

**Universidad ORT Uruguay**  
**Facultad de Ingeniería**

## **Obligatorio 2 - 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**

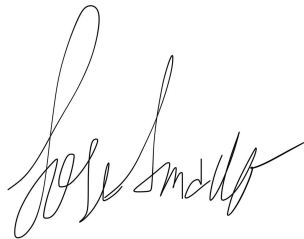
**Docente: Sergio Gutierrez**

**2020**

## Declaración de autoría

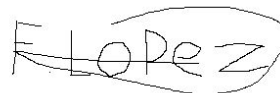
Nosotros, José Pedro Amado y Francisco López, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- 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

25/06/2020



Francisco López

25/06/2020

# Índice

<b>1. Descripción general del trabajo y del sistema</b>	<b>4</b>
<b>2. Descripción y justificación de diseño</b>	<b>5</b>
2.1 Explicación los mecanismos generales, descripción de las principales decisiones de diseño tomadas	5
2.2 Diagramas de interacción para especificar los principales comportamientos del sistema.	12
2.3 Modelo de tablas de la estructura de la base de datos	12
<b>3. Cobertura de pruebas unitarias</b>	<b>13</b>
<b>4. Manual de Instalación</b>	<b>14</b>

## 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](#).

Las primeras tareas asignadas fueron la implementación de las funcionalidades de autor y las nuevas implementaciones de las interfaces que guardan las entidades para que lo hagan contra la base de datos usando Entity Framework. Un integrante llevó a cabo el trabajo de crear de las nuevas clases necesarias para registrar autores y la implementación de todos los cambios necesarios para que una publicación indique quien la escribió. El otro comenzó a implementar las persistencia de las entidades más simples y su respectivo testeo, para luego avanzar hacia entidades con mayor complejidad como herencia. En los casos de herencia, se optó por tener una tabla por tipo, con el objetivo de tener las tablas normalizadas.

La implementación de Entity Framework implicó algunos cambios a las entidades para poder tener sincronizados los IDs de los objetos, por lo cual se optó por cambiarlos a Guid para que sea el dominio quien asigna ese valor y no la base de datos.

Luego se repartieron las tareas de implementar los cambios necesarios a la interfaz gráfica, el nuevo tipo de alarma, el reporte de alarmas y el reporte de autores.

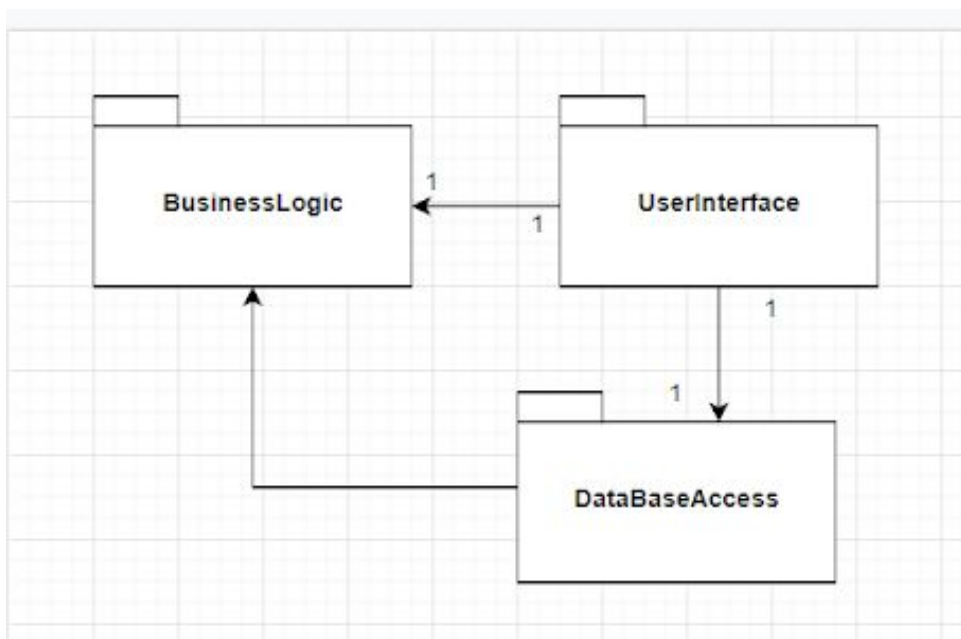
Finalmente se realizaron pruebas exploratorias con el objetivo de encontrar errores, lo cual resultó fructífero ya que se pudieron identificar y corregir varios problemas, entre ellos, problemas al eliminar entidades que tuviesen relaciones con otras entidades ya que el cascade delete de Entity Framework no estaba funcionando ya que al no tener todas las entidades cargadas en la memoria del programa, no se enviaban los comandos para eliminar todas las entidades, sino solamente las cargadas en memoria, lo cual

resultaba en un error de parte de la base y por lo cual fue necesario modificar los métodos de eliminación para eliminar de antemano sus entidades relacionadas.

## 2. Descripción y justificación de diseño

### 2.1 Explicación los mecanismos generales, descripción de las principales decisiones de diseño tomadas

Nuestro proyecto cuenta con tres paquetes, BusinessLogic, DatabaseAccess y UserInterface (figura 1). Como indican sus nombres, BusinessLogic compone toda la lógica del programa, las reglas del negocio, DatabaseAccess se encarga de la conexión del programa con la base de datos y UserInterface corresponde a la interfaz gráfica desarrollada en esta ocasión. Esta separación de backend, frontend y acceso a la base, se debe la probabilidad de cambio de la interfaz y de la base de datos utilizada son 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 o hacer un cambio de base de datos o el método de persistencia.



