

Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 1 - Diseño de Aplicaciones 1

Entregado como requisito para la obtención del crédito de la materia Diseño de Aplicaciones 1.

José Pedro Amado – 188885

Francisco López – 209003

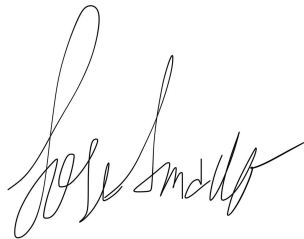
Docente: Sergio Gutierrez

2020

Declaración de autoría

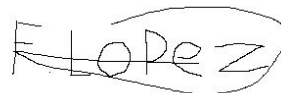
Nosotros, José Pedro Amado y Francisco López, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos la materia 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 que fue contribuido por otros, y que fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



José Pedro Amado

21/05/2020



Francisco López

21/05/2020

Índice

1. Descripción general del trabajo y del sistema	4
2. Descripción y justificación de diseño	5
3. Cobertura de pruebas unitarias	11
3.1 Caso de Prueba Análisis Frase	12
3.2 Caso de Prueba Activar Alarma	13

1. Descripción general del trabajo y del sistema

Durante la realización del proyecto, utilizamos una metodología de asignación de tareas donde decidimos quien desarrollaba qué funciones del programa. La totalidad del proyecto fue desarrollado utilizando la metodología Test Driven Development, la cual asegura la cobertura del código con pruebas unitarias. El proyecto fue almacenado y versionado utilizando GitHub. Pueden acceder haciendo [click aquí](#).

La primera semana constó de la implementación por parte de un integrante de las clases Entity, Sentiment, Publication y Alarm. Al mismo tiempo, el otro integrante se dedicó a desarrollar la interfaz gráfica.

Una vez finalizada la UI y las clases básicas del sistema, el desarrollo fue dirigido a la funcionalidad de guardado de los objetos realizados la semana anterior. Esto fue implementado mediante una clase llamada SystemData, quien contiene una serie de interfaces de guardado, una por tipo de objeto. Luego, se crearon implementaciones para estas interfaces con la serie de clases InMemory.

Una vez terminado esto, el enfoque cambió a comenzar a vincular la interfaz gráfica con la lógica del programa, para empezar a darle funcionalidad a la aplicación. Al mismo tiempo, se trabajó en la clase PublicationAnalyzer. Esta clase se encarga de analizar la frase para detectar una entidad y un sentimiento que hayan sido ingresados en el sistema. Luego crea una relación que vincule esa publicación con la entidad y el sentimiento detectado. Esta clase era fundamental para poder implementar la UI de Reportes.

Una vez finalizado el desarrollo de esa clase, se codificó la clase AlarmAnalyzer, que se encarga de verificar si alguna alarma debería activarse o no, siguiendo la lógica establecida en los requerimientos. Esta clase verifica que existan relaciones para la entidad de la alarma que cumplan con el requerimiento de tiempo y la cantidad de relaciones sea mayor o igual a la requerida por la alarma.

Finalmente, se desarrollaron casos de uso para las funcionalidades de análisis de publicaciones y alarmes. Además, se realizaron pruebas exploratorias del software.

Gracias a esta forma de trabajo metódica y ordenada, se logró cubrir todas las funcionalidades con un nivel de cobertura de pruebas del 100% para la lógica del sistema.

2. Descripción y justificación de diseño

Nuestro proyecto cuenta con dos paquetes, BusinessLogic y UserInterface (figura 1). Como indican sus nombres, BusinessLogic compone toda la lógica del programa, las reglas del negocio, mientras que UserInterface corresponde a la interfaz gráfica desarrollada en esta ocasión. Esta separación de backend y frontend, se debe a la probabilidad de cambio de la interfaz que es mucho mayor a la probabilidad de cambio del dominio, por lo cual la separación asegura la facilidad de cambio al momento de hacer una interfaz nueva.

