Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 1 -Diseño de aplicaciones 2 - Descripción del diseño

Nicolas Hernandez – 229992
Francisco Aguilar – 230143

2023

Índice

Abstract

En este documento se puede encontrar la descripción general de la solución implementada para el obligatorio de diseño de aplicaciones 2, esa descripción es acompañada por diferentes diagramas utilizados para la fácil comprensión de la solución y algunos flujos de la misma. Aquí también se encuentra la justificación del diseño y algunos de los errores conocidos.

Descripción general

La solución consta de un servicio backend basado en un sistema de blog en el cual se permita el registro, logueo, deslogueo y modificación de usuarios. Un sistema de administración de usuarios para los usuarios con roles de Admin en donde el mismo pueda crear, modificar y eliminar usuarios del sistema y también ver los usuarios más activos entre 2 fechas definidas. También se implementó un sistema de creación , modificación y eliminación de artículos por parte de los usuarios logueados con el rol de blogger, también se permite visualizar por otros usuarios los diferentes artículos públicos y comentar en los mismos, donde el creador del artículo también va a poder responder dichos comentarios. Los creadores de artículos también son capaces de colocar sus artículos en privados para que solo ellos sean capaces de visualizarlos. Otra característica del sistema es que se pueden buscar los artículos a partir de un texto.

Los usuarios no logueados son solamente capaces de ver los 10 últimos artículos públicos creados en el sistema y de registrarse.

Errores conocidos

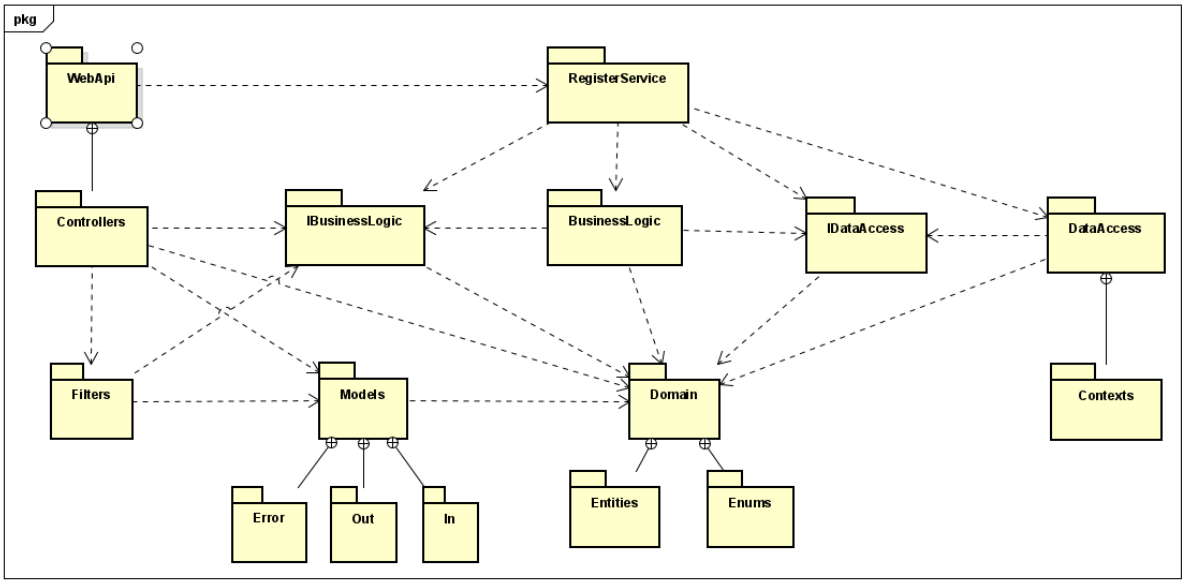
1. Al responder un comentario, en algunas ocasiones el usuario creador del comentario queda vacío, por lo que al obtener un artículo con dicho comentario dentro, se genera un error.

2. Los comentarios pueden ser respondidos por cualquier usuario del sistema con el rol blogger y logueado, no solamente por el usuario creador del artículo
3. Al borrar un comentario se obtiene un error
4. Al buscar un artículo por texto, cuando el texto se encuentra solamente en el título, no se encuentra el artículo

Se va a buscará corregir todos estos errores para la segunda instancia del obligatorio

