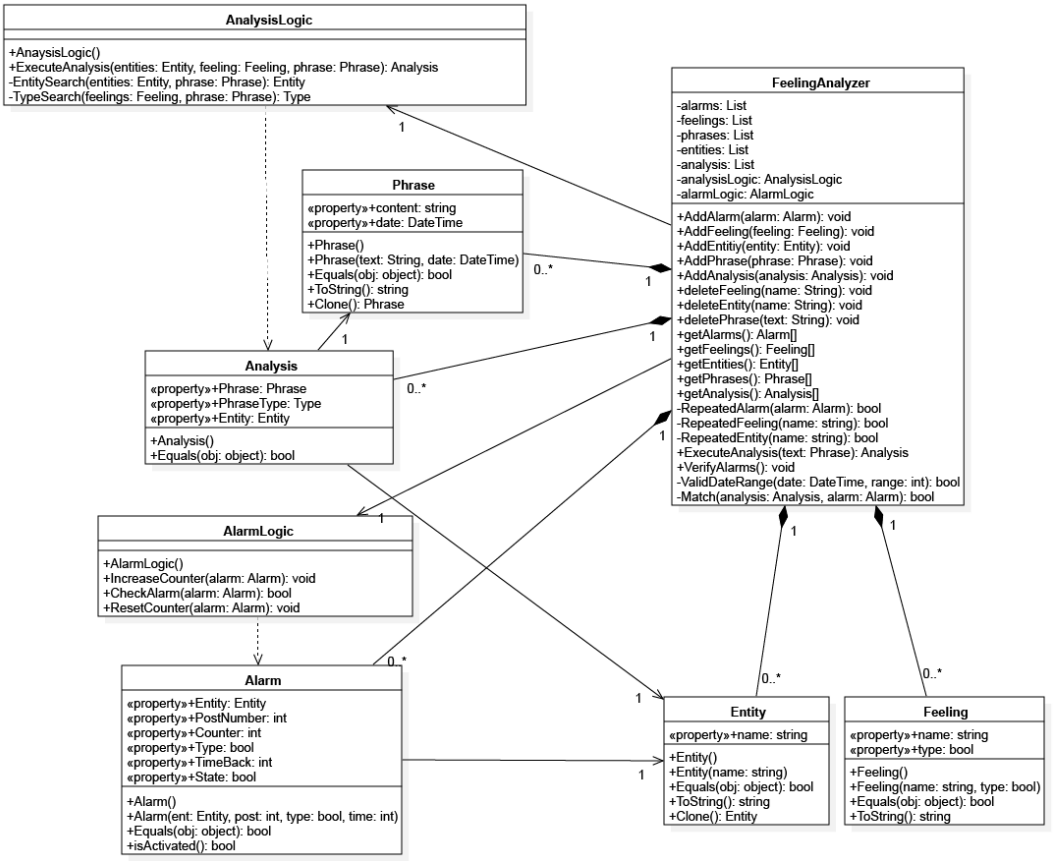
Las clases se agrupan de la siguiente manera en los paquetes:

- **BusinessLogic**
  - Alarm
  - AlarmLogic
  - Analysis
  - AnalysisLogic
  - Entity
  - Feeling
  - FeelingAnalyzer
  - Phrase
- **Tests**
  - AlarmTest
  - AlarmLogicTest
  - AnalysisTest
  - AnalysisLogicTest
  - EntityTest
  - FeelingTest
  - FeelingAnalyzerTest
  - PhraseTest
- **UI**:
  - AnalysisWindow
  - CreateAlarmWindow
  - EntityRegistrationWindow
  - FeelingRegistrationWindow
  - Main Window
  - PhraseRegistrationWindow
  - RegistrationWindow
  - SeeAlarmsWindow
  - SystemElementsWindow

4. Paquete: BusinessLogic

4.1. Organización del paquete

A continuación se presenta el diagrama de clases UML para el dominio, donde se denotan claramente los distintos tipos de relaciones entre las clases, con roles, cardinalidades y dependencias.