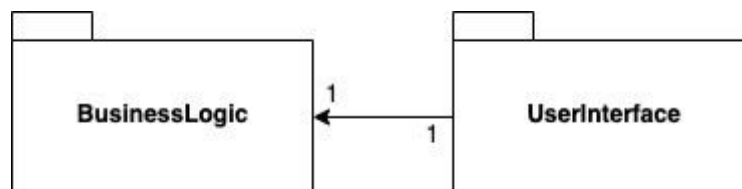


Figura 1 - UML de Paquetes de la Solución.

Las interfaces gráficas a la hora de realizar una acción, crean una instancia de una clase que tiene únicamente el propósito de cumplir dicha acción. Esta clase se encarga de comunicarse con SystemData, quien almacena la información del sistema en memoria, y con las demás clases de BusinessLogic para llevar a cabo dicha acción.

A nivel del paquete BusinessLogic, se decidió implementar los sentimientos mediante herencia, ya que un sentimiento positivo y negativo se comportan de igual forma pero en un futuro podrían cambiar y ser necesario comportamiento distinto para cada uno de ellos, o hasta ser necesarios nuevos tipos de sentimientos. Esto brinda una versatilidad

al momento de expandir el sistema. Por tanto, nuestro proyecto cuenta con una clase `Sentiment` padre y dos clases que lo heredan: `PositiveSentiment` y `NegativeSentiment`. La misma lógica rige para las alarmas, donde tenemos una clase `Alarm` padre y dos clases que la heredan, `PositiveAlarm` y `Negative alarm`.

Como fue comentado anteriormente, el guardado de datos fue implementado con la clase `SystemData`. Ésta utiliza varias interfaces de guardado, cada interfaz se encarga de una clase en específico, y luego esas interfaces son implementadas en clases llamadas `InMemory`. Estas implementaciones guardan los datos en memoria. No existe persistencia por el momento.

A nivel de interfaz, se tuvo en cuenta la usabilidad y la estética, por lo cual el diseño es minimalista y cuenta con un diseño de ventana única. La ventana principal cuenta con un panel el cual va cambiando las vistas al usuario a medida que el mismo selecciona las distintas funcionalidades del sistema. En los casos de ingresos, se tienen en cuenta las excepciones y cuando el usuario comete algún error, se le informa de forma no obstrusiva y se lo asiste a su resolución.

Se asumió que las publicaciones son analizadas al momento de ingresarlas únicamente. Esto implica que si luego se agregan nuevos sentimiento o entidades, las frases existentes no serán analizadas nuevamente. De lo contrario, se deberían analizar todas las publicaciones al momento de ingresar nuevos sentimientos y entidades, lo cual consideramos puede ser mucha carga para el sistema. Eventualmente se podría incorporar una función específica para realizar un análisis de las publicaciones históricas.

En cuanto a las alarmas, éstas serán activadas al momento de ingresar una alarma o una nueva frase y su estado será visible en el reporte de alarmas. Consideramos que ejecutar el analizador de alarmas en todo momento sería cargoso para el sistema, por lo cual a futuro se puede realizar una implementación más exhaustiva y eficiente, que permita se activen las alarmas sin importar la interacción del usuario.

El siguiente diagrama (figura 2) muestra la relación entre la clase SystemData y las clases que se crean para la interacción entre el dominio y las funcionalidades en la UI.