*Figura 1 - UML de Paquetes de la Solución.*

Se hizo énfasis en cumplir en la mayor capacidad posible los estándares de C Sharp, los estándares de Clean Code, principios de SOLID y GRASP.

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 se conecta con la clase correspondiente del paquete DatabaseAccess, quien a su vez se conecta con la base de datos para almacenar la información del sistema en la misma, y con las demás clases de BusinessLogic para llevar a cabo la acción solicitada.

A nivel del paquete BusinessLogic, se definieron clases para cada tipo de entidad necesaria en el sistema. 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. Esta arquitectura brinda una versatilidad al momento de expandir el sistema. Además, se creó una clase para el sentimiento neutral, el cual únicamente tendrá un objeto que se asignará en caso de no detectarse un sentimiento en una frase. Este objeto tiene un ID único asignado al momento de su creación.

De la misma forma que se creó el sentimiento neutral, se creó una entidad con un ID particular, la cual será referenciada cuando el analizador de publicaciones no pueda encontrar una entidad.

Por tanto, nuestro proyecto cuenta con una clase Sentiment padre y tres clases que lo heredan: PositiveSentiment, NegativeSentiment y NeutralSentiment. La misma lógica rige para las alarmas, donde tenemos una clase Alarm padre y tres clases que la heredan, PositiveAlarm, Negative alarm y AuthorAlarm. En este caso, la alarma de autor contiene atributos adicionales necesarios para el correcto funcionamiento de la misma. Además, existen las clases de entidades Author, Publication, Entity y Relation.

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 DatabaseSaver, logrando invertir la dependencia. Nuestro sistema no depende de las implementaciones, sino de las interfaces y las implementaciones dependen de las interfaces. Estas implementaciones guardan los datos en una base de datos de Microsoft SQL.

La clase SentimentAnalysisContext contiene toda la información de en qué tablas debe guardar cada tipo de objeto, nombres para columnas, entre otros. Se optó por realizar la configuración utilizando Fluent API para tener toda esta configuración centralizada en un lugar. Para los casos de herencia de sentimientos y alarmas, se decidió tener una tabla por tipo para mantener la base de datos normalizada.

Se utilizan objetos denominados Dto para persistir los objetos y no tener que modificar nuestras entidades para poder persistirlas de acuerdo a los requerimientos de Entity Framework. De esta forma, nuestros objetos mantienen su diseño original y no dependen de Entity Framework, que es probable cambie en mayor medida que nuestro dominio. Con el objetivo de poder traducir estos Dtos a nuestros objetos reales y viceversa, se creó la clase ObjectConversion, quien únicamente expone métodos para convertir objetos a Dtos y Dtos a objetos.

Para realizar el testing unitario del acceso a la base de datos, se configuró el proyecto de forma que no utilice la misma base que utiliza el programa de forma normal, para poder eliminar y modificar datos libremente, logrando testear la base sin inconvenientes.

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, una nueva frase y al momento de consultar el reporte de alarmas, en cual se visualizará su estado y demás datos correspondientes según el tipo de alarma. 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.