- Dependencias:

    1. AnalysisLogic - Analysis: AnalysisLogic interactúa brevemente con los objetos Analysis creandolos y prestando servicios sin mantener ninguna referencia hacia los mismos.
    2. AlarmLogic - Alarm: AlarmLogic interactúa brevemente con los objetos Alarm prestandoles servicios y sin mantener ninguna referencia hacia los mismos.

- Asociación:

    1. Analysis - Phrase: Los objetos Analysis poseen una instancia de Phrase, que no comparten con ninguna otra entidad. Clonan el objeto para no compartirlo con el sistema.
    2. Analysis - Entity: Los objetos Analysis poseen una instancia de Entity, que no comparten con ninguna otra entidad. Clonan el objeto para no compartirlo con el sistema.
    3. FeelingAnalyzer - AlarmLogic: FeelingAnalyzer posee una instancia privada de AlarmLogic, que no comparte con ninguna otra entidad.
    4. FeelingAnalyzer - AnalysisLogic: FeelingAnalyzer posee una instancia privada de AnalysisLogic, que no comparte con ninguna otra entidad.
    5. Alarm - Entity: Los objetos Alarm poseen una instancia de Entity, que no comparten con ninguna otra entidad. Clonan el objeto para no compartirlo con el sistema.

- Composición:

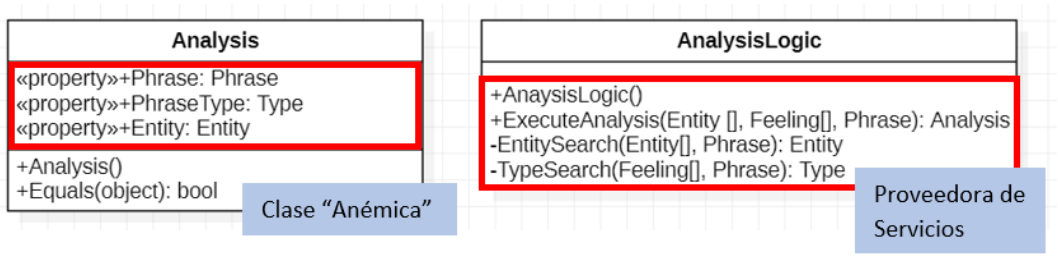
    1. Phrase - FeelingAnalyzer: FeelingAnalyzer posee una lista de objetos Phrase privada, que no comparte con ninguna otra entidad del sistema.
    2. Analysis - FeelingAnalyzer: FeelingAnalyzer posee una lista de objetos Analysis privada, que no comparte con ninguna otra entidad del sistema.
    3. Alarm - FeelingAnalyzer: FeelingAnalyzer posee una lista de objetos Alarm privada, que no comparte con ninguna otra entidad del sistema.
    4. Entity - FeelingAnalyzer: FeelingAnalyzer posee una lista de objetos Entity privada, que no comparte con ninguna otra entidad del sistema.
    5. Feeling - FeelingAnalyzer: FeelingAnalyzer posee una lista de objetos Feeling privada, que no comparte con ninguna otra entidad del sistema.

4.2. Descripción y justificación de diseño

4.2.1. Clases Anémicas

Una de las decisiones de diseño tomadas que resultó más influyente en la estructuración de nuestro proyecto fue la separación de las clases del dominio y su lógica en clases proveedoras de servicio.
La estrategia tenía como fin independizar los "esqueletos" de clases, de sus funcionalidades y reglas de negocio. Como parte del ejercicio de no confundir responsabilidades nos pareció una buena idea que todas las validaciones y operaciones internas de una clase se separaran de la estructura propia de la misma.

Por ejemplo: La funcionalidad de ejecutar un análisis a partir de una frase y las listas de entidades del sistema, se independizó de la instanciación de cada objeto **Análisis** utilizando una clase **AnalysisLogic**, que a partir de la misma información, crea un **Análisis** sin lógica interna.