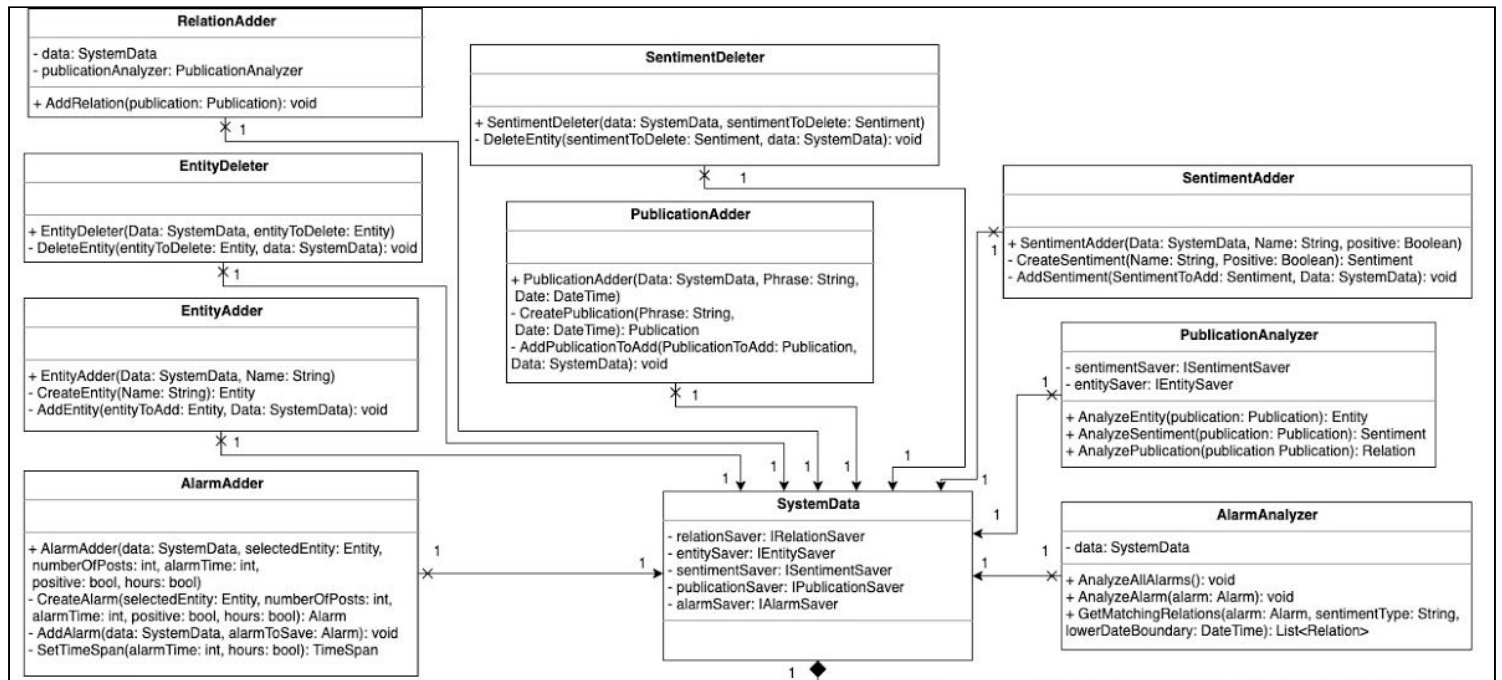


Figura 2 - UML de clase SystemData y asociados.

Las clases Adder y Deleter son clases que se encargan de recibir órdenes de la interfaz gráfica e implementarlas. Fue resuelto de esta forma para que la interfaz gráfica conozca lo mínimo posible sobre la biblioteca de clases BusinessLogic. La razón de esto es que la interfaz gráfica no tenga que cambiar por cambios de BusinessLogic y también que tenga la mínima carga de lógica posible.

Resumiendo, estas clases lo que hacen es ser construidas por parámetro desde la UI y ya desde el constructor hacer lo que requiere la UI sin que la misma de más órdenes. Estas clases están implementadas para solo responder a una orden de la UI (una sola responsabilidad) y todas se comunican con el SystemData para modificar los datos de este. Por ejemplo, EntityAdder se encarga de agregar una entidad al SystemData, o mejor dicho decirle a SystemData que se agregue una Entidad.

La siguientes dos imágenes (Figura 3 y 4) muestran otro extracto del UML del paquete BusinessLogic, en la misma se aprecia la relación entre SystemData y las interfaces por las cuales está compuesta. Además, se aprecian las implementaciones de las interfaces.

