Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio II

Diseño de Aplicaciones I

Tatiana Bellón (232624)

Camila Taeger (234087)

Sebastian Gonzalez (208938)

25 de junio 2020

Índice

1. Descripción general del Trabajo y del Sistema.

El trabajo se enmarca en la implementación de un sistema de análisis de sentimiento 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 las 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.

Además se solicitó que se pudiera identificar y persistir el autor de cada frase, para luego realizar diferentes reportes.

La UI nos permite:

Dar mantenimiento (Alta, Baja, Modificación) de las objetos: Entity, Phrase, Author, Sentiment y Alarm.

Además se cuenta con tres reportes. Uno de ellos llamado Report Alarm, en este podemos ver un listado con todos los tipos de alarmas, siendo estos AuthorAlarm y EntityAlarm. El listado Permitirá ver cuales de estas están activas y sus características principales.

Por Ejemplo en el caso de las alarmas de tipo author, veremos quienes fueron los autores que participaron en la activación de esa alarma.En el Report Analysis podremos ver un listado de las frases ingresadas en el sistema, así como una evaluación, del grado de sentimentalismo (Positivo o negativo) de una frase, catalogados entre: Low, Medium, High. Por último en el Author Report podremos ver un listado de los autores en el cual se podrán aplicar diferentes filtros, como por ejemplo promedio de frases de cada author, cantidad de entidades mencionadas por cada uno, etc. Esta información filtrada no solo se verá en un listado sino que se representará en una gráfica para los primeros 10 authors.

2. Descripción de la implementación

Para llevar a cabo la ejecución de el segundo obligatorio, y luego de haber modificado nuestros equipos, pasamos por la instancia de unificar dos proyectos para obtener lo mejor de ambos, es por esto que luego de decidir un proyecto base sobre el cual trabajar se implementaron los cambios necesarios para cumplir con los requisitos extras para grupos de tres integrantes de la letra del obligatorio 1, ya que nos pareció pertinente no obviar y centrarnos solamente en el requisito extra del obligatorio dos.

Alguno de estos cambios fueron por ejemplo validar que una frase pudiera efectivamente contener más de un sentimiento, permitiéndonos esto medir el grado de positivismo y negativismo de una frase.

También respecto al requisito extra del obligatorio 2, se realizó la implementación de una gráfica tal y como se solicitaba que fuera capaz de representar tres tipos diferentes de consultas para un número acotado de autores.

El proyecto fue desarrollado utilizando code first con Entity Framework y La persistencia ahora se aloja en una base de datos especificada, utilizando como motor Microsoft SQL Server 2014.

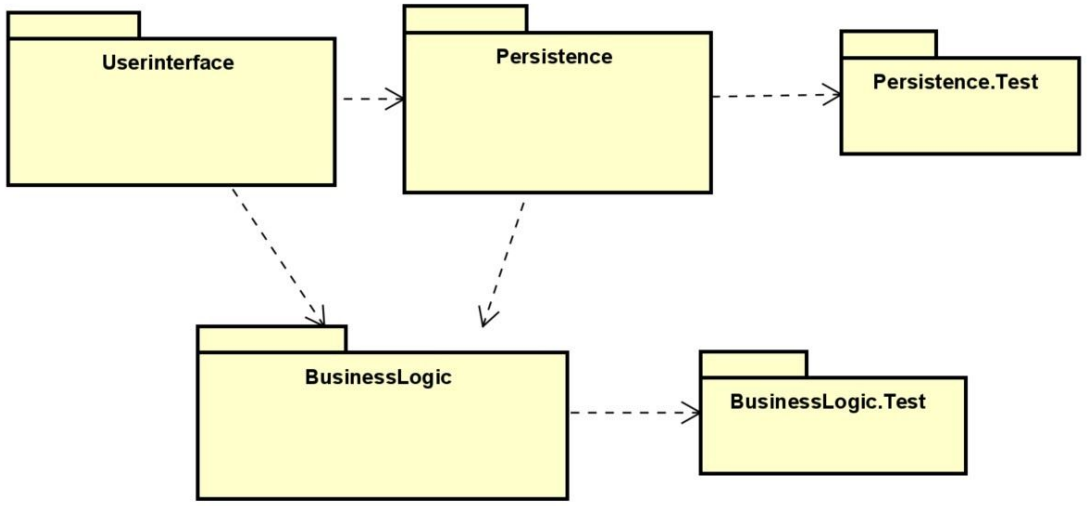
También se crearon clases auxiliares para poder trabajar con modelos de objetos para cosas específicas, por ejemplo en la UserInterface dentro de la carpeta DTO existe una clase llamada AuthorListItem creada para cargar la información que posteriormente se muestra en el AuthorReport, donde se carga una lista y una chart.