Cuando ya hubimos implementado el dominio siguiendo este paradigma, nos informamos sobre el nombre que se le otorga en la industria a esta estrategia, "Clases Anémicas". Luego de investigar las opiniones de algunos referentes en el tema, como por ejemplo la opinión de Martin Fowler en la entrada de su blog en 2003, nos replanteamos la decisión. La crítica de Fowler radica en que la esencia del paradigma de P.O.O. se ve vulnerada con esta técnica, no aprovechando la posibilidad de asignar responsabilidades de negocio a las clases pertinentes.[1]

Por razones de plazo y de avance de proyecto optamos por ya no modificar la estructura interna, entendiendo que no era de suma relevancia.

4.2.2. Decisiones de Funcionamiento Interno de Clases

Durante el desarrollo del obligatorio intentamos llevar un registro actualizado de las decisiones de diseño no-triviales tomadas. Clasificamos estas según las clases a las que pertenecen. A continuación presentamos el listado completo de las decisiones más importantes de diseño a bajo nivel:

- Clase Analysis

  1. **PhraseType enum**

El atributo PhraseType es de tipo Type, un enumerado que toma los valores positive,negative,neutral

- Clase AnalysisLogic

  1. **ExecuteAnalysis, EntitySearch y TypeSearch**
     Los 3 métodos reciben arrays con los objetos del sistema (Entidades y Feelings), para evitar dar acceso real a las listas.
  2. **EntitySearch return null**
     Si no se encuentra una Entity en Phrase, retorna null.
  3. **EntitySearch return Entity**
     Si se encuentra una Entity en la frase, se clona al atributo Entity del Analysis a retornar para no dejar una referencia al objeto. Las entidades podrían borrarse del sistema.
  4. **Alias Type**
     Se utilizó un alias Type = BusinessLogic.Analysis.Type. para referenciar el tipo enumerado de la clase Analysis.

- Clase Alarm

  1. **TimeBack**
     El atributo TimeBack guarda el tiempo de evaluación de la alarma en unidad horas. Se optó por guardar el tiempo solo en horas en lugar de en días y horas para simplificar el algoritmo de activación posterior.
  2. **Type y State**
     Se utiliza un atributo booleano para definir el Type de la alarma positivo(true)/negativo(false).
     Se utiliza un atributo booleano para definir el State actual de la alarma positivo(true)/negativo(false).
  3. **Type y State**
     El constructor con parámetros de Alarm clona la Entity para no dejar una referencia al objeto. Las entidades podrían borrarse del sistema.

- Clase AlarmLogic

  1. **Operaciones sobre Alarmas**
     - Permite resetear el Counter de una alarma con ResetCounter()
     - Permite aumentar el Counter de uno en uno con IncreaseCounter().
     - Permite chequear el estado para ver si su contador ya llegó a lo necesario para activarse con CheckAlarm().

- Clase FeelingAnalyzer (Sistema)

  1. **Metodos de add-delete en listas**
     Se usa un método único para cada lista. Esta decisión intenta preparar el programa para un posible cambio en la lógica de alguno de los objetos en

el sistema. Por ejemplo: Agregar una Entity o un Feeling podría implicar distintas reglas de negocio.

2. **Atributos de Clase AnalysisLogic y AlarmLogic**
   El sistema cuenta con una instancia de AnalysisLogic y AlarmLogic para poder interactuar con los Analysis y Alarmas y ejecutar operaciones sobre ellas.

- Clase Feeling

  1. **Feeling Type**
     Se utiliza un atributo booleano para definir el Type del Feeling positivo(true)/negativo(false)

- Clase Entity

  1. **Clone**
     El método Clone permite realizar una copia del objeto para utilizar en los Analysis y Alarmas sin mantener referencias.