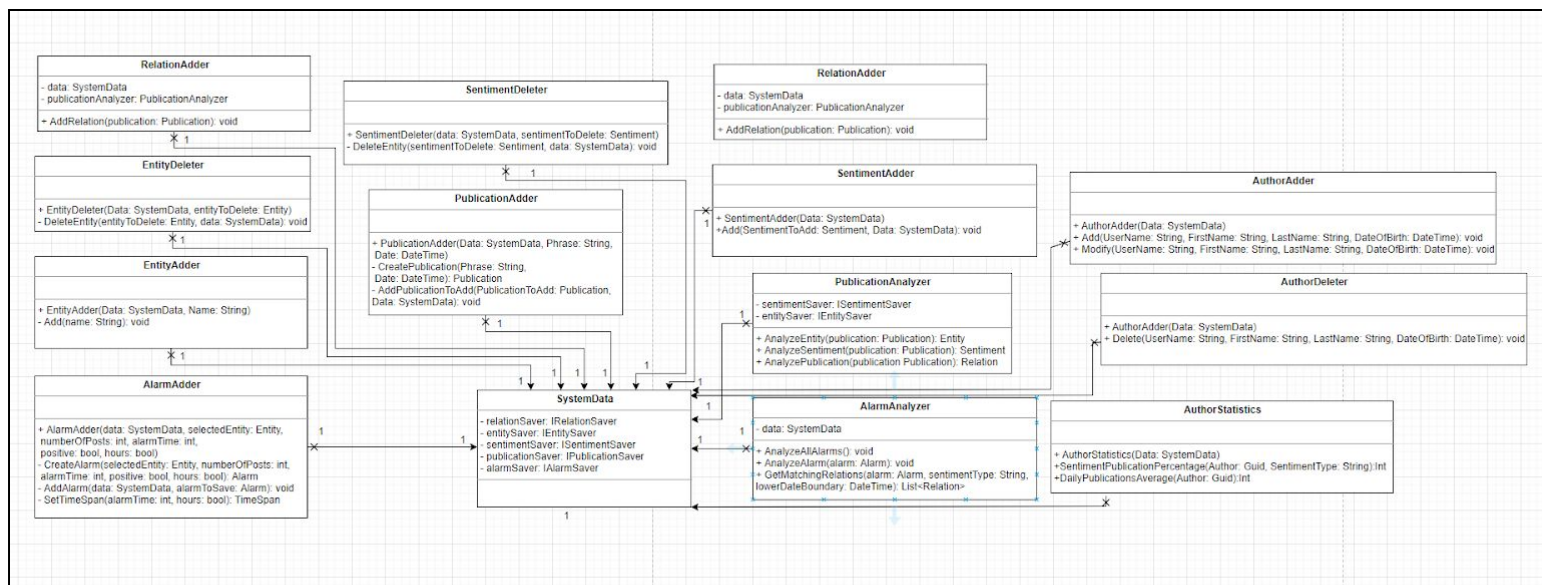


Figura 2 - UML de clase SystemData y asociados.

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 se construyen y luego invocan al método que precisa utilizar la UI para cumplir su funcionalidad. Estas clases están implementadas para solo satisfacer 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 agregue una Entidad.

Las 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.

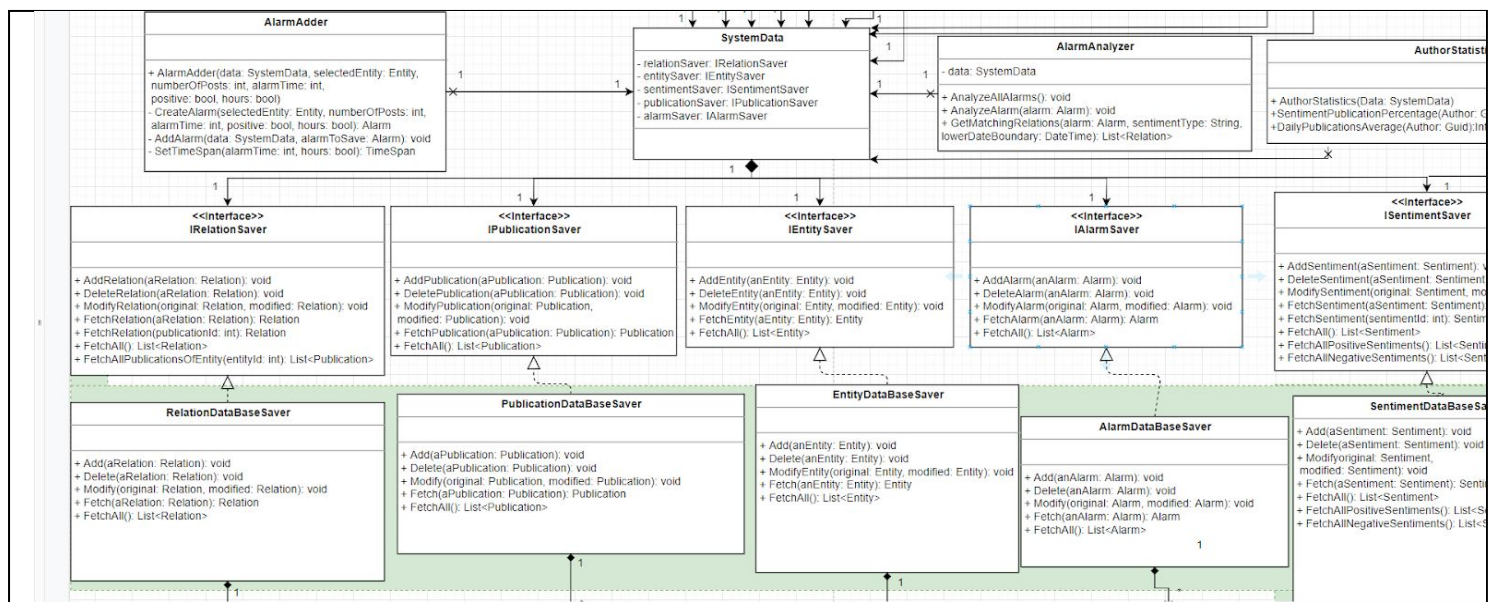


Figura 3 - UML de la relación entre SystemData y algunas de las interfaces.

La clase SystemData tiene la responsabilidad de guardar los datos 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 DatabaseSaver,

quienes utilizan Entity Framework para persistir sus datos en una base de datos. Esta solución permite luego poder cambiar la implementación de guardado sin que tener que cambiar clases como SystemData u otras que usen SystemData, que fue justamente un cambio realizado del obligatorio anterior al actual, se cambiaron las implementaciones por otras nuevas.