3. Diagrama de paquetes

Se presenta el diagrama de paquetes realizado para el actual estado del sistema.

Cabe destacar que todos los diagramas estarán incluidos en una carpeta con dicho nombre en el proyecto, para su mejor visualización.

El diseño aplicado permitió una adecuada dependencia entre proyectos, evitando que en futuros mantenimientos las modificaciones sean mínimas. Por ejemplo si se decidiera cambiar la forma en que se persiste la información, los proyectos de BusinessLogic e UserInterface no se verán afectados.

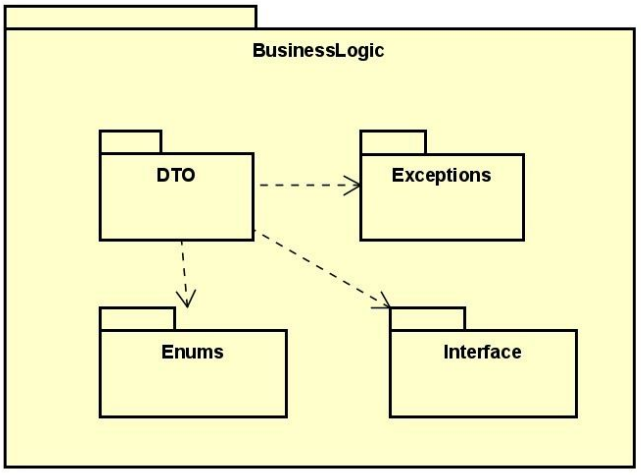


4. Diagramas de clases

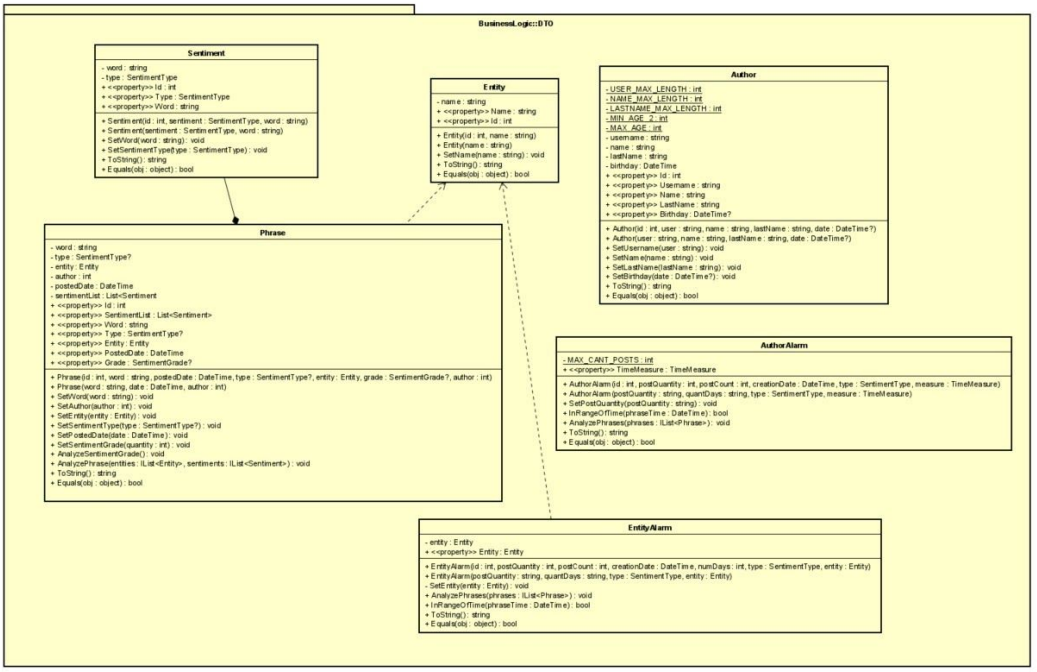
A continuación se detallan los diagramas de clases de cada paquete.