- Clase Phrase

  1. **Clone**
     El método Clone permite realizar una copia del objeto para utilizar en los Analysis y Alarmas sin mantener referencias.

*Se utilizaron properties para permitir el acceso de lectura/escritura a todos los atributos de clases del dominio, a excepción de las listas del Sistema.

4.2.3. Almacenamiento de Datos

En esta versión primaria del sistema el almacenamiento de la información se organiza de forma no-persistente, en memoria de ejecución.

Las estructuras de datos elegidas para ello fueron las listas nativas *List* de C#. Ver Diagrama de clases de BusinessLogic en **sección 4.1**.

Este es el detalle de las listas de la clase FeelingAnalyzer:

```
21 referencias
public class FeelingAnalyzer
{
    private List<Alarm> alarms;
    private List<Feeling> feelings;
    private List<Phrase> phrases;
    private List<Entity> entities;
    private List<Analysis> analysis;
    private AnalysisLogic analysisLogic;
    private AlarmLogic alarmLogic;
```

Para preservar la seguridad de la aplicación, sin permitir el acceso de escritura o borrado por parte de otras clases a las listas del sistema, los métodos de acceso a las listas en la clase FeelingAnalyzer retornan arrays del tipo correspondiente para la lectura, y utilizan métodos individuales para el agregado de elementos, como se observa a continuación.

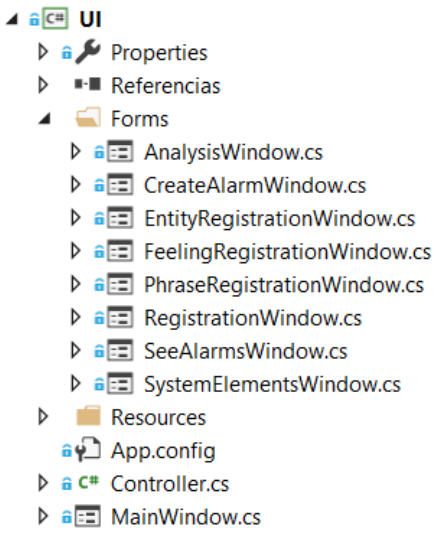
```
6 referencias
public void AddPhrase(Phrase aPhrase)
{
    phrases.Add(aPhrase);
}
3 referencias
public Phrase[] GetPhrases
{
    get { return phrases.ToArray(); }
}
```

Se optó por separar los métodos que realizan estas acciones en cada una de las listas para diferenciar las acciones, habilitando a mayor flexibilidad frente a potenciales cambios en las reglas de negocio o validaciones que se tengan que hacer en cada caso en algún futuro.

5. Paquete: UI

5.1. Organización del paquete

La estructura de ventanas elegida para el User Interface fue la siguiente:



Se dividió el proyecto en 3 partes principales de forma que sea sencillo identificar cada componente de la UI:

- **Ventana Principal**
  En primer lugar tenemos "MainWindow", la cual contiene el menú principal de la aplicación.
  Se estructura en: un panel lateral con cada una de las funciones que realiza el sistema, un panel de titulo de ventana para identificar la ventana en la que se encuentra y un panel principal el cual invocará a las ventanas secundarias.
  La estructura se muestra en la Figura 5.1.

Figura 5.1: Estructura Ventana Principal

- **Ventanas Secundarias**
  Creamos una carpeta para organizar todas las ventanas secundarias del sistema, cada una de ellas cumple con una funcionalidad requerida de la aplicación, estas se invocarán desde la ventana principal.

- **Controller**
  Se incluye el controlador de la aplicación, como el punto de entrada al sistema. Desde aquí se inicia la aplicación.