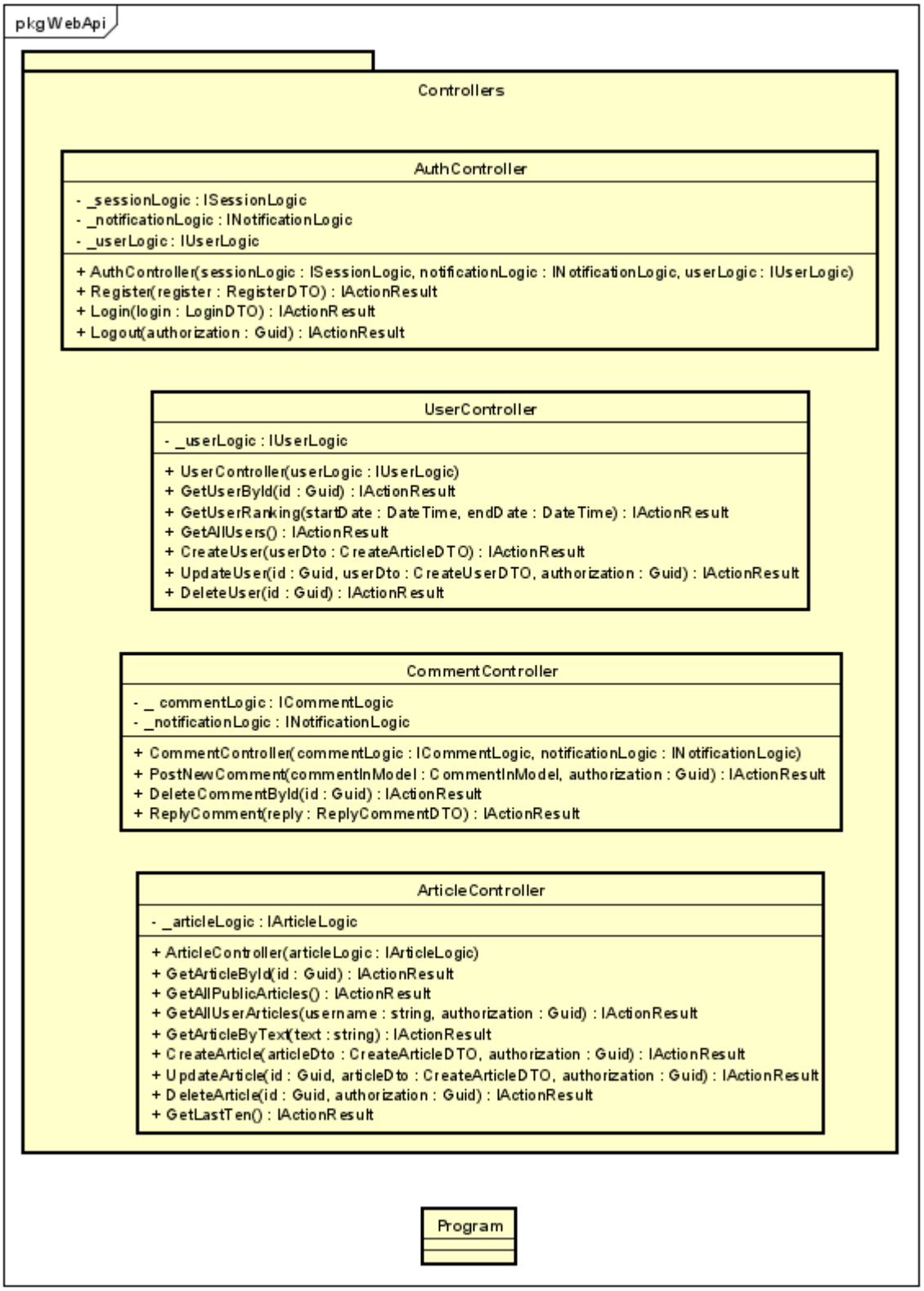
Diagrama general de paquetes

Nuestra solución se dividió en 12 paquetes los cuales cuentan con diferentes responsabilidades.



Blog.WebApi

Este paquete es el punto de ingreso a nuestro backend y es donde se encuentran todos los controladores con los diferentes endpoints.

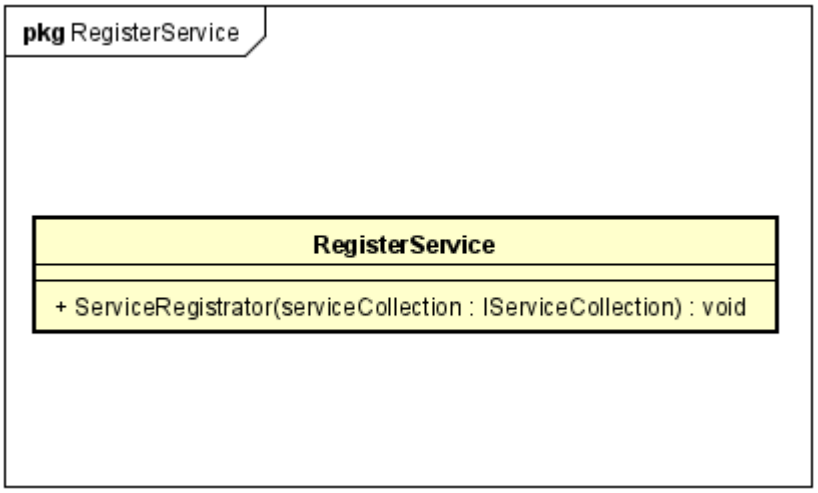


Blog.Tests

Este paquete contiene los test unitarios de WebApi y de las entidades del dominio