BusinessLogic:

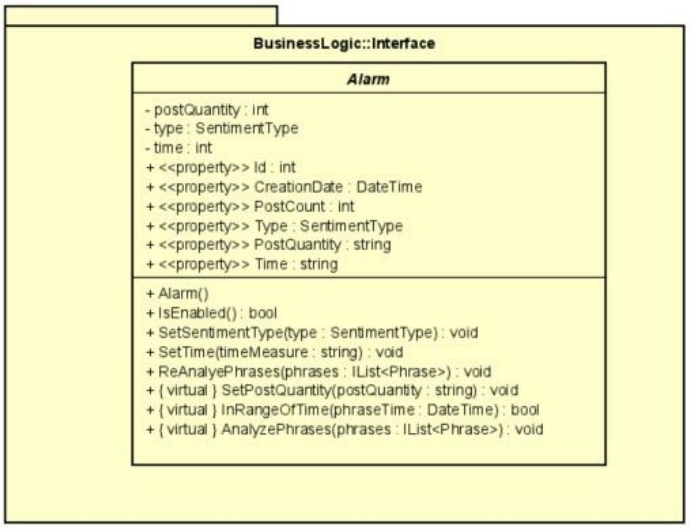
Este paquete está compuesto por cuatro subcarpetas: DTO, Interface, Exceptions y Enums.



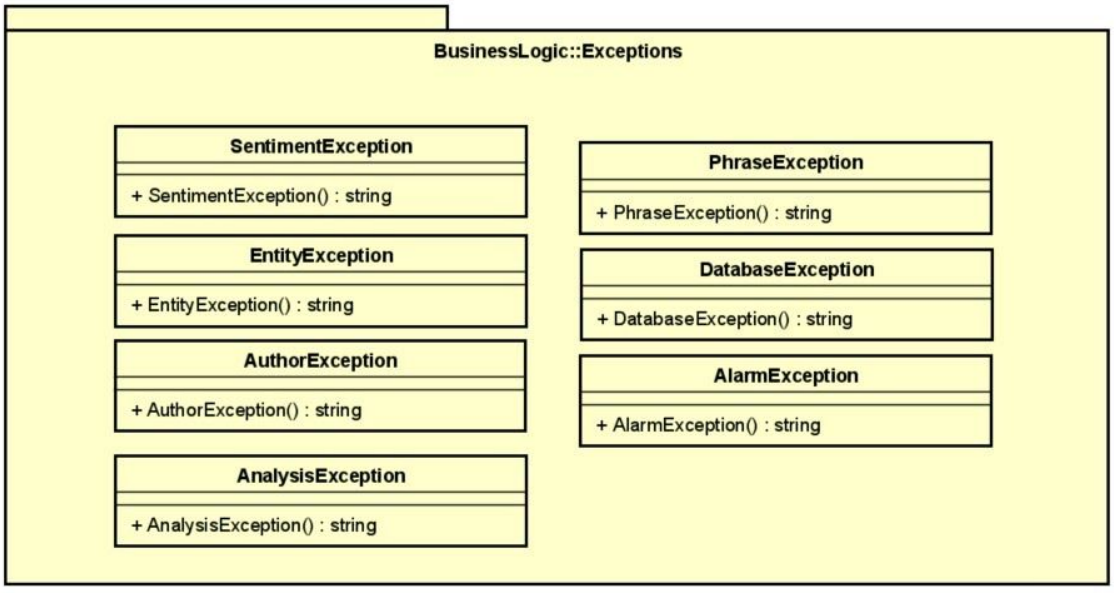
En DTO tenemos las clases correspondientes a la lógica de las clases del dominio de nuestra aplicación, como se ve en el siguiente diagrama existen por ejemplo dos tipos de alarmas representadas en dos clases diferentes, AuthorAlarm y EntityAlarm, estas extienden de una clase abstracta denominada Alarm.



