Universidad ORT Uruguay

Facultad de Ingeniería

Descripción del diseño para primer obligatorio de Diseño de Aplicaciones 2

Emiliano Yozzi - 230710

Franco Thomasset - 239611

Tutores:

Juan Irabedra,Santiago Tonarelli, Francisco Bouza

2022

Enlace al repositorio:ORT-DA2/239611-230710 (github.com)

Declaración de autoría:

Nosotros, Franco Thomasset y Emiliano Yozzi, 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 2;

• 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

Abstract:

El objetivo de este trabajo es el de representar un Sistema de blogs que permita la creación y gestión de contenido escrito en forma de artículos. La plataforma permite a los usuarios registrarse, crear perfiles y publicar artículos (de manera privada o pública) de forma cronológica que pueden ser comentados por otros usuarios.

Además el sistema ofrece funcionalidades como notificaciones de nuevos comentarios.

Se pueden crear, eliminar y modificar usuarios con roles de Blogger o Administrador.

A alto nivel el sistema está compuesto por una API Rest que permite interacción con nuestro Backend y Bases de Datos relacional.

Utilizamos las siguientes tecnologías:

- Microsoft Visual Studio 2022 Enterprise
- Microsoft SQL Server Express 2017
- Astah UML
- Entity Framework Core 6
- Postman
- ASP.NET Core 6.0

Índice

Descripción general del trabajo

Nuestra solución consta de tres capas:BlogApplication, BlogServices y BlogDataAccess.
BlogApplication es nuestro proyecto Web API, contiene tres carpetas importantes:

- Controllers: controladores responsables de gestionar y procesar las solicitudes HTTP del cliente para pasarlas a nuestra capa de servicios y retornar una respuesta.
- Models: Modelos de In y Out utilizados para recibir y enviar datos en las solicitudes y respuestas HTTP
- Filters: Para el manejo de excepciones que puedan ocurrir

BlogServices: Contiene los servicios para las funcionalidades de nuestro sistema.

BlogDataAccess: Contiene los repositorios, se encarga de agregar, modificar o buscar datos en nuestra Base de datos

Cuando se realiza un request con Postman para realizar una operación sobre un recurso, se ejecuta el método correspondiente en el controlador adecuado, este le pasa la información recibida a BlogServices donde se procesa y se pasa a BlogDataAccess donde se provee, modifica o se agrega información según el tipo de petición ya que es la capa que mantiene conexión con la base de datos.

Además de estos, hay 3 paquetes más, Domain, Exceptions y ServiceFactory:

- **Domain**: Contiene las clases que utilizamos para representar los objetos de dominio con los que trabaja nuestro sistema.
- **Exceptions**: Contiene Exceptions personalizadas que usamos a lo largo del sistema.
- **ServiceFactory**: Contiene una clase ServiceExtension utilizada para configurar y registrar los servicios de la aplicación utilizando inyección de dependencias.

Errores conocidos

Funcionalidades varias:

Modificar usuario y comentar no funcionan correctamente y no tenemos el tiempo suficiente para arreglarlo.

Ausencia de código de estado 201:

En los action methods correspondientes a verbos POST y PUT no se retorna un código 201, debido a que la implementación requería mockear demasiadas cosas en los test y el tiempo para esta entrega escasea.

Vista de componentes

Diagrama de paquetes:

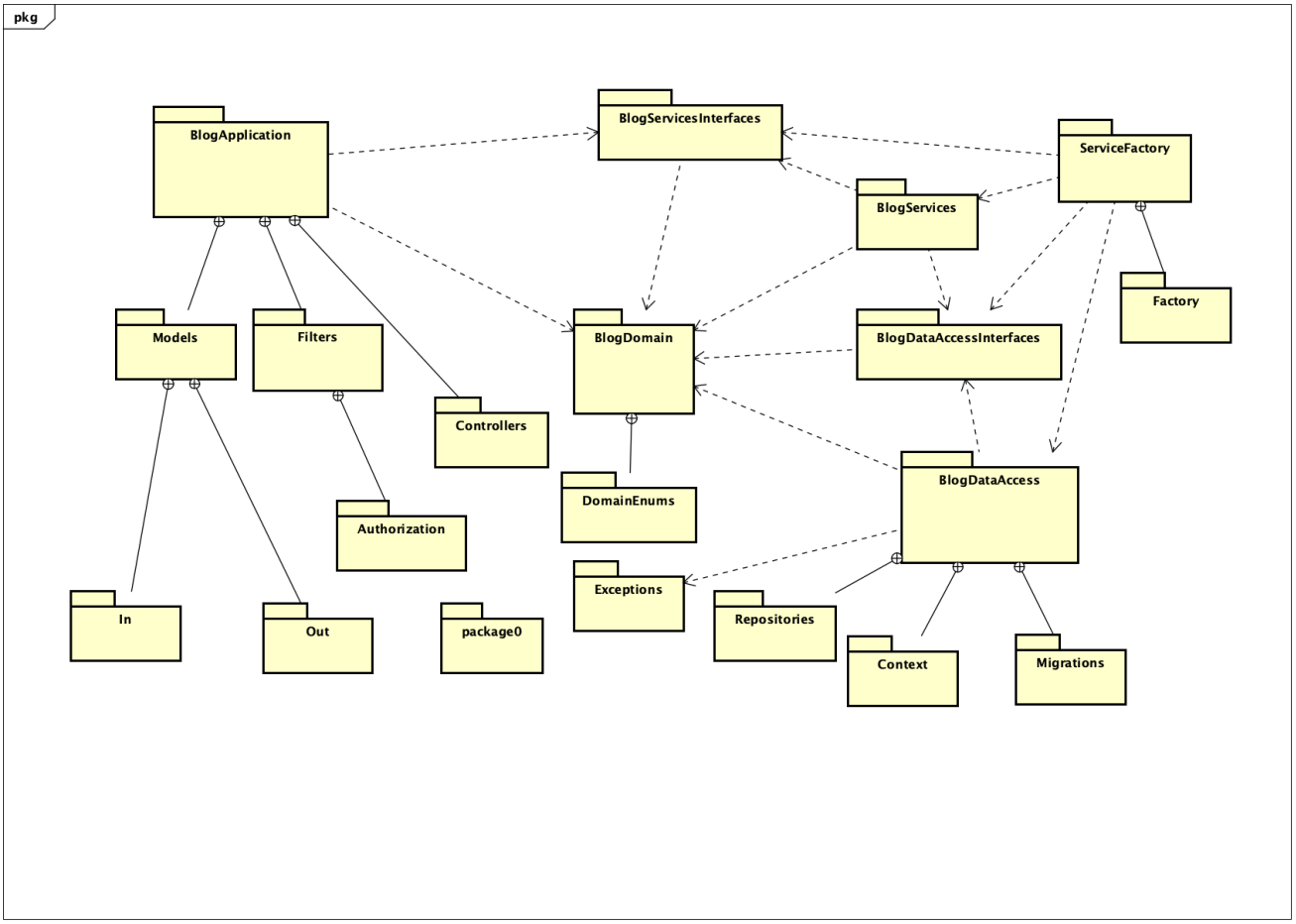
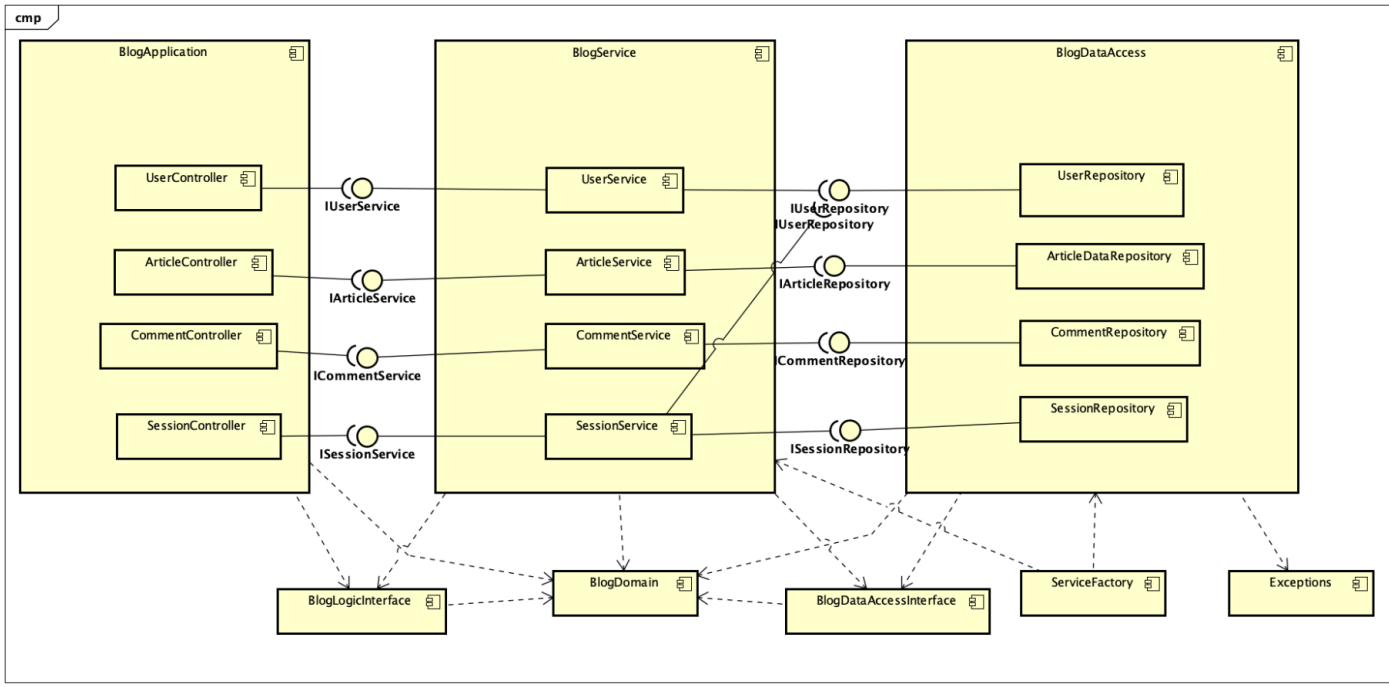