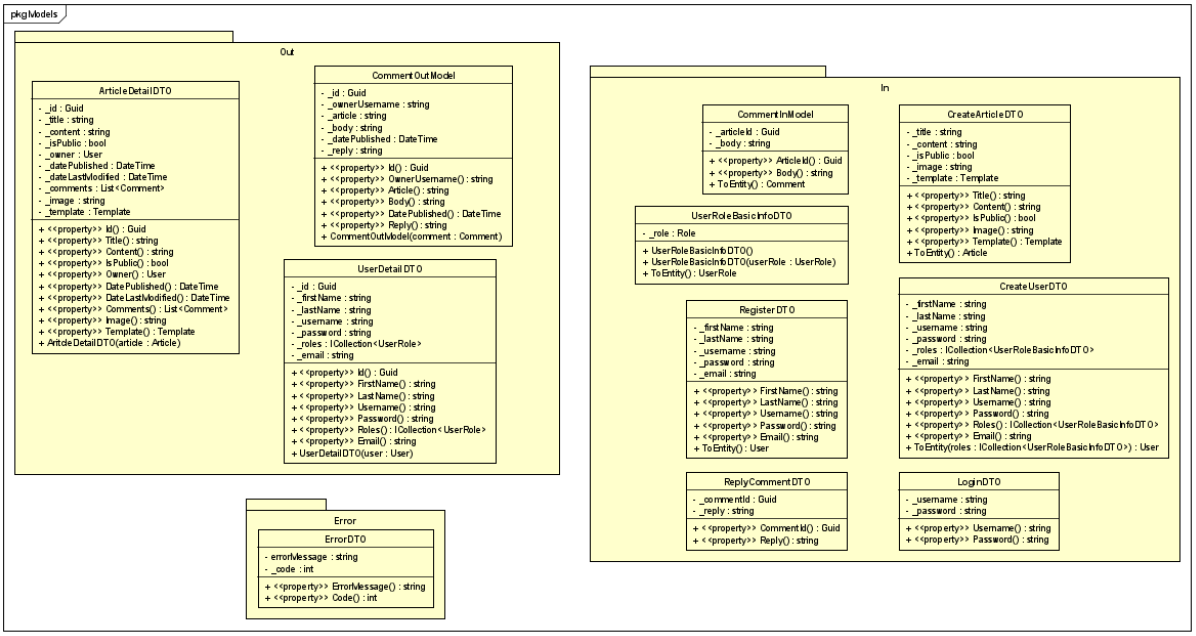
Blog.RegisterService

Este paquete es donde se realizan todas las inyecciones de dependencia, también está allí el registro del servicio del DbContext y de los filtros de autorización y de roles. Este paquete lo creamos para sacar esa responsabilidad del paquete de WebApi ya que anteriormente se realizaban en la clase program.



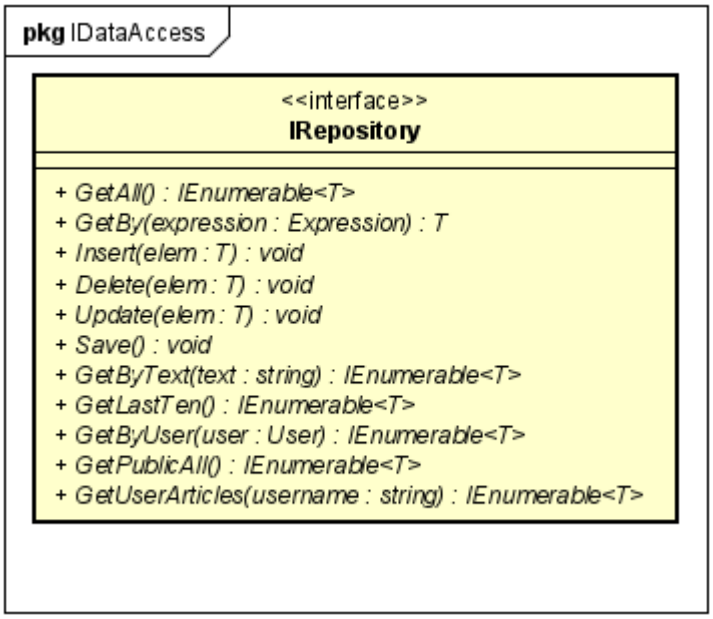
Blog.Models

Este paquete contiene todos los dtos que utilizamos en la aplicación, osea los que utilizamos como parámetro de ingreso en los controllers como los de salida de dichos controllers y también el dto de de errores.