6. Integración entre Paquetes

6.1. Interacción UI-Dominio

6.1.1. Manejo del sistema

Para utilizar el sistema entre las diferentes ventanas del User Inteface, se tomaron las siguientes medidas:

Cuando se crea la Ventana Principal “MainWindow” se creará con ella la instancia de FeelingAnalyzer, el cual contendrá toda las listas del sistema (en un principio vacías).
Esta instancia única del sistema se pasa entre las ventanas a través del constructor de cada una de ellas, que todas reciben un objeto de tipo ”FeelingAnalyzer”.
De esta manera nos aseguramos trabajar en todas las ventanas sobre el mismo sistema.

Por otra parte, al momento de agregar un elemento al sistema capturamos primero todos los atributos ingresados por el usuario para verificar que esta información sea valida para su ingreso al sistema antes de crear el objeto. Esto nos ayuda a detectar fácilmente cual de los atributos es incorrecto al momento en que el usuario ingresa la información.

6.2. Manejo de errores

De la mano del ingreso de los elementos al sistema tenemos el manejo de errores. Para manejar campos de información incorrecta y excepciones, decidimos mostrar estos mensajes en pantalla con el componente “messageBox”.

Cabe remarcar que los errores producidos a través de reglas de negocio se encuentran en el backend y son lanzados mediante excepciones de tipo “ApplicationException” que se muestran a través de los “messageBox” con try/catch. Por otro lado, los errores para campos vacíos, cadenas de caracteres no admitidas, y otras validaciones de formatos en front, se validan directamente en la UI y se muestran también con “messageBox”.

```
private void btnRegisterFeeling_Click(object sender, EventArgs e)
{
    if (AreEmptyFields())
    {
        MessageBox.Show("No pueden haber campos vacíos");
    }
    else
    {
        try
        {
            Feeling f = new Feeling()
            {
                Name = txtName.Text,
                Type = rbtnPositive.Checked ? true : false
            };
            system.AddFeeling(f);
        }
        catch (ApplicationException ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
        EmptyFields();
}
```

INFO INCORRECTA

EXCEPCIONES

Figura 6.1: Ejemplo Manejo de Errores

7. Pruebas

7.1. Cobertura de Pruebas Unitarias

Estas pruebas consisten en aislar una parte del código y comprobar que funciona a la perfección. Son pequeños tests que validan el comportamiento de una funcionalidad particular de una clase y su lógica aislada.
Resultaron de gran ayuda, ya que con ellas pudimos detectar errores que hubieran sido mas difícil de detectar en fases más avanzadas del proyecto.

Se obtuvo un porcentaje de cobertura de un 96,50 % sobre los paquetes businesslogic y tests.

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
felip_FELIPE-PC 2020-05-17 14_56_16.covera...	31	3,50 %	854	96,50 %
businesslogic.dll	22	5,08 %	411	94,92 %
{ } BusinessLogic	22	5,08 %	411	94,92 %
Alarm	4	8,70 %	42	91,30 %
AlarmLogic	0	0,00 %	14	100,00 %
Analysis	3	6,98 %	40	93,02 %
AnalysisLogic	3	5,45 %	52	94,55 %
Entity	1	4,35 %	22	95,65 %
Feeling	0	0,00 %	28	100,00 %
FeelingAnalyzer	10	5,26 %	180	94,74 %
Phrase	1	2,94 %	33	97,06 %
tests.dll	9	1,99 %	443	98,01 %
{ } Tests	9	1,99 %	443	98,01 %
AlarmLogicTests	0	0,00 %	37	100,00 %
AlarmTest	0	0,00 %	56	100,00 %
AnalysisLogicTests	0	0,00 %	45	100,00 %
AnalysisTest	0	0,00 %	41	100,00 %
EntityTest	0	0,00 %	33	100,00 %
FeelingAnalyzerTests	9	5,52 %	154	94,48 %
FeelingTests	0	0,00 %	34	100,00 %
PhraseTests	0	0,00 %	43	100,00 %

Figura 7.1: Cobertura de Pruebas Unitarias - Visual Studio 2017

El resumen de las pruebas unitarias demuestran que la lógica del código se encuentra estable y verificada, y que funcionará en la mayoría de los casos.