Diagrama de componentes



Los diferentes componentes se comunican a través de Interfaces. Esto nos proporciona:

- Separación de responsabilidades: Cada componente se ocupa de una única función o conjunto de funciones, lo que facilita la comprensión y el mantenimiento del código.

- Bajo acoplamiento: La conexión de componentes a través de interfaces minimiza el acoplamiento entre ellos. Esto significa que cada componente puede evolucionar y modificarse de forma independiente, sin afectar a los demás.

- Facilita la inyección de dependencias: El uso de interfaces permite implementar fácilmente la inyección de dependencias. En lugar de crear instancias de dependencias dentro de un componente, estas se suministran desde el exterior, lo que hace que el código sea más flexible y fácil de probar.

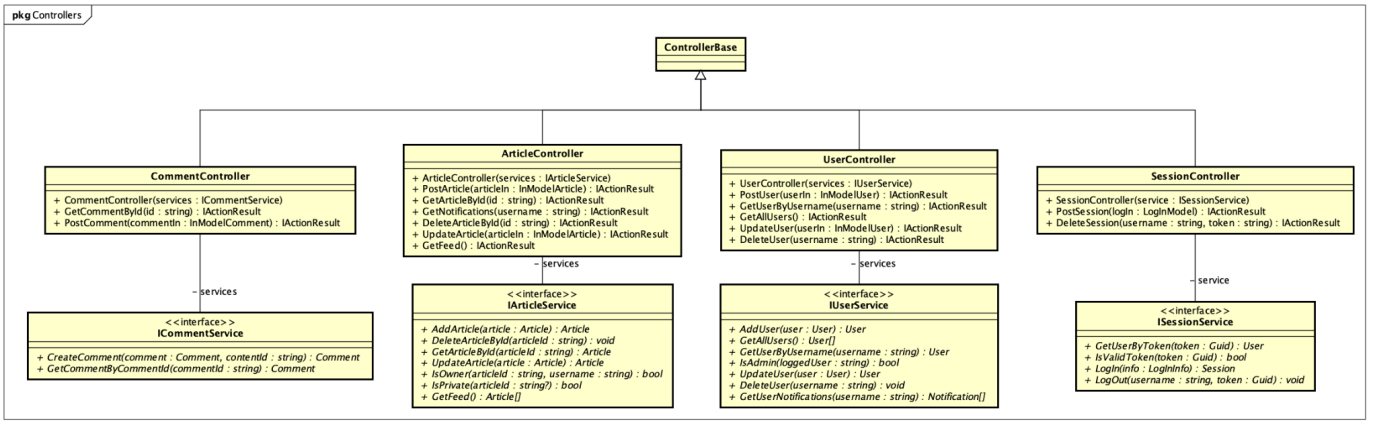
- Facilita la reutilización de código: Al aislar las funcionalidades en componentes separados e interconectarlos a través de interfaces, se fomenta la reutilización de código.

- Facilita la prueba unitaria: La utilización de interfaces para interconectar componentes facilita la creación de pruebas unitarias, ya que permite la sustitución de dependencias reales por objetos simulados (mocks). Esto simplifica el proceso de prueba y asegura que cada componente funcione correctamente.

Vista de Diseño

Dejamos los diagramas en una carpeta aparte para facilitar la legibilidad.