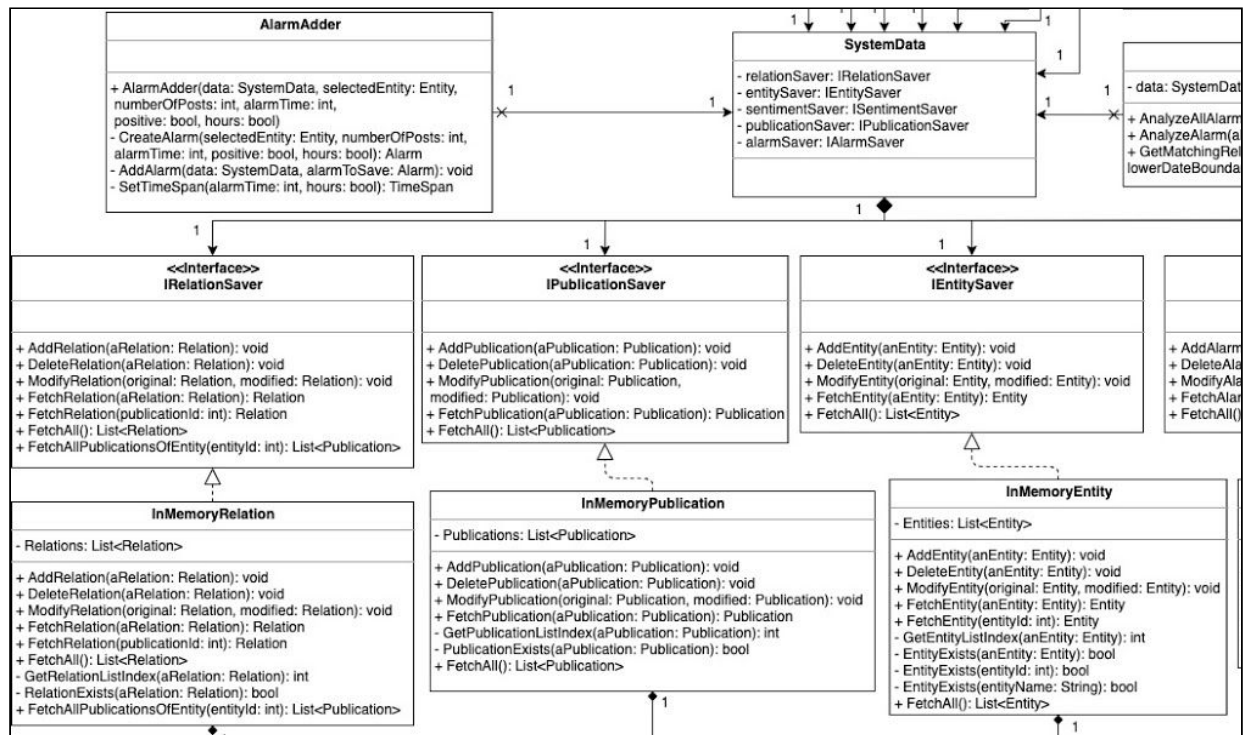


Figura 3 - UML de la relación entre SystemData y algunas de las interfaces.

La clase SystemData tiene la responsabilidad de guardar los datos en runtime de la aplicación. Fue resuelto el guardado a través de varias interfaces, cada una enfocada a un objeto distinto. Estas interfaces son luego implementadas por las clases InMemory. Esta solución permite luego poder cambiar la implementación de guardado sin que tener que cambiar clases como SystemData u otras que usen SystemData.