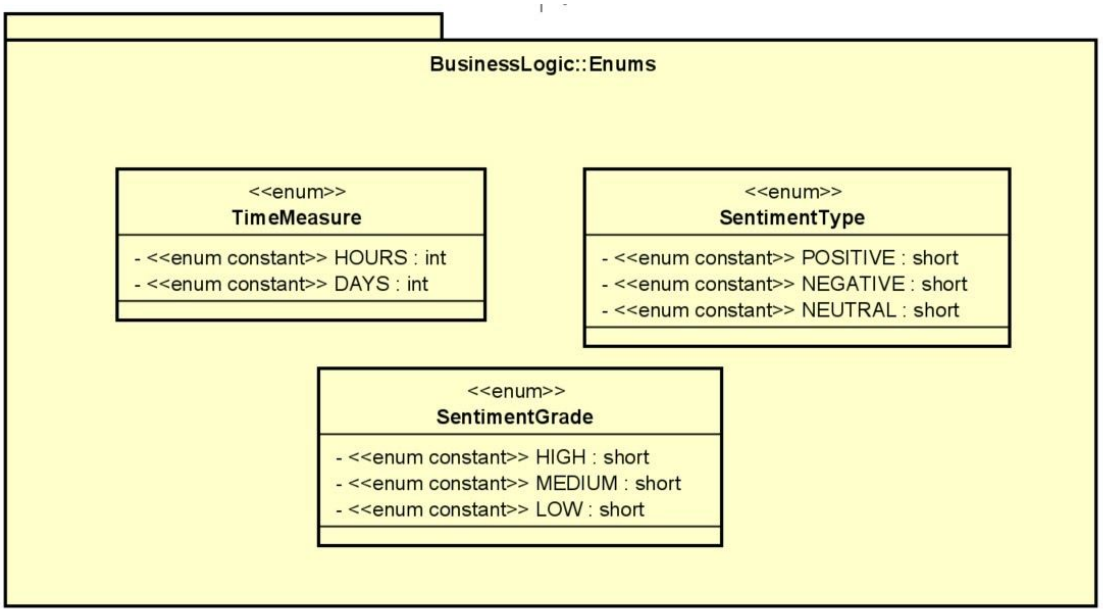
En Interface tenemos la clase Alarm anteriormente mencionada, esta es una clase abstracta donde se definieron los comportamientos que los tipos de alarma tienen en común.



En Exceptions tenemos un conjunto de clases destinadas a controlar diferentes excepciones.

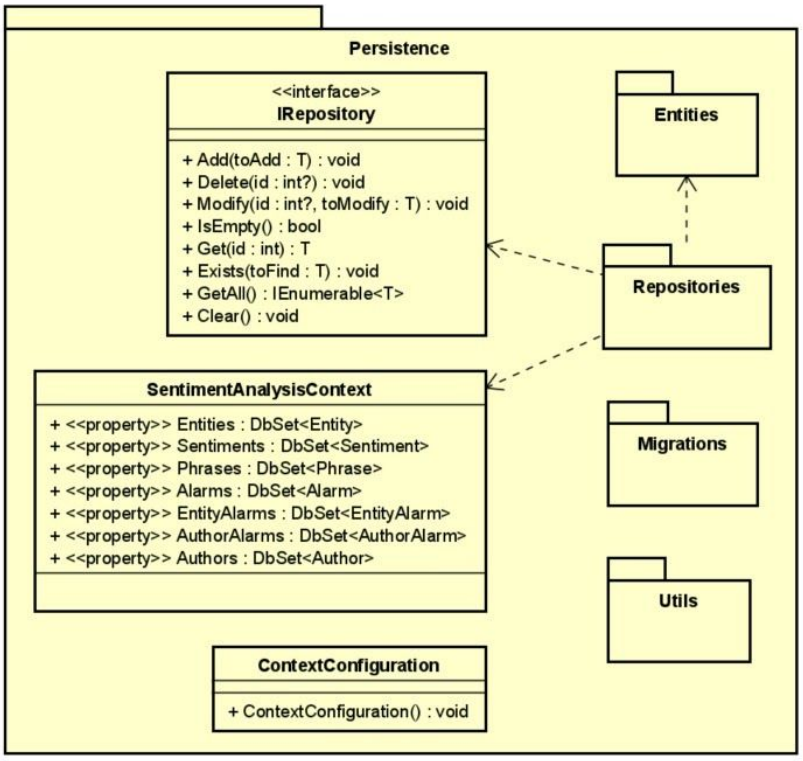


Enums contiene los enumerados que se utilizaron para definir los posibles valores que diferentes objetos o atributos implementados en el sistema podían tomar. Por ejemplo, Los únicos posibles valores para el atributo type de la clase Phrase , serán los definidos por el enum SentimentType.