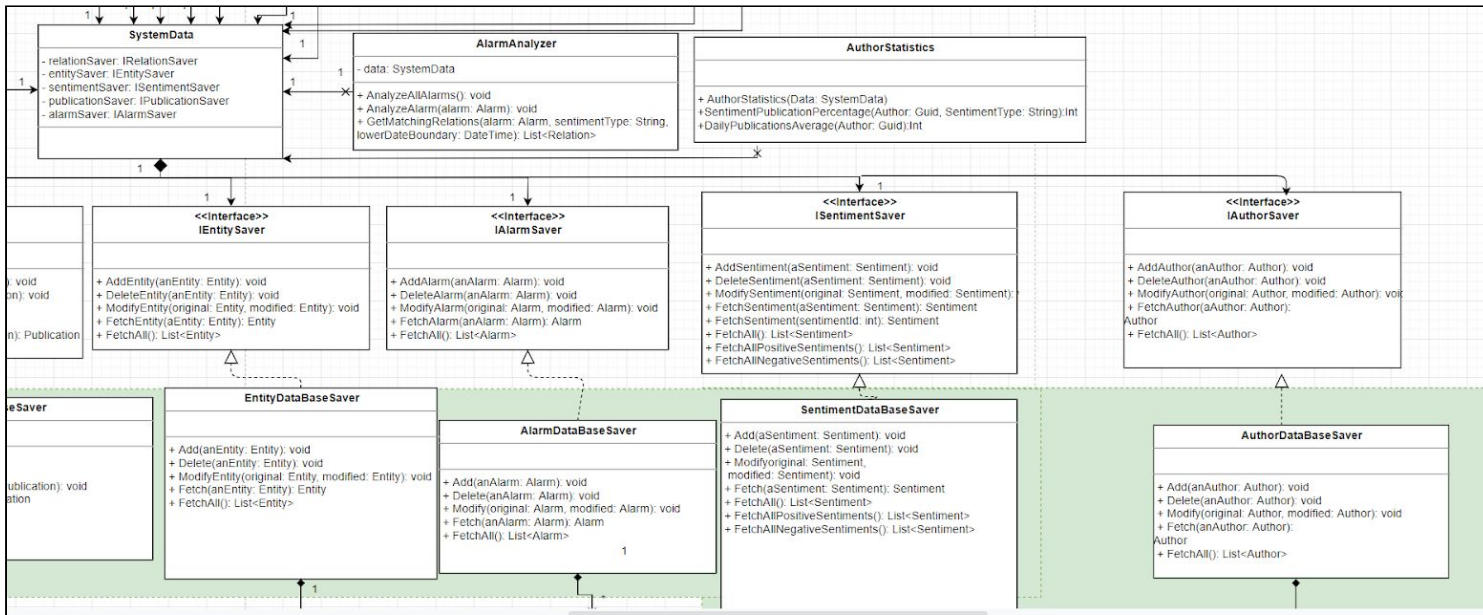
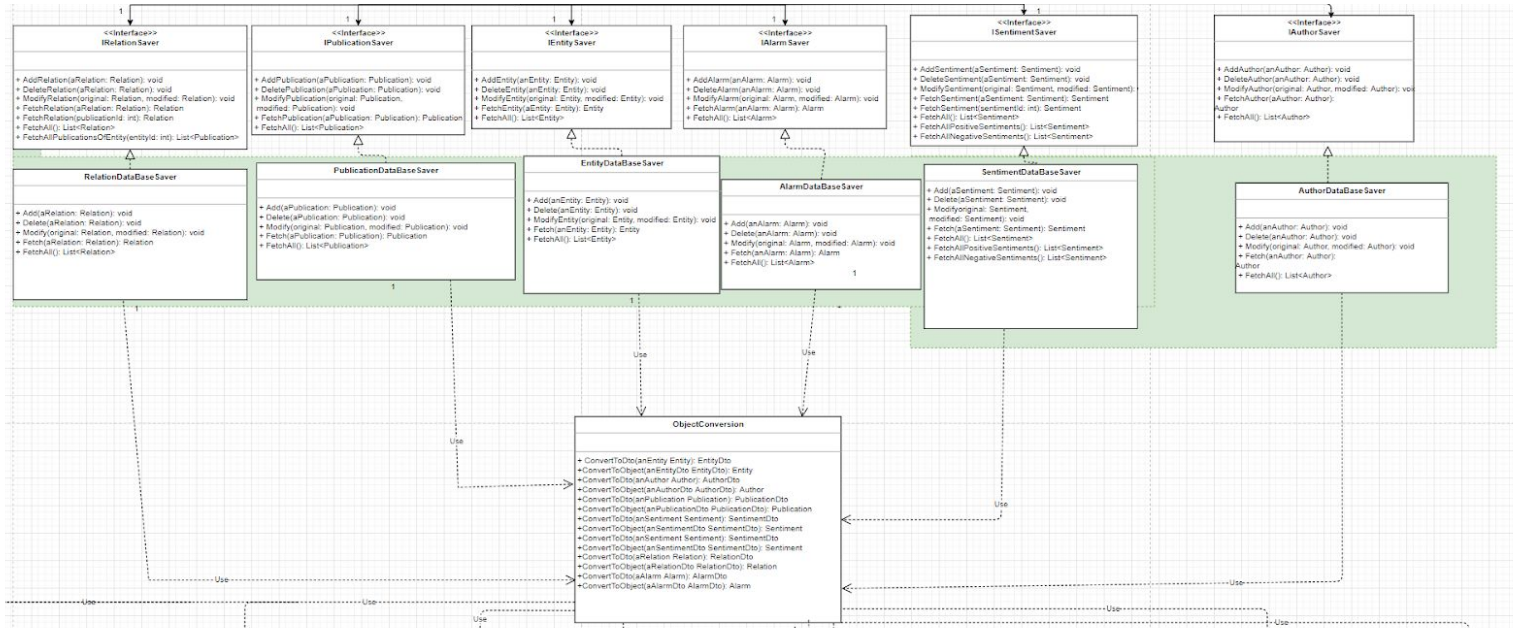
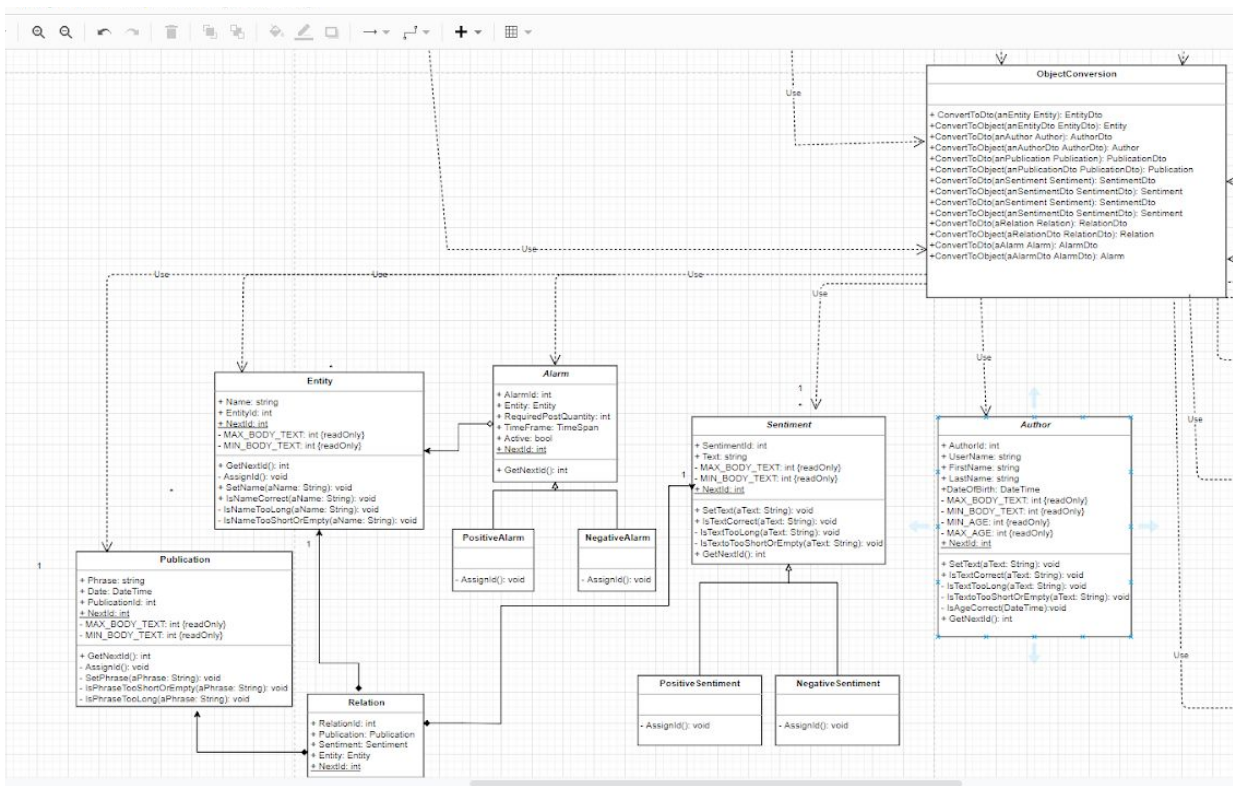