Controllers:



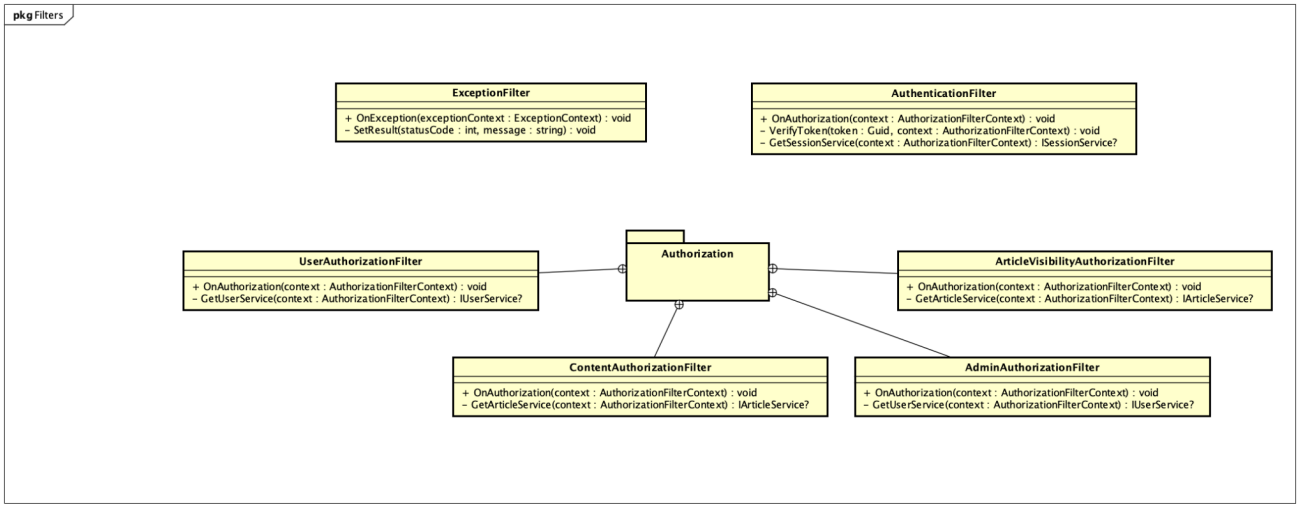
Paquete de controllers dentro del namespace BlogApplication, con los métodos para recibir las peticiones del usuario

Filters:

Hicimos uso de filtros con el objetivo de mejorar la calidad del código en los controllers.
Para evitar escribir try y varios catch en cada controller, creamos el **ExceptionFilter**. Es una clase que implementa IExceptionFilter, generando un método que se ejecuta cada vez que una excepción es lanzada. En ese método escribimos los catch que consideramos necesarios para transformar la excepción en un mensaje útil como respuesta a la request.
Luego creamos el filtro de autenticación “**AuthenticationFilter**”, este filtro se encarga de verificar que el recibido en el header sea válido.
Después están los filtros de autorización:

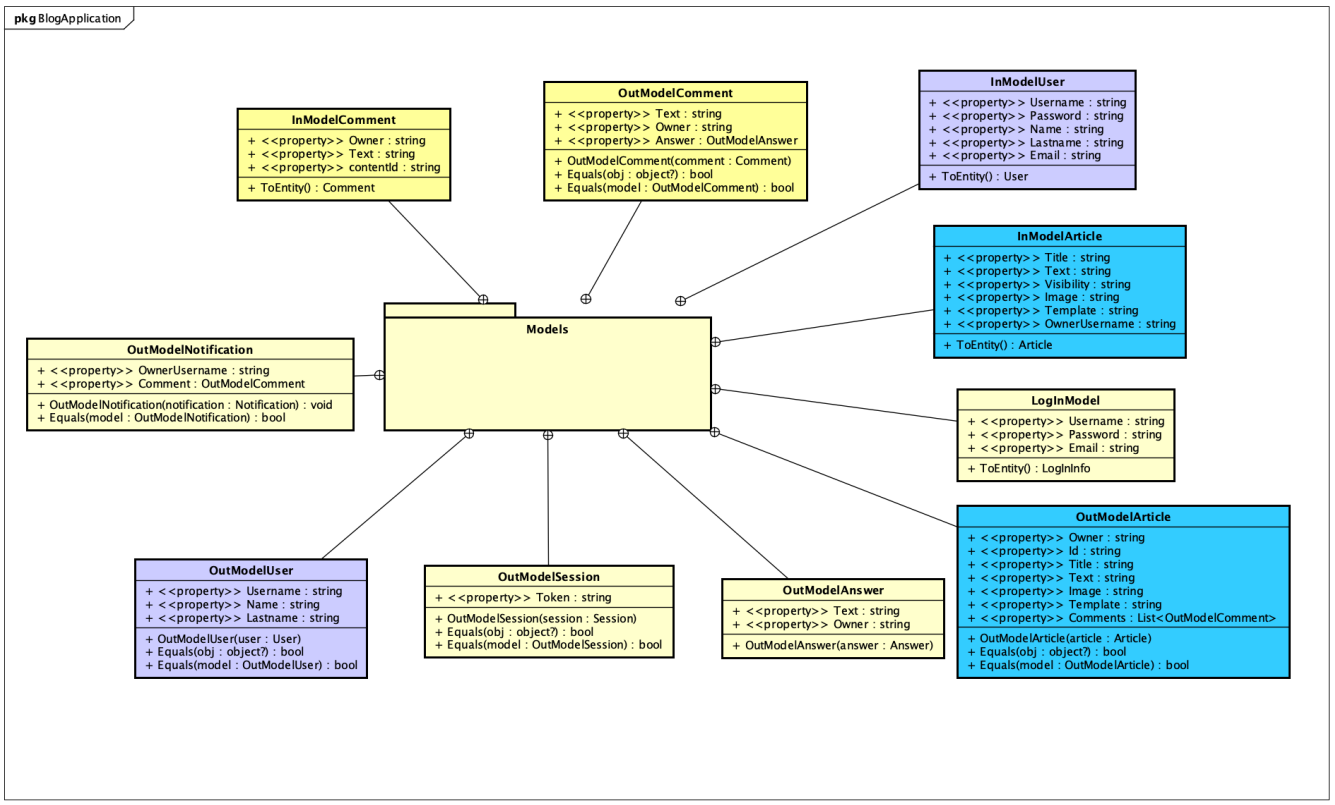
- **AdminAuthorizationFilter**: Verifica que el token pertenezca a un administrador.
- **ArticleVisibilityAuthorizationFilter**: Si el artículo es privado, verifica que el token pertenezca al dueño.