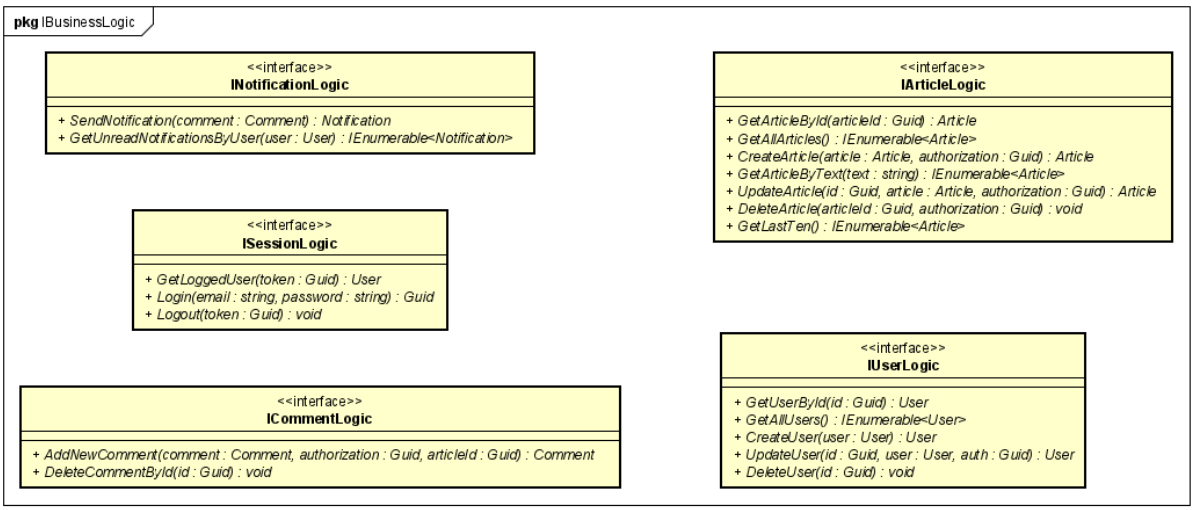
Blog.IDataAccess

Este paquete es el que contiene las interfaces que luego van a ser implementadas por el paquete Blog.DataAccess, esto lo realizamos para que se dependa de interfaces y no de implementaciones y así lograr más estabilidad en la solución y respetar el quinto principio solid, que las clases dependen de abstracciones y no de clases concretas. Las interfaces en este paquete luego van ser utilizadas en las clases del paquete de Blog.BusinessLogic. Aquí se encuentra solamente una clase, ya que decidimos crear un IRepository generico para evitar repetir código.