Figura 4 - UML de la relación entre SystemData y el resto de las interfaces.

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 el Conversor y se usa para pasar de DTO a Object y viceversa*



```

classDiagram
    class ObjectConversion {
        +ConvertToDto(anEntity Entity): EntityDto
        +ConvertToObject(anEntityDto EntityDto): Entity
        +ConvertToDto(anAuthor Author): AuthorDto
        +ConvertToObject(anAuthorDto AuthorDto): Author
        +ConvertToDto(anPublication Publication): PublicationDto
        +ConvertToObject(anPublicationDto PublicationDto): Publication
        +ConvertToDto(anSentiment Sentiment): SentimentDto
        +ConvertToObject(anSentimentDto SentimentDto): Sentiment
        +ConvertToDto(aRelation Relation): RelationDto
        +ConvertToObject(aRelationDto RelationDto): Relation
        +ConvertToDto(aAlarm Alarm): AlarmDto
        +ConvertToObject(aAlarmDto AlarmDto): Alarm
    }

    class Author {
        +AuthorId: int
        +UserName: string
        +FirstName: string
        +LastName: string
        +DateOfBirth: DateTime
        +MAX_BODY_TEXT: int (readOnly)
        +MIN_AGE: int (readOnly)
        +MAX_AGE: int (readOnly)
        +NextId: int
        +SetText(aText: String): void
        +IsTextCorrect(aText: String): void
        +IsTextTooLong(aText: String): void
        +IsTextTooShortOrEmpty(aText: String): void
        +IsAgeCorrect(aDate: Time): void
        +GetNextId(): int
    }

    class AuthorDTO {
        +AuthorDtoId: Guid
        +UserName: String
        +FirstName: String
        +LastName: String
        +DateOfBirth: DateTime
    }

    class EntityDTO {
        +EntityDtoId: Guid
        +Name: String
    }

    class PublicationDTO {
        +PublicationDtoId: Guid
        +Phrase: String
        +Date: DateTime
    }

    class SentimentDTO {
        +SentimentDtoId: Guid
        +Name: String
    }

    class RelationDTO {
        +RelationDtoId: Guid
    }

    ObjectConversion --> Author : Use
    ObjectConversion --> AuthorDTO : Use
    ObjectConversion --> EntityDTO : Use
    ObjectConversion --> PublicationDTO : Use
    ObjectConversion --> SentimentDTO : Use
    ObjectConversion --> RelationDTO : Use
    Author --> AuthorDTO : Use
    Author --> EntityDTO : Use
    Author --> PublicationDTO : Use
    Author --> SentimentDTO : Use
    Author --> RelationDTO : Use
    AuthorDTO --> EntityDTO : Use
    AuthorDTO --> PublicationDTO : Use
    AuthorDTO --> SentimentDTO : Use
    AuthorDTO --> RelationDTO : Use
    EntityDTO --> PublicationDTO : Use
    EntityDTO --> SentimentDTO : Use
    EntityDTO --> RelationDTO : Use
    PublicationDTO --> SentimentDTO : Use
    PublicationDTO --> RelationDTO : Use
    SentimentDTO --> RelationDTO : Use
    
```