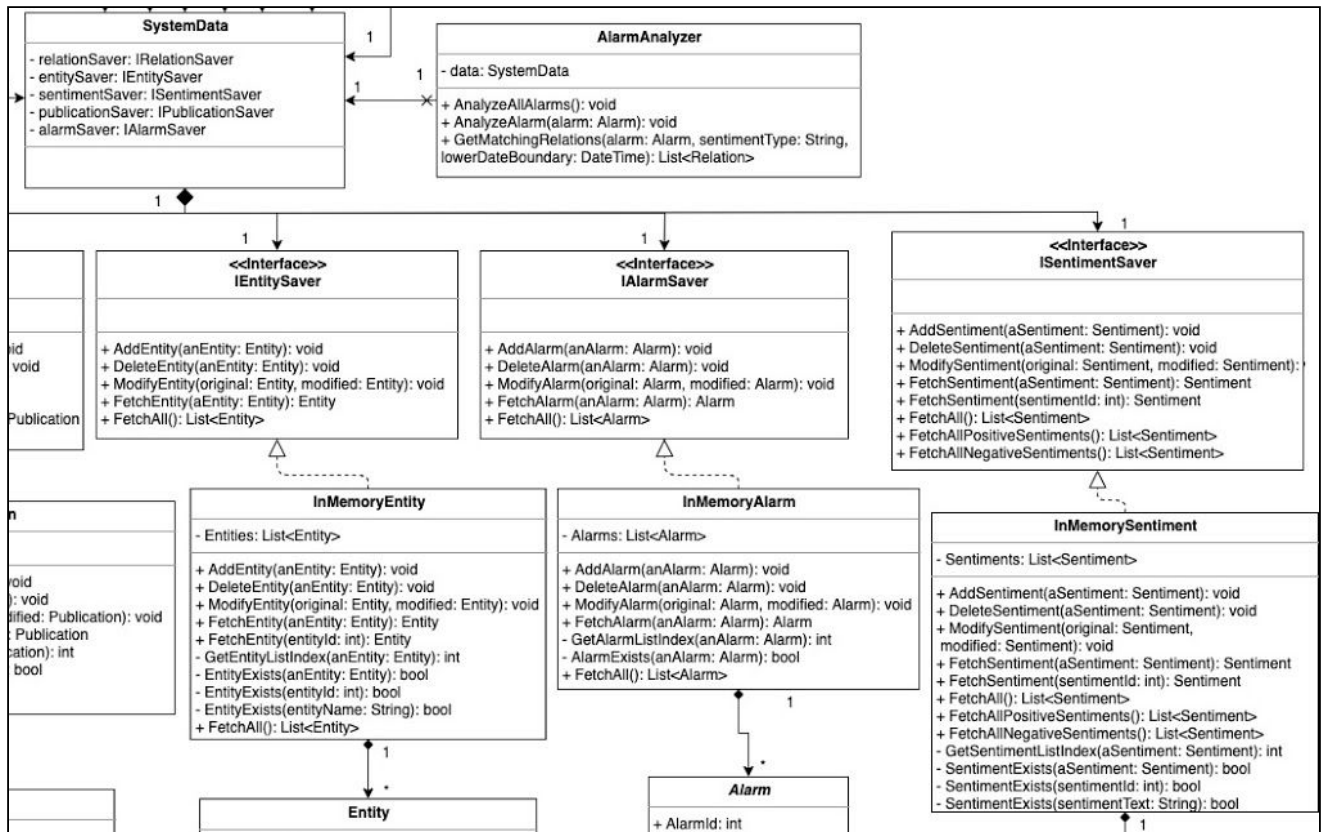


Figura 4 - UML de la relación entre SystemData y el resto de las interfaces.

Finalmente, esta imagen (figura 5) muestra la relación entre las implementaciones de las interfaces y los objetos base que almacenan.