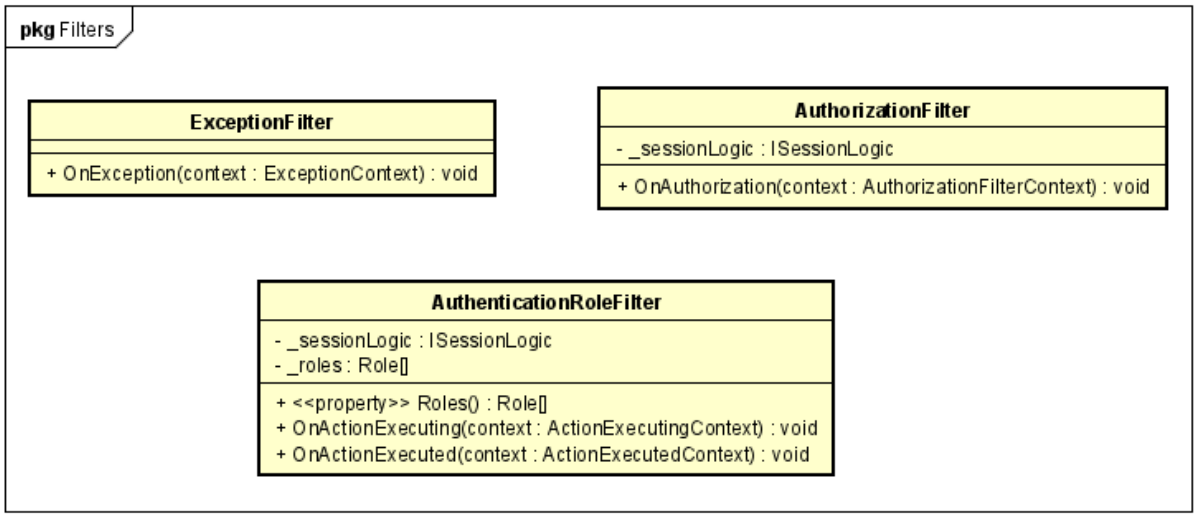
Blog.IBusinessLogic

Este paquete cumple la misma función que el de Blog.IDataAcces lo único que para las clases del dominio y que las interfaces creadas aqui son utilizadas posteriormente por los controladores del paquete de Blog.WebApi, en este paquete se encuentran las interfaces que luego van a ser implementadas en el paquete de Blog.BusinessLogic.



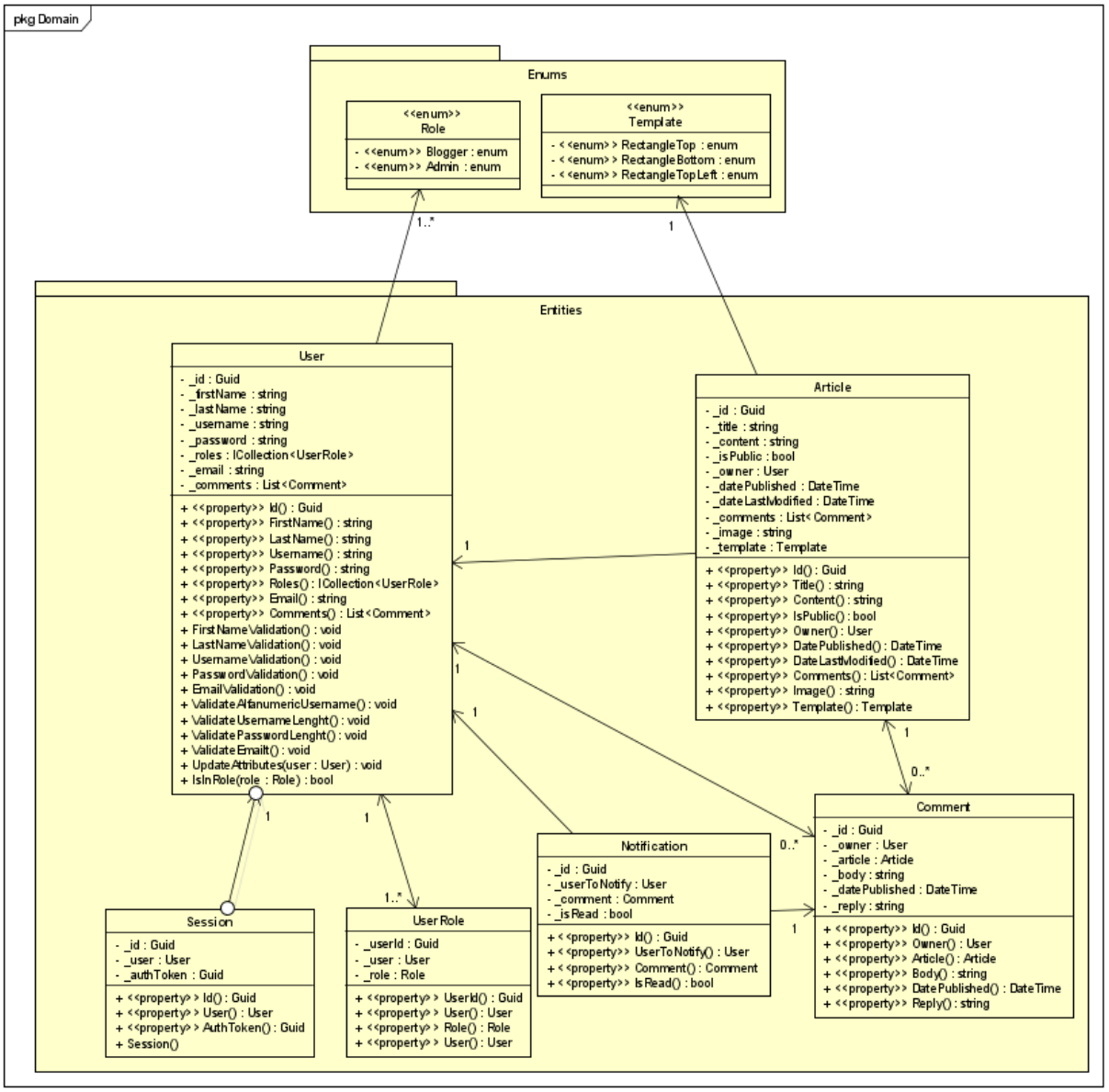
Blog.Filters

En este paquete se encuentran los diferentes filtros implementados para la solución, estos son: Filtro de autorización de usuarios, Filtro de autorización por roles de usuario y Filtro de excepciones. Decidimos poner estas clases en un paquete separado de Blog.WebApi para que allí se encuentren simplemente los controladores y así desacoplar esta lógica de este paquete.