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 DatabaseSaver se encargan de guardarlas sin que se enteren las clases de objetos. Estas clases DatabaseSaver implementan las interfaces Saver para así poder cambiar el método de guardado si es necesario y no afectar a ninguna clase con este cambio.

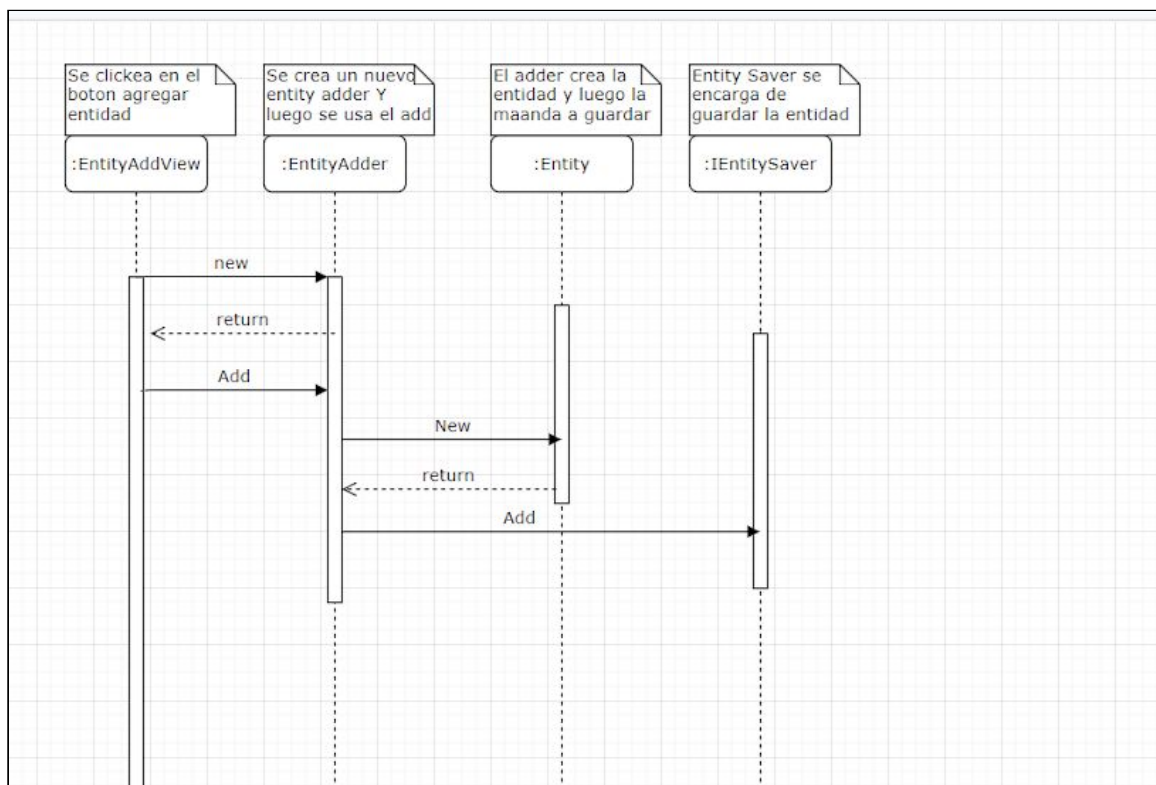
En cuanto a las excepciones, se utilizan clases propias para manejar excepciones de existencia, creación de objetos, largo de textos, controles de edad, controles de cantidad, entre otras. No consideramos fuera necesario incluirlas en los diagramas dado que