- **ContentAuthorizationFilter**: Verifica que el token pertenezca al dueño del articulo.
- **UserAuthorizationFilter**: Verifica que el token pertenezca al usuario o a un administrador.



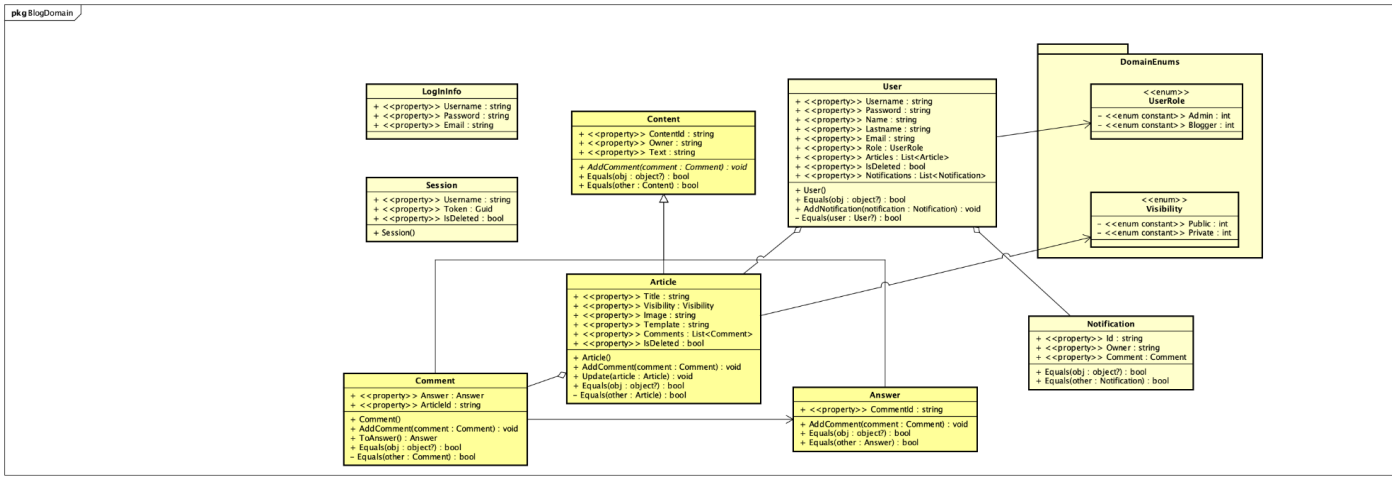
Models:

Objetos utilizados para controlar la entrada y salida de información para nuestros controllers, sin exponer la información completa de los objetos de nuestro dominio.