Blog.Domain

Este paquete es donde se encuentran todas las clases del dominio, al igual que los diferentes enumerados y las excepciones que personalizadas que decidimos crear en la solución.

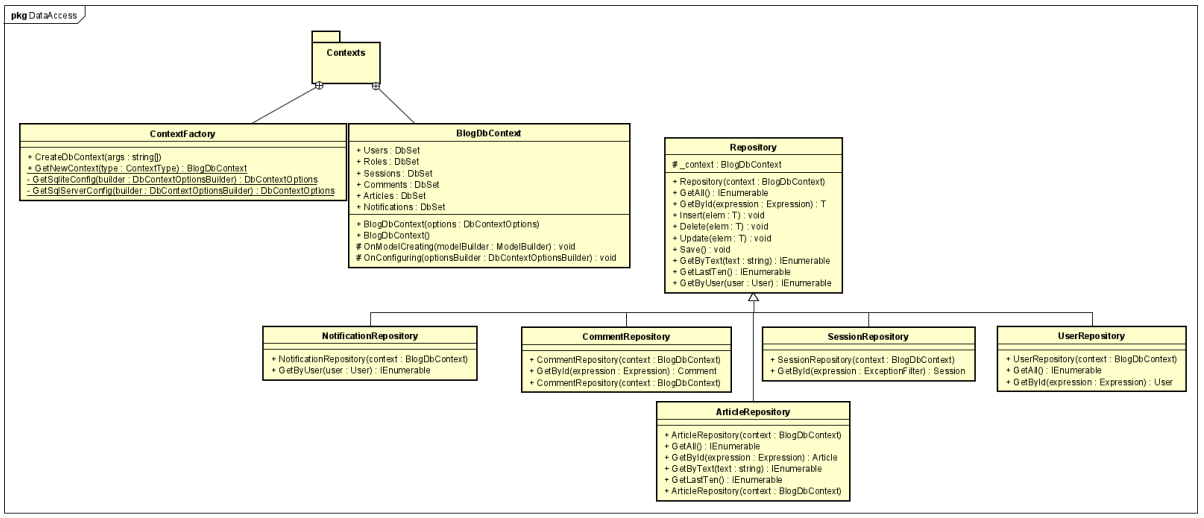


Blog.DataAccess.Test

Este paquete es el responsable de contener a todos los test unitarios que realizamos para las clases del paquete de Blog.DataAccess, en un principio teníamos todos los test unitarios dentro del paquete Blog.Tests, pero por el gran volumen de test utilizados con TDD y para que el paquete de Blog.Tests no quede muy sobrecargado, decidimos crear este paquete donde colocar solo los test unitarios creados para el acceso a datos.

Blog.DataAccess

En este paquete se encuentran todas las implementaciones que utilizamos para el acceso a datos, Aquí decidimos crear una implementación genérica con todos los métodos básicos de un acceso a datos, y luego crear las diferentes clases de repositorio para casos específicos, en estas implementaciones específicas se hace override de los métodos que requieren dicha lógica extra.