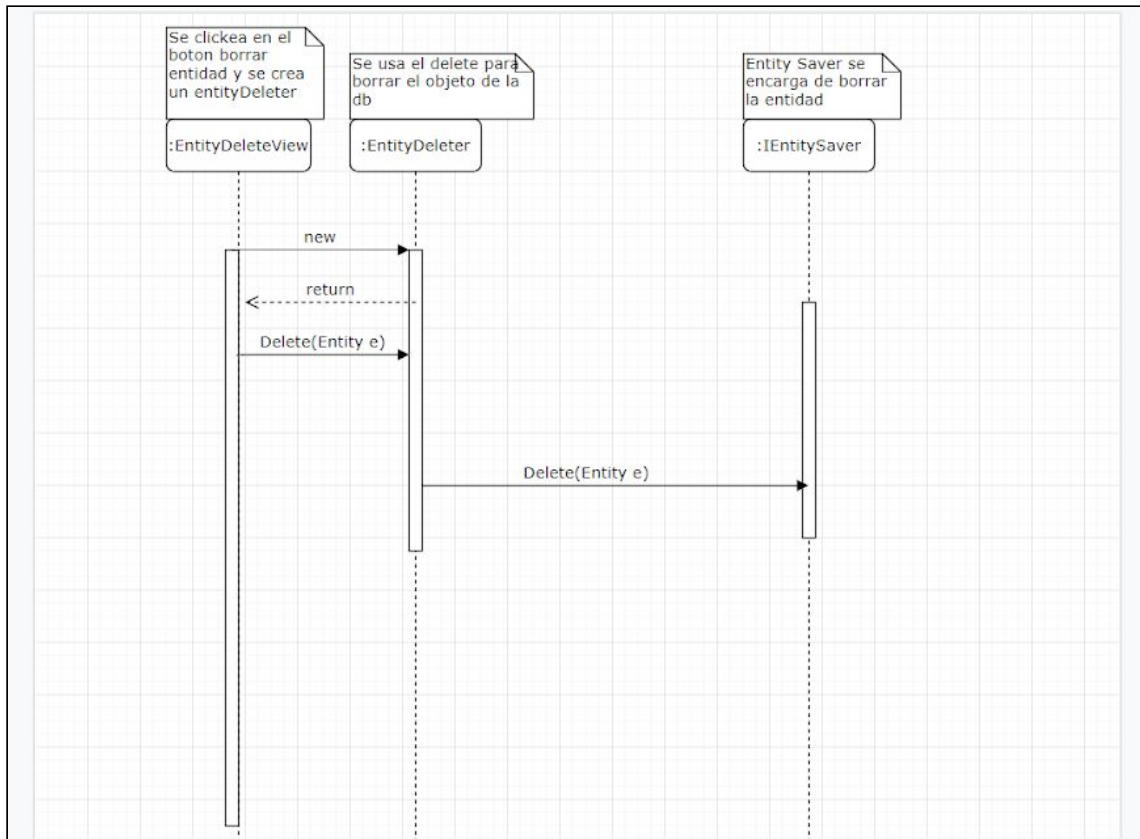
entendemos no aportan valor al momento de analizar el mismo. Los mensajes dados por estas son pasados a la UI para que el usuario pueda percatarse de los errores y solucionarlos.

## 2.2 Diagramas de interacción para especificar los principales comportamientos del sistema.

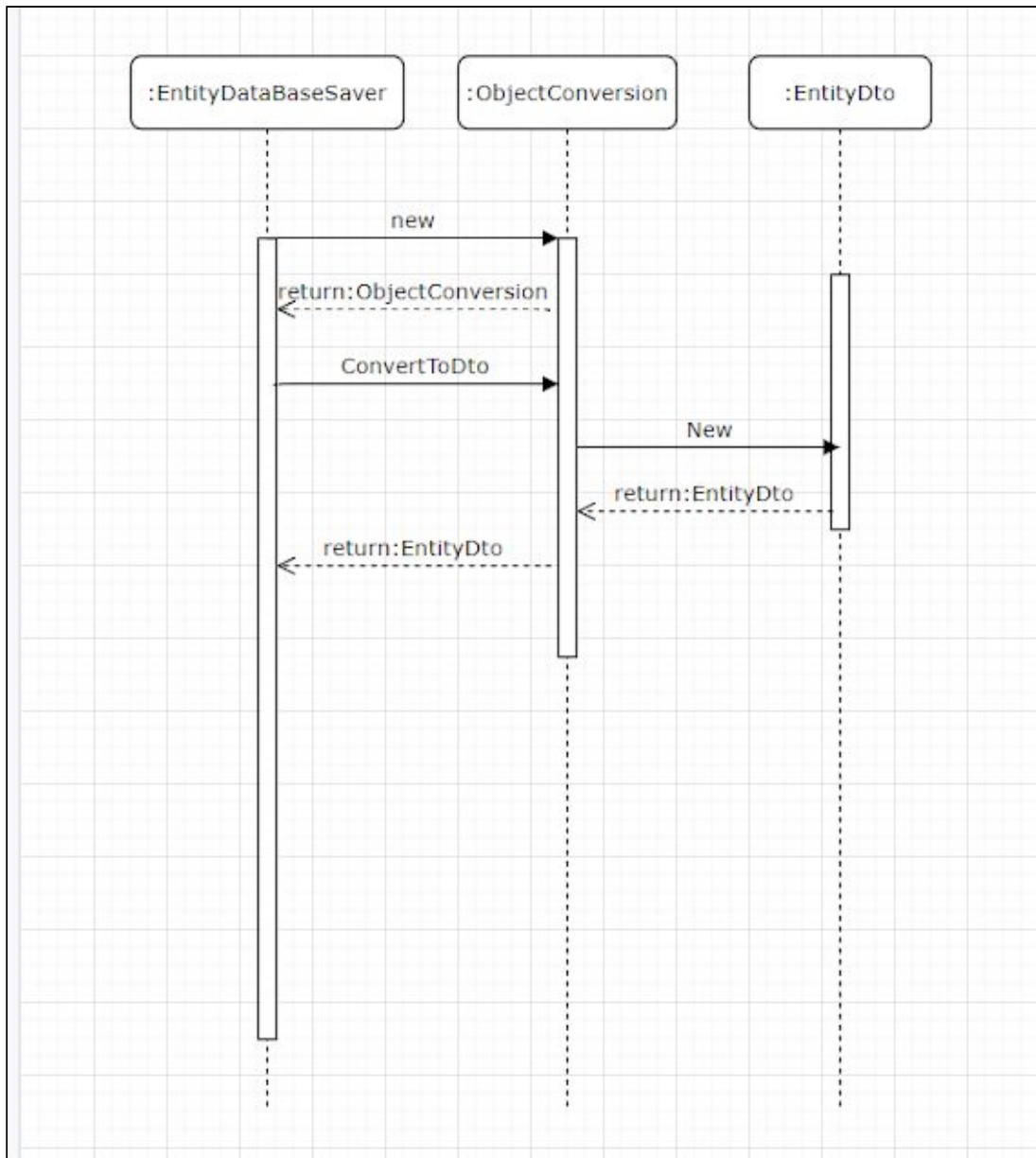
El primer diagrama de interacción muestra cómo funciona la acción de crear una entidad. Primero se da al botón de agregar entidad, se crea el EntityAdder, luego se le pide al EntityAdder que agregue la entidad, pasándole los datos necesarios, se crea la entidad y la envía a DataAccess para que la guarde en la base de datos.