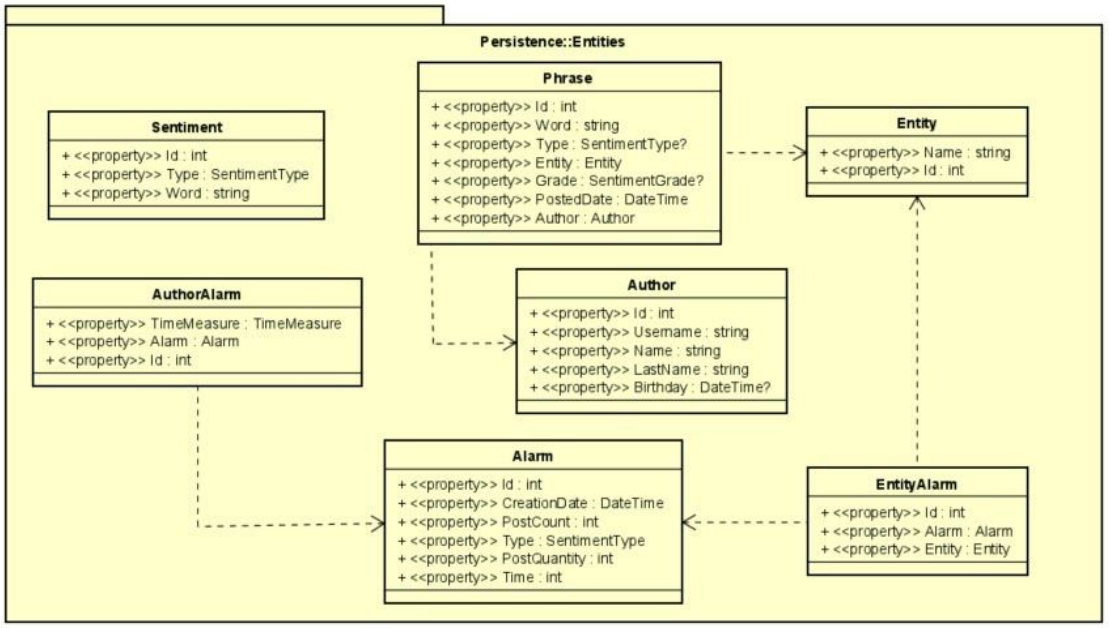


Persistence:

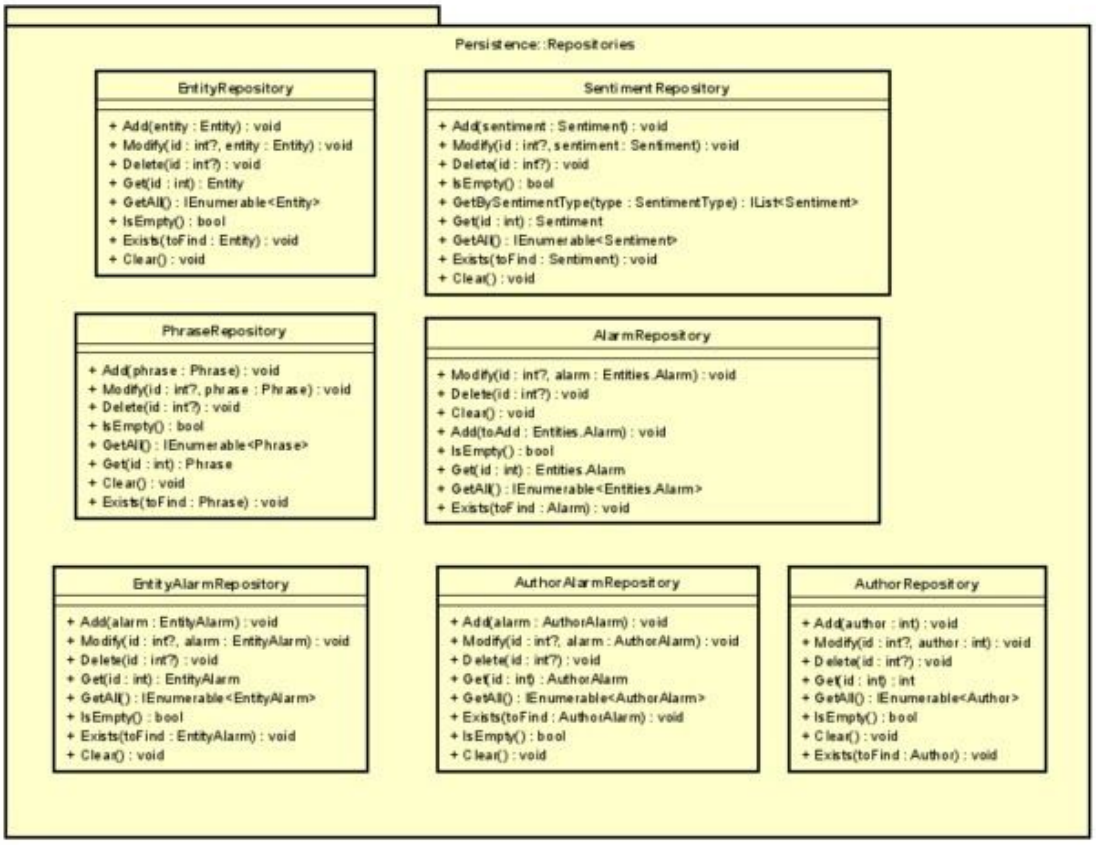
Este proyecto está compuesto por las subcarpetas Entity, Migrations, Repository, Utils y tres clase llamadas, contextConfiguration implementada para setear configuraciones iniciales que le indican a Entity Framework cómo interactuar con la base de datos, IRepository, una interface que define los comportamientos que tienen en común los repositorios, para que luego estos los implementen, y por último una clase SentimentAnalysisContext que extiende de DbContext donde definimos los dataset para todos los objetos que queremos persistir.