BlogDomain

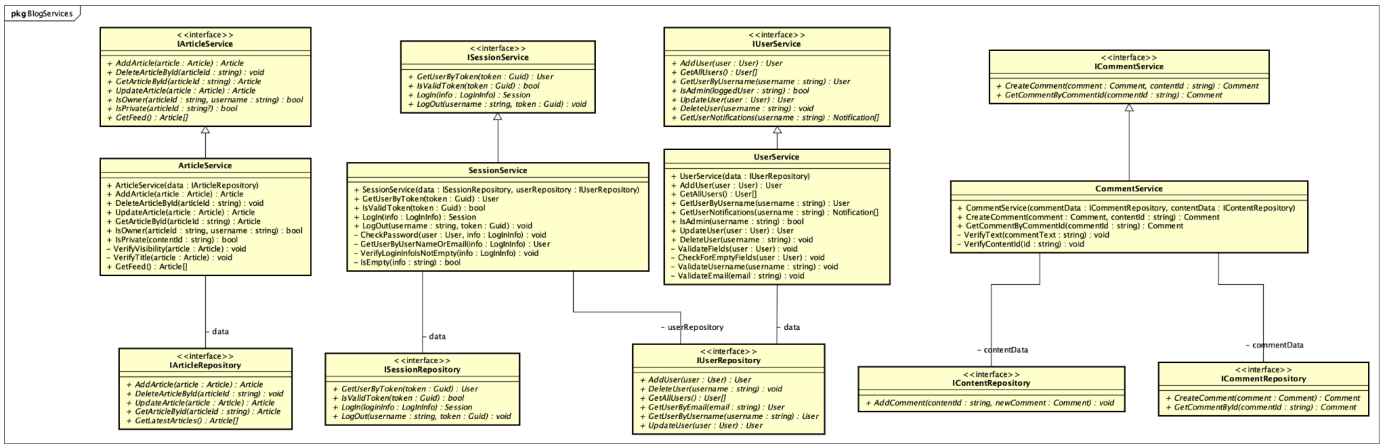
El dominio de nuestra aplicación tiene la información de cada objeto y sus relaciones entre sí. Es importante destacar que tenemos muy pocos métodos ya que la mayor parte de la funcionalidad se encuentra en BlogServices.



Para la asignación de id para los contenidos (articulos y comentarios) usamos un método llamado GenerateId en ContentRepository, que retorna una variable estática privada la cual incrementa cada vez que el método es invocado. Esta estrategia tiene un defecto, si el sistema se reinicia, esa variable también.

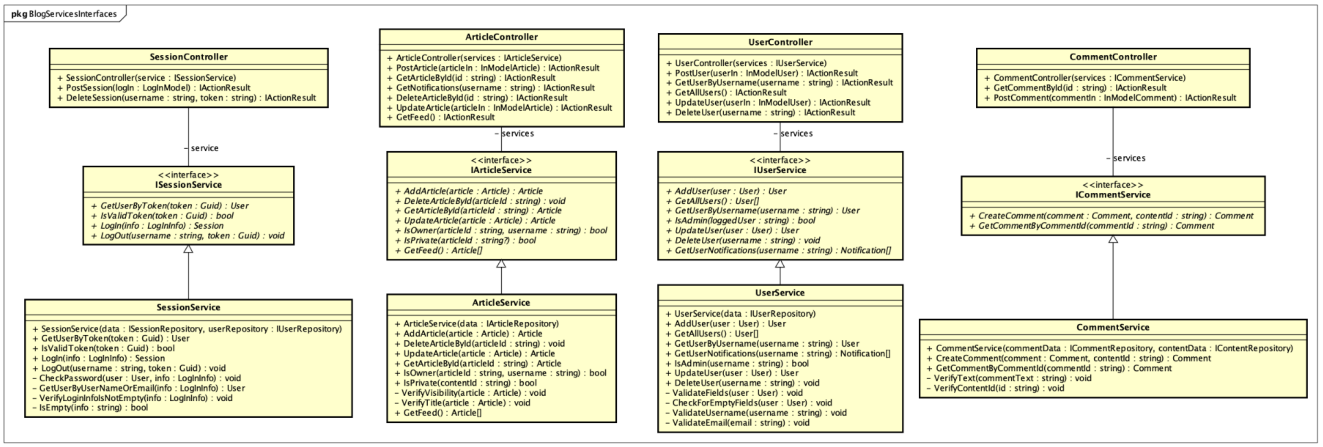
BlogServices

BlogServices contiene la mayor parte de la lógica de nuestra aplicación. Cada Service implementa un contrato dado por una interfaz por un lado y por otro requieren una interfaz que se encargue del manejo de datos.