7.2. Casos de prueba para ExecuteAnalysis y VerifyAlarm

Los casos de prueba para estas funcionalidades se encuentran en el **Anexo** al final del documento.

Las pruebas de ejecución en video se encuentran disponibles en el siguiente link.
`https://www.youtube.com/watch?v=DWx2rdAIiX8`

8. Repositorio GitHub

Adjuntamos el link al repositorio de GitHub donde se realizó el desarrollo de la aplicación. Este repositorio se encuentra dentro de la organización ORT-DA1.

`https://github.com/ORT-DA1/DA1-M5B-OBL1-228360-232863.git`

9. Conclusiones

En el plazo de casi dos meses dedicados al proyecto **Análisis de Sentimiento** pudimos cumplir con los objetivos del obligatorio. Desarrollamos una aplicación de tipo WinForms utilizando el framework de .NET (lenguaje C#), e implementando TDD para la creación de las clases del Dominio. Además, tomamos decisiones de diseño bien fundadas lo que consideramos una evolución en la calidad del desarrollo, con respecto a otros proyectos de programación que habíamos realizado anteriormente.

Creemos que logramos un buen producto final, que cumple con los requerimientos especificados y en la mayor medida posible respeta las decisiones de diseño más importantes comentadas en el curso, como el manejo de errores basado en excepciones y la separación de lógica de negocio e interfaz.

Se intentó arribar a un producto final que permitiera su posterior mantenimiento y adaptabilidad a cambios (Desarrollaremos sobre esto en la sección 2.3). También se consideraron ciertas sugerencias de los profesores para profesionalizar el proceso de trabajo. Por ejemplo, optamos por utilizar GitFlow como proceso de SCM para el manejo consistente del repositorio y el control de versiones. Realmente creemos que esta decisión nos ayudó a organizarnos más eficazmente y aunque implicó una curva de aprendizaje inicial mayor, nos introdujo a una gestión más profesional de los cambios y redujo la aparición de conflictos por trabajar en features separadas paralelamente. Siguiendo por la misma línea, tomamos como premisa el desarrollo en idioma Inglés, con comentarios y commits incluidos. También seguimos sus consejos sobre la organización de la aplicación, creando 3 paquetes con responsabilidades claramente diferenciadas: Dominio, Test, y UI.

Bibliografía

[1] M. Fowler. AnemicDomainModel.

Anexo

Referencias:
positivo = +
negativo = -
neutro = ~

El contexto hace referencia a los elementos preexistentes en el sistema al momento del caso de prueba.

Casos de Prueba - Análisis de Frases

Input	Contexto	Resultado	Resultado Esperado	Tipo de Caso
Frase: Me gusta la Coca Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca **Sentimientos**: Me Gusta, + Odio, -	**Análisis** con: Entidad= Coca Frase= Me gusta la Coca TipoFrase = +	**Análisis** con: Entidad= Coca Frase= Me gusta la Coca TipoFrase = +	Normal
Frase:Odio la Coca Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca **Sentimientos**: Me Gusta, + Odio, -	**Análisis** con: Entidad= Coca Frase= Odio la Coca TipoFrase = -	**Análisis** con: Entidad= Coca Frase= Odio la Coca TipoFrase = -	Normal
Frase: Me tomé una Sprite Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Odio, -	**Análisis** con: Entidad= Sprite Frase= Me tomé una Sprite TipoFrase = ~	**Análisis** con: Entidad= Sprite Frase= Me tomé una Sprite TipoFrase = ~	Límite
Frase: Que ganas de tomar una bebida Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	**Análisis** con: Entidad=Vacío Frase=Que ganas de tomar una bebida TipoFrase = ~	**Análisis** con: Entidad=Vacío Frase=Que ganas de tomar una bebida TipoFrase = ~	Límite
Frase:Amo tomar gaseosas Fecha:	**Entidades**: Coca Sprite	**Análisis** con: Entidad=Vacío Frase=Amo	**Análisis** con: Entidad=Vacío Frase=Amo	Límite