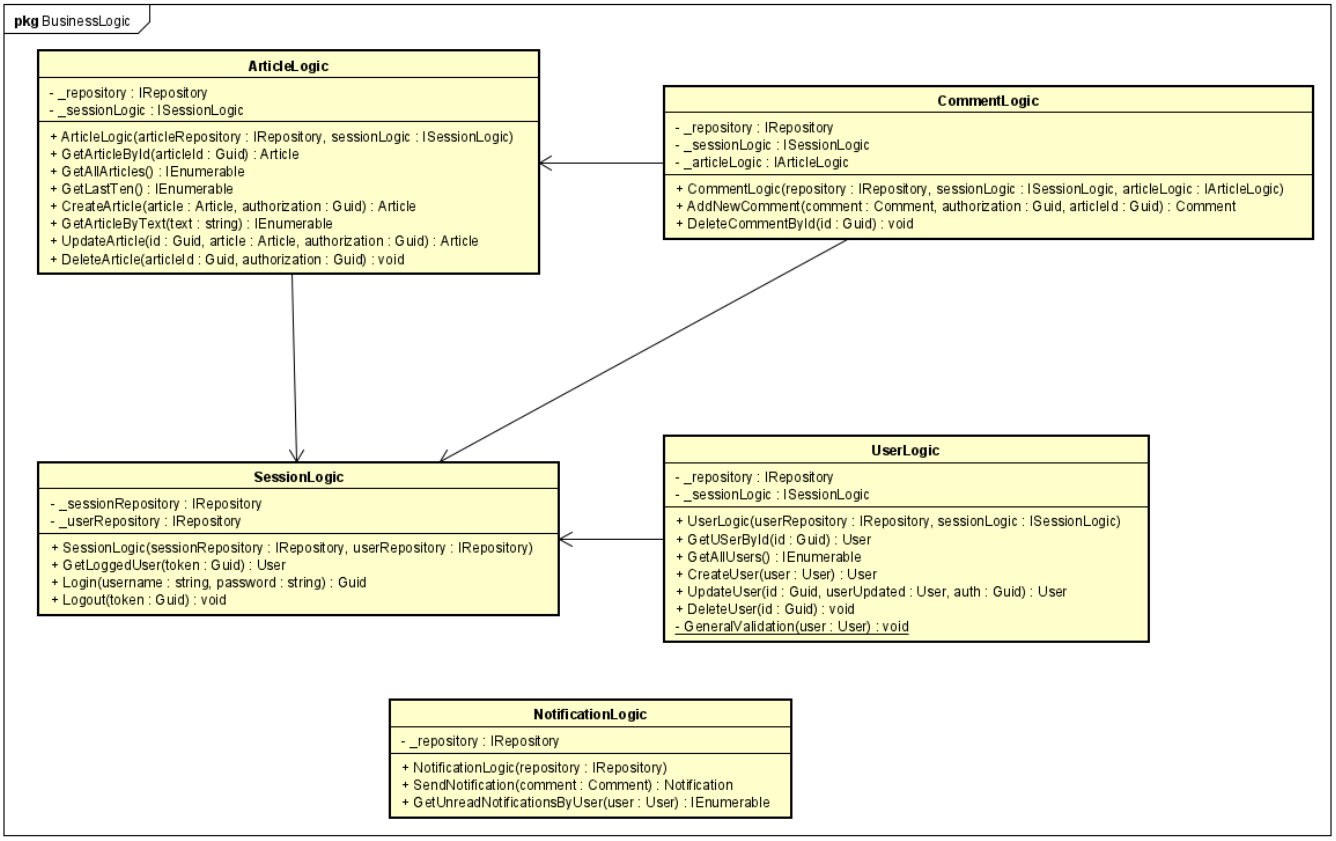


Blog.BusinessLogic.Test

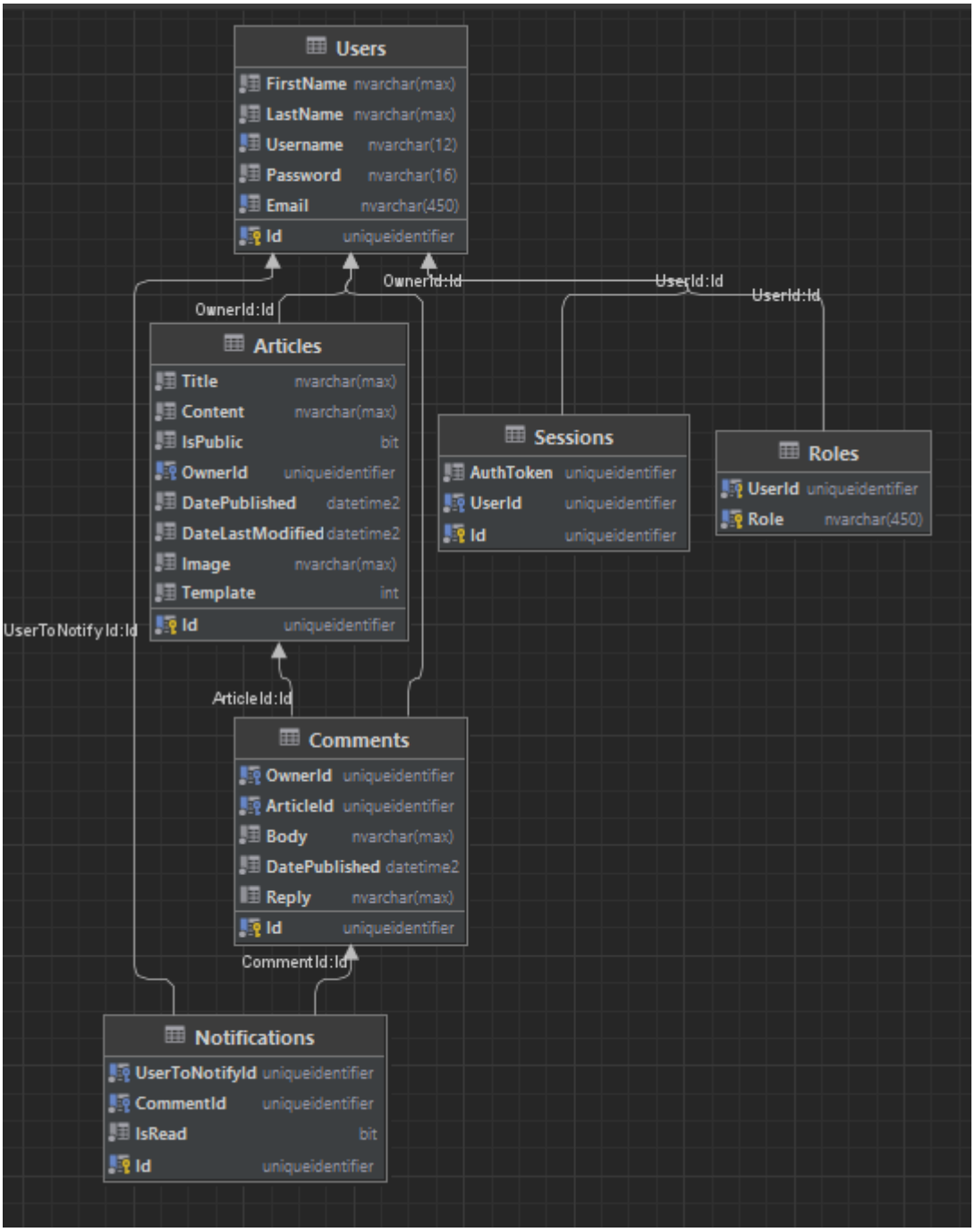
En este paquete al igual que sucedió con el de Blog.DataAccess.Test, en un principio estaba dentro del paquete Blog.Test, pero decidimos crear un proyecto aparte para no sobrecargar al otro paquete con tantos test y que no cumpla con tantas responsabilidades. Aquí se encuentran todos los test unitarios creados para la lógica de negocio de la solución.

Blog.BusinessLogic

Este paquete es el encargado de contener a todas las clases responsables de la lógica de negocio de la solución, esta clase implementa todas las interfaces anteriormente mencionadas dentro del paquete de Blog.IBusinessLogic y así cumplir con el quinto principio solid como ya se mencionó.


pkg BusinessLogic
ArticleLogic
- _repository : IRepository
- _sessionLogic : ISessionLogic
+ ArticleLogic(articleRepository : IRepository, sessionLogic : ISessionLogic)
+ GetArticleById(articleId : Guid) : Article
+ GetAllArticles() : IEnumerable
+ GetLastTen() : IEnumerable
+ CreateArticle(article : Article, authorization : Guid) : Article
+ GetArticleByText(text : string) : IEnumerable
+ UpdateArticle(id : Guid, article : Article, authorization : Guid) : Article
+ DeleteArticle(articleId : Guid, authorization : Guid) : void
CommentLogic
- _repository : IRepository