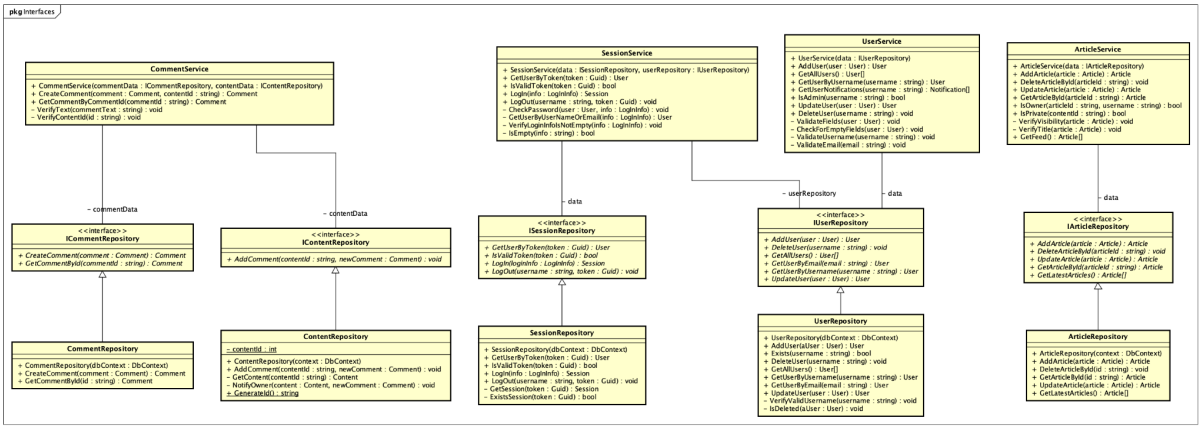
BlogServicesInterface

Contiene todos los contratos que necesite cada controller con su respectiva implementación



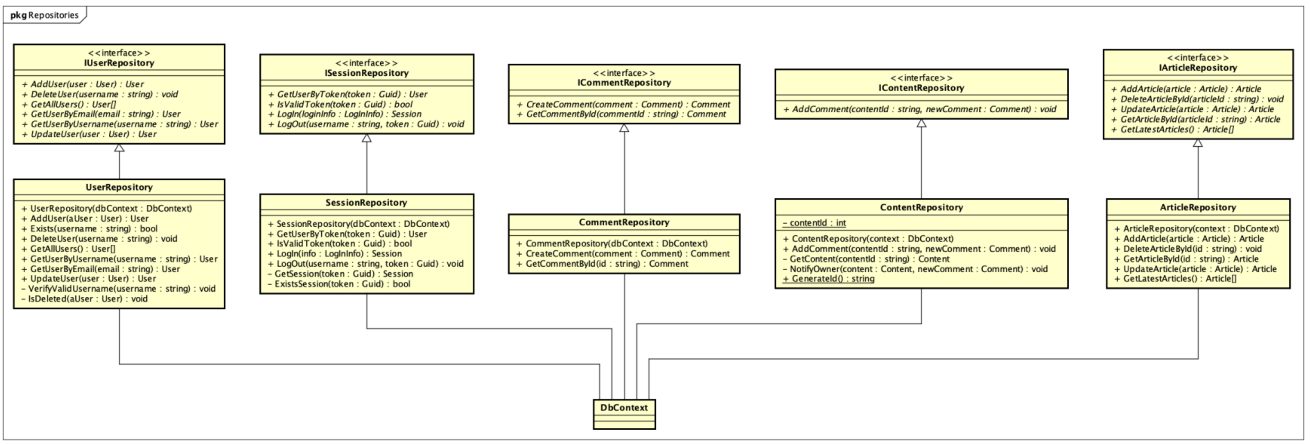
DataAccessInterface

Capa encargada de desacoplar BlogService de la implementación de la base de datos, esto, sumando a la flexibilidad de Entity Framework Core nos permitiría cambiar de ORM



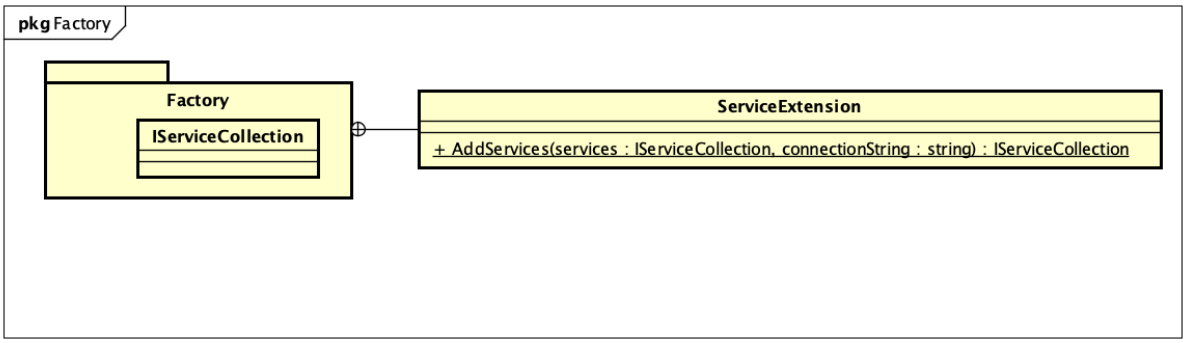
DataAccess

Tiene las implementaciones de cada interfaz repository y será donde se agregarán, modificarán y se consultarán datos de nuestra base de datos.