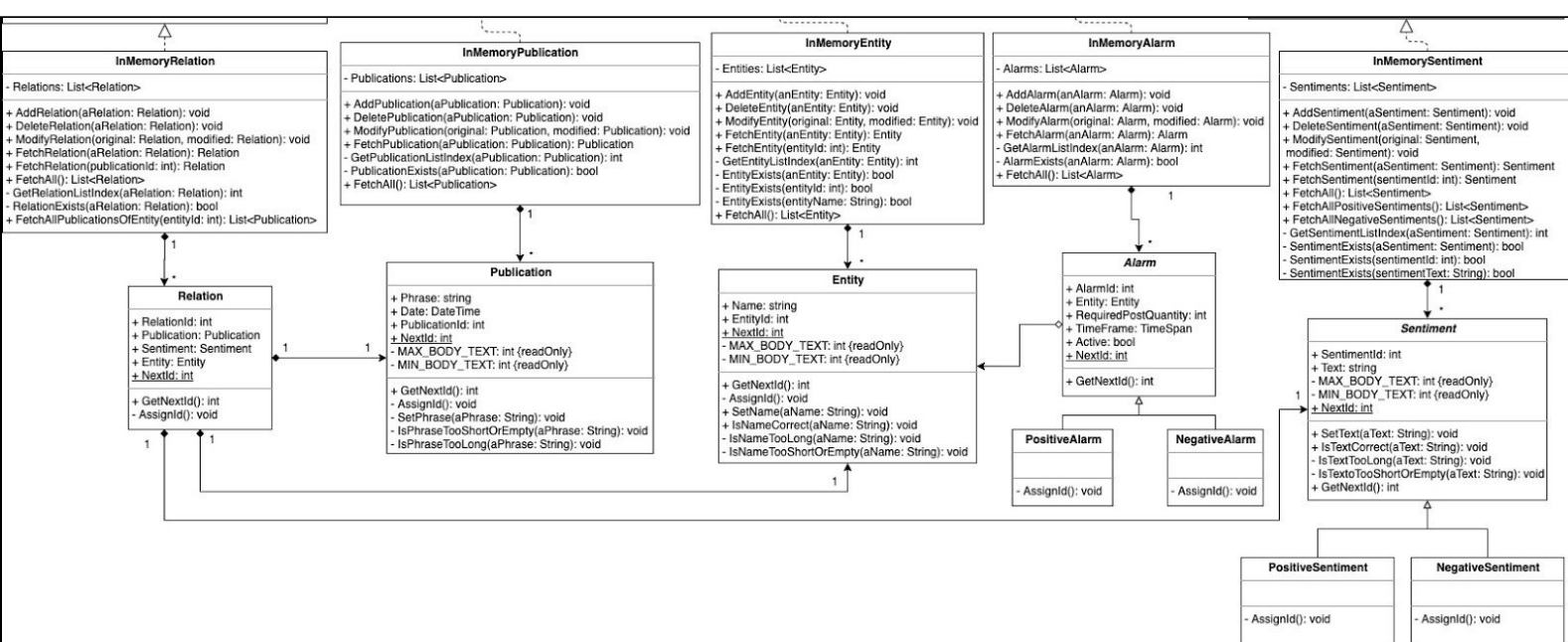


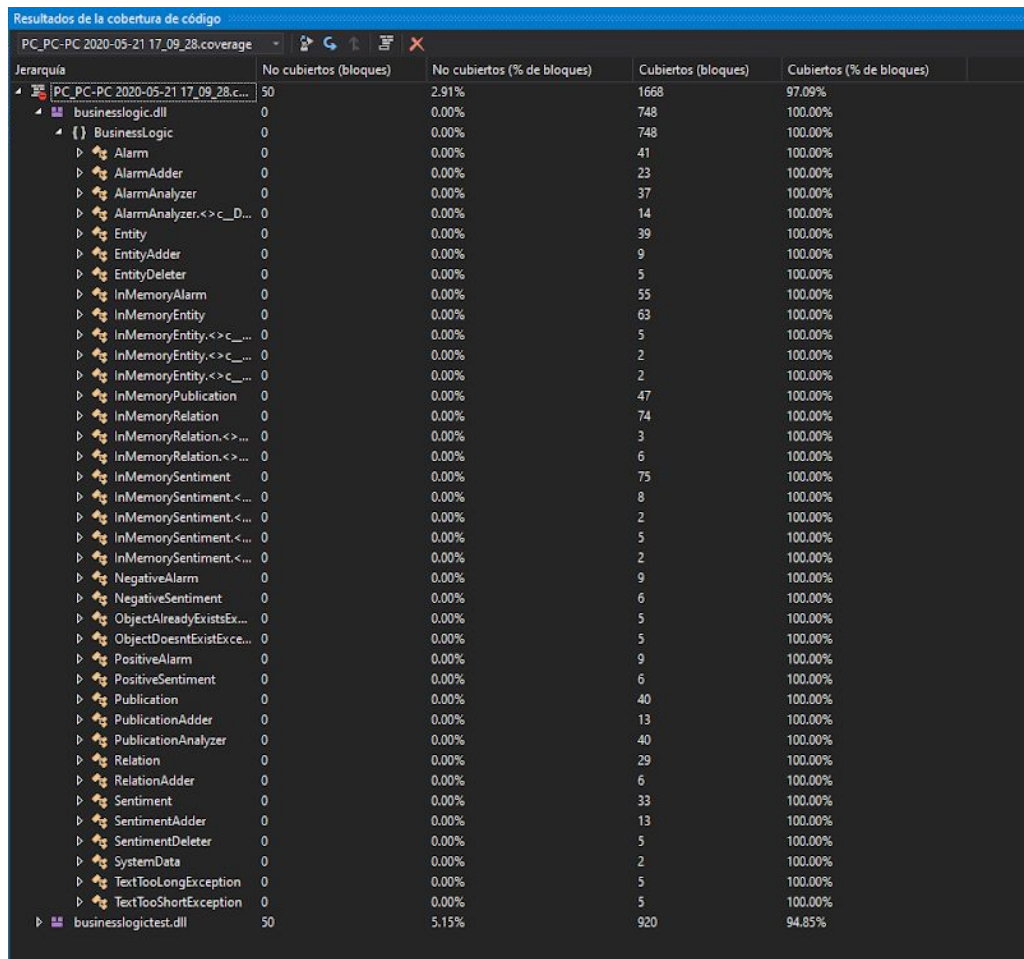
Figura 5 - UML de la relación entre las implementaciones de las interfaces y las clases base que almacenan.