Dentro de la carpeta Entity tenemos un conjunto de clases que representan el dominio de nuestra aplicación, con cada una de sus properties

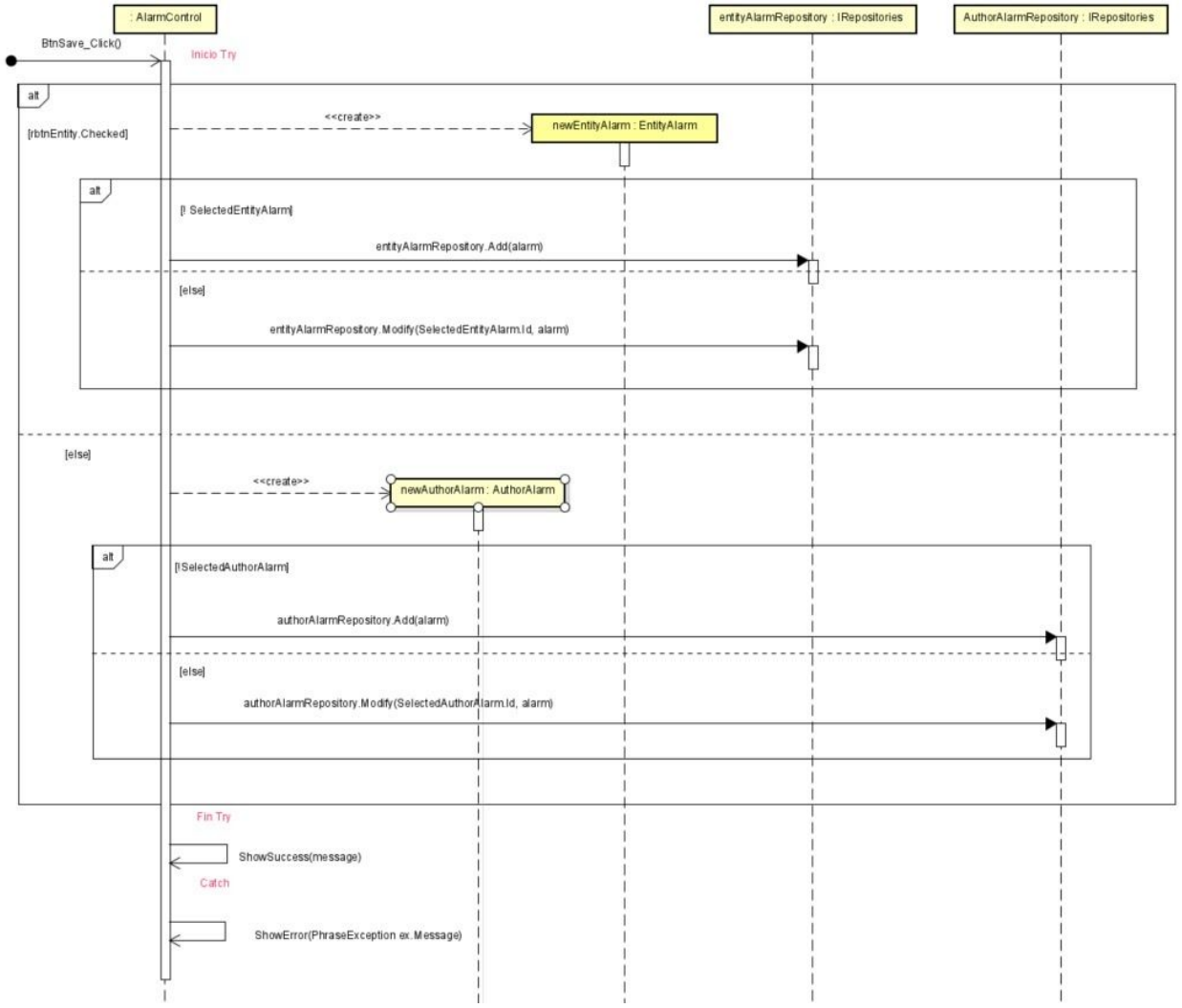


En la carpeta Repository tenemos agrupadas las clases repositorios, estas son las encargadas de comunicarse con el context para lograr persistir nuestros datos.