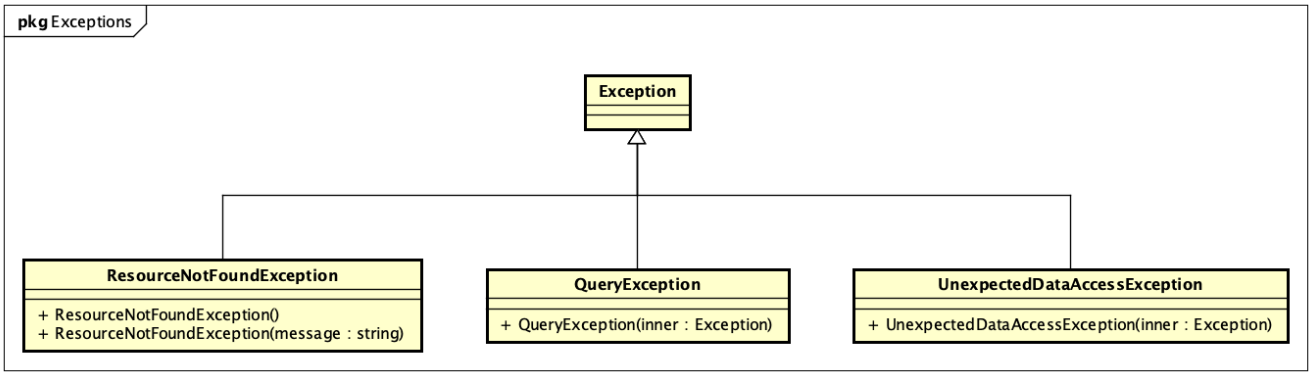
ServiceFactory

Paquete con la ServiceExtension encargado de asociar



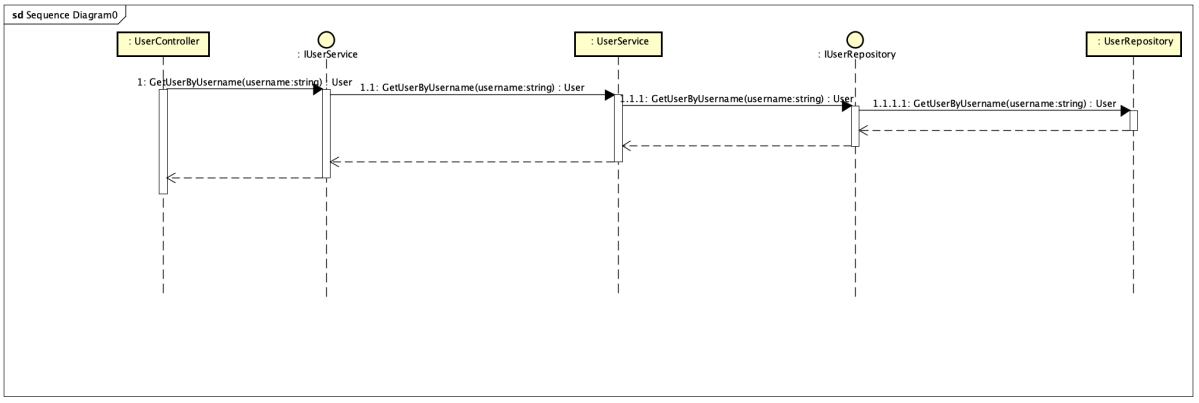
Exceptions

Contiene dos excepciones custom para cuando no encontramos elementos requeridos en la base de datos QueryException y UnexpectedDataAccessException



Vista de Procesos

Diagrama de secuencia de GetUserById



Modelado de tablas

Users

Username	Password	Name	Lastname	Email	Role	IsDeleted

Articles

ContentId	Title	Visibility	Image	Template	IsDeleted	Username	Owner	Text

Sessions

Token	Username	IsDeleted

Notifications

Id	Owner	CommentContentId	Username

Comments

ContentId	ArticleId	Owner	Text

Answers

ContentId	CommentId	Owner	Text