La razón por la que el sistema de guardado fue implementado de esta manera es dividir responsabilidades. Las clases como Entity se encargan de sí mismas y las clases InMemory se encargan de guardarlas sin que se enteren las clases de objetos. Estas clases InMemory implementan las interfaces Saver para así poder cambiar el método de guardado si es necesario y no afectar a ninguna clase con este cambio.

En cuanto a las excepciones, se utilizan algunas clases propias para manejar excepciones de existencia y creación de objetos, y otras para largo de textos. Estas no consideramos fuera necesario incluirlas en los diagramas. Los mensajes dados por estas son pasados a la UI para que el usuario pueda percatarse de los errores y solucionarlos.

3. Cobertura de pruebas unitarias

Gracias al uso de la metodología de TDD, la cobertura de código del dominio de la solución es del 100%. La siguiente imagen evidencia la existencia y los niveles de cobertura reportados (figura 6).



Results of code coverage analysis for PC_PC-PC 2020-05-21 17_09_28.coverage. The table shows a hierarchy of code blocks with 0 uncovered blocks and 100% coverage across all items.

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
PC_PC-PC 2020-05-21 17_09_28.c...	50	2.91%	1668	97.09%
businesslogic.dll	0	0.00%	748	100.00%
BusinessLogic	0	0.00%	748	100.00%
Alarm	0	0.00%	41	100.00%
AlarmAdder	0	0.00%	23	100.00%
AlarmAnalyzer	0	0.00%	37	100.00%
AlarmAnalyzer.<>c_D...	0	0.00%	14	100.00%
Entity	0	0.00%	39	100.00%
EntityAdder	0	0.00%	9	100.00%
EntityDeleter	0	0.00%	5	100.00%
InMemoryAlarm	0	0.00%	55	100.00%
InMemoryEntity	0	0.00%	63	100.00%
InMemoryEntity.<>c_...	0	0.00%	5	100.00%
InMemoryEntity.<>c_...	0	0.00%	2	100.00%
InMemoryEntity.<>c_...	0	0.00%	2	100.00%
InMemoryPublication	0	0.00%	47	100.00%
InMemoryRelation	0	0.00%	74	100.00%
InMemoryRelation.<>...	0	0.00%	3	100.00%
InMemoryRelation.<>...	0	0.00%	6	100.00%
InMemorySentiment	0	0.00%	75	100.00%
InMemorySentiment.<...	0	0.00%	8	100.00%
InMemorySentiment.<...	0	0.00%	2	100.00%
InMemorySentiment.<...	0	0.00%	5	100.00%
InMemorySentiment.<...	0	0.00%	2	100.00%
NegativeAlarm	0	0.00%	9	100.00%
NegativeSentiment	0	0.00%	6	100.00%
ObjectAlreadyExistsEx...	0	0.00%	5	100.00%
ObjectDoesntExistExce...	0	0.00%	5	100.00%
PositiveAlarm	0	0.00%	9	100.00%
PositiveSentiment	0	0.00%	6	100.00%
Publication	0	0.00%	40	100.00%
PublicationAdder	0	0.00%	13	100.00%
PublicationAnalyzer	0	0.00%	40	100.00%
Relation	0	0.00%	29	100.00%
RelationAdder	0	0.00%	6	100.00%
Sentiment	0	0.00%	33	100.00%
SentimentAdder	0	0.00%	13	100.00%
SentimentDeleter	0	0.00%	5	100.00%
SystemData	0	0.00%	2	100.00%
TextTooLongException	0	0.00%	5	100.00%
TextTooShortException	0	0.00%	5	100.00%
businesslogictest.dll	50	5.15%	920	94.85%