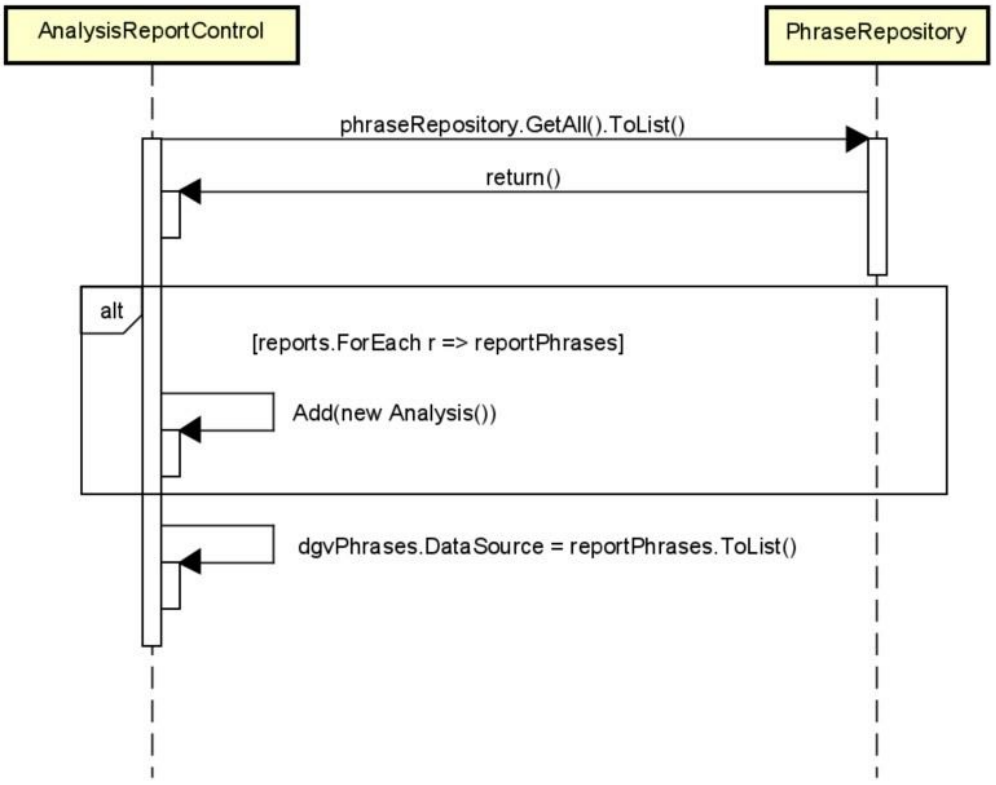


5. Diagramas de Interacción

A continuación se detalla el flujo de comportamiento de un alta de alarma, pasando desde la creación de su instancia y las validaciones correspondientes, hasta su posterior persistencia en base de datos.