16/05/2020 11:31A.M	**Sentimientos**: Me Gusta, + Amo, + Odio, -	tomar gaseosas TipoFrase = +	tomar gaseosas TipoFrase = +	
Frase:Me gustaría tomar una Coca Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	**Análisis** con: Entidad=Coca Frase=Me gustaría tomar una Coca TipoFrase = +	**Análisis** con: Entidad=Coca Frase=Me gustaría tomar una Coca TipoFrase = +	Límite
Frase:Me gustaría tomar una Cocacolita Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	**Análisis** con: Entidad=Coca Frase=Me gustaría tomar una Cocacolita TipoFrase = +	**Análisis** con: Entidad=Coca Frase=Me gustaría tomar una Cocacolita TipoFrase = +	Límite
Frase:AmO tomar SPrITe Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	**Análisis** con: Entidad=Sprite Frase=AmO tomar SPrITe TipoFrase = +	**Análisis** con: Entidad=Sprite Frase=AmO tomar SPrITe TipoFrase = +	Límite
Frase: Me gusta la Coca y la Sprite Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Odio, -	**Análisis** con: Entidad= Coca Frase= Me gusta la Coca TipoFrase = +	**Análisis** con: Entidad= Coca Frase= Me gusta la Coca TipoFrase = +	Regla de Negocio (2 Entidades)
Frase: Me gusta la Coca pero odio la light Fecha: 16/05/2020 11:31A.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Odio, -	**Análisis** con: Entidad= Coca Frase= Me gusta la Coca pero odio la light TipoFrase = ~	**Análisis** con: Entidad= Coca Frase= Me gusta la Coca pero odio la light TipoFrase = ~	Regla de Negocio (2 sentimientos distinto tipo)
Frase: Amo la Sprite! Me gusta mucho Fecha:	**Entidades**: Coca Sprite **Sentimientos**:	**Análisis** con: Entidad= Sprite Frase=Amo la Sprite! Me	**Análisis** con: Entidad= Sprite Frase=Amo la Sprite! Me	Regla de Negocio (2 sentimientos mismo tipo)

16/05/2020 11:31A.M	Me Gusta, + Amo, + Odio, -	gusta mucho TipoFrase = +	gusta mucho TipoFrase = +	
Frase: Me gustaría que me regalen una coca! Fecha: Vacío	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	No es Posible	No es Posible	Datos Vacíos
Frase: Me gustaría que me regalen una coca! Fecha: 2021/05/04 18:57P.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	Mensaje Error: "Fecha inválida, la fecha debe ser igual o anterior al día actual"	Mensaje Error: "Fecha inválida, la fecha debe ser igual o anterior al día actual"	Dato Erróneo
Frase: Me gustaría que me regalen una coca! Fecha: 16/05/2021 19:57P.M	**Entidades**: Coca Sprite **Sentimientos**: Me Gusta, + Amo, + Odio, -	Mensaje Error: "Fecha inválida, la fecha debe ser igual o anterior al día actual"	Mensaje Error: "Fecha inválida, la fecha debe ser igual o anterior al día actual"	Dato Erróneo

Casos de Prueba - Generación de Alertas

Input/s	Contexto	Resultado	Resultado Esperado	Tipo de Caso
Alarma "X": Entidad=Coca Tipo=Positiva Posts=2 Tiempo=2 días **Frase**: Me gusta la Coca Fecha: 16/5/2020 20:07	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho! Fecha=17/5/20 Tipo=+	Se activa Alarma X	Se activa Alarma X	Normal
Alarma "X": Entidad=Coca Tipo=Positiva Posts=2 Tiempo=2 días **Frase**: Me gusta la Coca Fecha: 15/5/2020 20:09	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho! Fecha=17/5/20 Tipo=+	No se activa Alarma X	No se activa Alarma X	Normal
Frase: Me gusta la Fanta Fecha: 17/5/2020 15:37	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho! Fecha=14/5/20 Tipo=+ 2. Entidad=Coca Frase= Que rica es la Coca! Fecha=14/5/20 Tipo=+ **Alarma "Y"**: Entidad=Coca Tipo=Positiva Posts=2 Tiempo= 2 días Estado=Activada	Se desactiva Alarma Y	Se desactiva Alarma Y	Normal
	Análisis: 1. Entidad=Coca	No se activa la	No se activa la Alarma Y	Normal