Este diagrama de interacción detalla cómo borrar una Entidad, Es un proceso similar al de agregar una entidad, donde se crea un EntityDeleter que recibe una entidad y se encarga de enviarla a DataAccess para que elimine la entidad.



El último diagrama muestra cómo DataAccess usa la clase ObjectConversion para pasar de DTO a objetos comunes. El proceso es el siguiente, Se crea un Objeto ObjectConversion, se le pasa el objeto entity en el método ConvertObject y se devuelve un EntityDto que podrá ser usado para guardar en la base de datos. Esto también se hace en el sentido opuesto para traer objetos de la base de datos.



## 2.3 Modelo de tablas de la estructura de la base de datos

### 3. Cobertura de pruebas unitarias

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

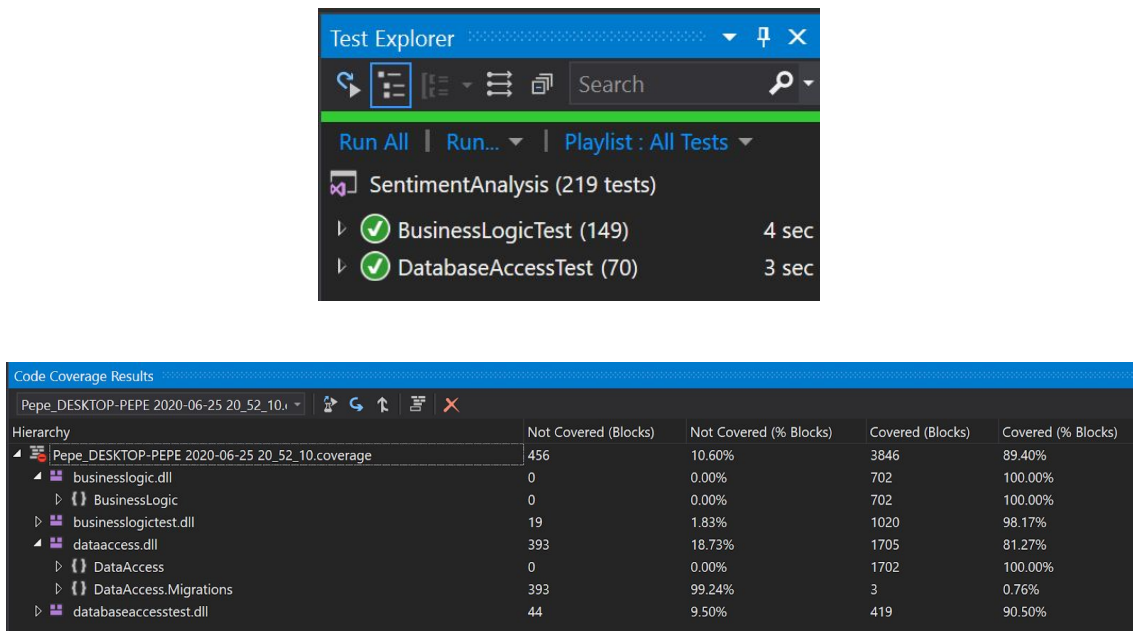


Figura 6 - Captura del resultado del análisis de cobertura de pruebas unitarias.



## 4. Manual de Instalación

### Requerimientos de Software:

- Windows 7 o superior.
- Microsoft SQL Express 2014 con nombre de instancia SQLEXPRESS con autenticación de Windows.
- Microsoft SQL Management Studio.

### Procedimiento:

1. Copiar el programa encontrado en la carpeta Release a la ruta donde se desea almacenar el mismo.
2. Crear un acceso directo al archivo SentimentAnalysis.exe en la carpeta C:\ProgramData\Microsoft\Windows\Start Menu\Programs.
3. Si desea cargar los datos de prueba realizar los siguientes pasos:
  - a. Abrir Microsoft SQL Management Studio
  - b. Conectarse a la instancia SQLEXPRESS
  - c. Asegurarse que no exista la base llamada SentimentAnalysis
  - d. Darle click derecho en Databases
  - e. Seleccionar la opción Restore Database
  - f. Seleccionar Device, buscar el archivo de respaldo FullDatabaseBackup.bak y seleccionar Ok
  - g. Volver a darle Ok para restaurar la base
4. Buscar el programa en el menú de inicio y ejecutarlo.