Además Se diagramó el flujo interactivo que realiza el sistema al listar el reporte de frases en AnalysisReportControl, desde solicitar los datos al repositorio en persistence hasta listarlos en una grid de la UserInterface.



6. Justificación de diseño

En el desarrollo de la solución se buscó adaptarse y aplicar la mayor cantidad de patrones de diseño vistos en clase, así como los denominados SOLID Y GRASP.

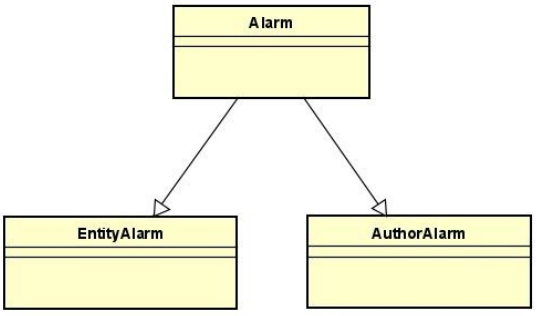
A continuación mencionaremos un ejemplo de cómo algunos de estos se ven reflejados en el proyecto.

1. Single Responsibility principle

En el proyecto las clases fueron creadas para administrar solo una responsabilidad, por ejemplo dentro del proyecto Persistence existen un conjunto de clases con el sufijo "Repository" con la única responsabilidad de administrar las ABM (o CRUD) de sí mismas.

2. Abstract Factory

Otra decisión de diseño relevante fue la de aplicar el patrón **Abstract Factory** para representar los diferentes tipos de alarmas. El alcance de esta solución incluye el alta de alarmas de un nuevo tipo, teniendo que modificar el diseño implementado para la primer entrega. Para esto se crearon dos clases llamadas AuthorAlarm y EntityAlarm que extienden de una clase Alarm, la cual implementa varias de las funcionalidades que las dos primeras tienen en común.



3. Builder

También se decidió crear una clase helper que permitiera centralizar la transformación de objetos. Ej, La clase Helper tiene las funciones necesarias para pasar de un objeto de tipo BusinessLogic a EntityFramework y viceversa.

4. Strategy

Podríamos decir que este diseño cumple con el patrón "**strategy**", porque nuestro programa tiene diferentes comportamientos según el tipo de alarma con el que se decida trabajar, todo esto gracias a la composición, delegación y polimorfismo de nuestro sistema.

5. Singleton

Se implementó Singleton en la clase Helper de nuestro sistema, dado que el único propósito de la misma era la transformación de datos, objetos de tipo BussinesLogic a EntityFramwork, se proporcionó un punto de acceso centralizado.

6. Clean Architecture

Se definió una arquitectura con múltiples capas. Tomando (sin llegar a su nivel de sofisticación) algunos aspectos de la “clean architecture” donde los distintos niveles se van referenciando hacia el interior. Y donde el centro contiene las “Entities” y las diversas interfaces.

7. Diagrama de tablas

Sentiment (ID, Type, Word);

Alarm (ID, CreationDate, PostCount, PostQuantity, Type, Time);

AuthorAlarm (ID, TimeMeasure, Alarm_Id);

EntityAlarm (ID, Alarm_Id, Entity_Id);

Phrase (ID, Word, Type, PostedDate, Grade, Author_Id, Entity_Id);

Author (ID, Username, Name, LastName, Birthdate);

Entity (ID, Name);

8. Cobertura de Pruebas Unitarias

Las siguientes imágenes evidencian la cobertura total del proyecto, superando esta el 91% de los test.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
sgonzalez_02802-NB 2020-06...	94	6.26%	1408	93.74%
businesslogic.dll	39	5.74%	640	94.26%
businesslogic.test.dll	55	6.68%	768	93.32%

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
sgonzalez_02802-NB 2020-06...	384	9.82%	3526	90.18%
businesslogic.dll	132	19.44%	547	80.56%
persistence.dll	170	8.17%	1911	91.83%
Persistence	0	0.00%	22	100.00%
Persistence.Entities	0	0.00%	58	100.00%
Persistence.Migrations	1	25.00%	3	75.00%
Persistence.Repositories	168	9.46%	1608	90.54%
Persistence.Utils	1	0.45%	220	99.55%
persistence.test.dll	82	7.13%	1068	92.87%