Frase: Me gusta la Coca Fecha: 17/5/2020 15:37	Frase= La coca me gusta mucho! Fecha=14/5/20 Tipo=+ **Alarma "Y"**: Entidad=Coca Tipo=Positiva Posts=2 Tiempo= 2 días Estado=Desactivada Counter=1 (Frase 1)	Alarma Y		
Frase: Me gusta la Coca Fecha: 8/5/2020 9:50	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho! Fecha=14/5/20 Tipo=+ **Alarma "Y"**: Entidad=Coca Tipo=Positiva Posts=2 Tiempo= 10 días Estado=Desactivada Counter=1 (Frase 1)	Se activa la Alarma Y	Se activa la Alarma Y	Límite
Se elimina la entidad Coca **Frase**: Me gusta la Coca Fecha: 16/5/2020 20:53	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho! Fecha=17/5/20 Tipo=+ **Alarma "Y"**: Entidad=Coca Tipo=Positiva Posts=2 Tiempo= 10 días Estado=Desactivada Counter=1 (Frase 1)	No se activa la Alarma Y	No se activa la Alarma Y	Regla de Negocio(No es posible sumar nuevos análisis si se elimina la entidad)
Se elimina la entidad Coca	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho!	No se desactiva la Alarma Y	No se desactiva la Alarma Y	Regla de Negocio(Los análisis se mantienen más allá de que se

	Fecha=17/5/20 Tipo=+ **Alarma "Y"**: Entidad=Coca Tipo=Positiva Posts=1 Tiempo= 10 días Estado=Activada			elimine la entidad asociada)
Se elimina el sentimiento Me Gusta **Frase**: Odio la Sprite: 16/5/2020 20:58	**Análisis:** 1. Entidad=Coca Frase= La coca me gusta mucho! Fecha=17/5/20 Tipo=+ **Alarma "Z"**: Entidad=Coca Tipo=Positiva Posts=1 Tiempo= 10 días Estado=Activada	No se desactiva la Alarma Z	No se desactiva la Alarma Z	Regla de Negocio (Los análisis se mantienen más allá de que se eliminen los sentimientos que los generaron)
Se intenta agregar Alarma:: Entidad=Coca Tipo=Positiva Posts=1 Tiempo= 10 días	**Alarma "Z"**: Entidad=Coca Tipo=Positiva Posts=1 Tiempo= 10 días	Mensaje Error: "No es posible agregar exactamente la misma alarma"	Mensaje Error: "No es posible agregar exactamente la misma alarma"	Regla de Negocio (No se permiten alarmas repetidas)
Se intenta agregar una alarma con cantidad de Posts -2		No es posible	No es posible	Datos Erróneo
Se intenta agregar una alarma con tiempo= -2 días		No es posible	No es posible	Datos Erróneo

Se quiere ingresar una alarma sin tiempo o sin cantidad de posts.+		Se agrega con cantidades por defecto Tiempo=24hs CantidadPosts=1	Se agrega con cantidades por defecto Tiempo=24hs CantidadPosts=1	Datos Vacíos

Observación:
El caso 3 y 4 de los casos de prueba de “Generación de Alertas” no los pudimos registrar en video porque los plazos no lo permitían.
Fueron probados internamente, y se obtuvo el resultado que se muestra en la tabla.