Figura 6 - Captura del resultado del análisis de cobertura de pruebas unitarias.

A continuación presentamos las pruebas básicas realizada sobre el software de forma exploratoria. Dada las restricciones de tiempo y el alto nivel de cobertura unitaria alcanzado, se realizaron pruebas funcionales pero consideramos prudente realizar pruebas adicionales antes de pasar el programa a producción.

3.1 Caso de Prueba Análisis Frase

- **Nombre del Caso:** Análisis de Frase
- **Descripción:** El motivo de este caso es el de ver cómo funciona la funcionalidad de analizar frases
- **Escrito por:** Francisco López - Fecha 20/05/2020
- **Ejecución uno:** Análisis de Frase - **Estado:** Correcto
- **Ejecutada por:** Francisco López - **Fecha:** 20/05/2020
- **Precondiciones:**
 - Una entidad Creada
 - Un sentimiento Creado

Paso	Acción	Resultado esperado	Pasado/Fallado
1	Ingresa frase : CocaCola Es bueno	Se crea la frase sin problemas	Pasado
2	Ir a la ventana de reportes	Se muestra la ventana de reportes y podemos encontrar nuestra frase	Pasado
3	Seleccionar nuestra frase	Se muestra que esta publicación habla de la entidad CocaCola y que es una publicación positiva porque contiene el texto “Es bueno”	Pasado

Curso Alternativo: Mayúsculas

Paso	Acción	Resultado esperado	Pasado/Fallado
1	Ingresa frase : CocACoLa Es buEnO	Se crea la frase sin problemas	Pasado
2	Ir a la ventana de reportes	Se muestra la ventana de reportes y podemos encontrar nuestra frase, Aquí vemos como no importan las mayúsculas a la hora de encontrar entidades.	Pasado
3	Seleccionar nuestra frase	Se muestra que esta publicación habla de la entidad CocaCola y que es una publicación positiva porque contiene el texto “Es bueno”. Observamos que no se modificaron las mayúsculas	Pasado

3.2 Caso de Prueba Activar Alarma

- **Nombre del Caso:** Generación de Alerta
- **Descripción:** El motivo de este caso es el de ver cómo funciona la funcionalidad de levantar alarmas
- **Escrito por:** Francisco López - Fecha 20/05/2020
- **Ejecución uno:** Generacion de Alerta - **Estado:** Correcto
- **Ejecutada por:** Francisco López - **Fecha:** 20/05/2020
- **Precondiciones:**
 - Una entidad Creada
 - Un sentimiento Creado
 - Una alarma creada

Paso	Acción	Resultado esperado	Pasado/Fallado
1	Ingresar frase : CocaCola Es bueno	Se crea la frase sin problemas	Pasado
2	Se va a la ventana de alarmas	Se muestra la ventana de alarmas y podemos encontrar que luego de ingresar la frase se ha activado una alarma que sigue las publicaciones positivas de CocaCola	Pasado