- _sessionLogic : ISessionLogic
- _articleLogic : IArticleLogic
+ CommentLogic(repository : IRepository, sessionLogic : ISessionLogic, articleLogic : IArticleLogic)
+ AddNewComment(comment : Comment, authorization : Guid, articleId : Guid) : Comment
+ DeleteCommentById(id : Guid) : void
SessionLogic
- _sessionRepository : IRepository
- _userRepository : IRepository
+ SessionLogic(sessionRepository : IRepository, userRepository : IRepository)
+ GetLoggedUser(token : Guid) : User
+ Login(username : string, password : string) : Guid
+ Logout(token : Guid) : void
UserLogic
- _repository : IRepository
- _sessionLogic : ISessionLogic
+ UserLogic(userRepository : IRepository, sessionLogic : ISessionLogic)
+ GetUSerById(id : Guid) : User
+ GetAllUsers() : IEnumerable
+ CreateUser(user : User) : User
+ UpdateUser(id : Guid, userUpdated : User, auth : Guid) : User
+ DeleteUser(id : Guid) : void
- GeneralValidation(user : User) : void
NotificationLogic
- _repository : IRepository
+ NotificationLogic(repository : IRepository)
+ SendNotification(comment : Comment) : Notification
+ GetUnreadNotificationsByUser(user : User) : IEnumerable

Modelo de tablas



Diagramas de interacción

Diagrama de secuencia de Crear Artículo

Se puede encontrar una copia del diagrama en github y junto a la documentación por si no se puede apreciar al 100% todo

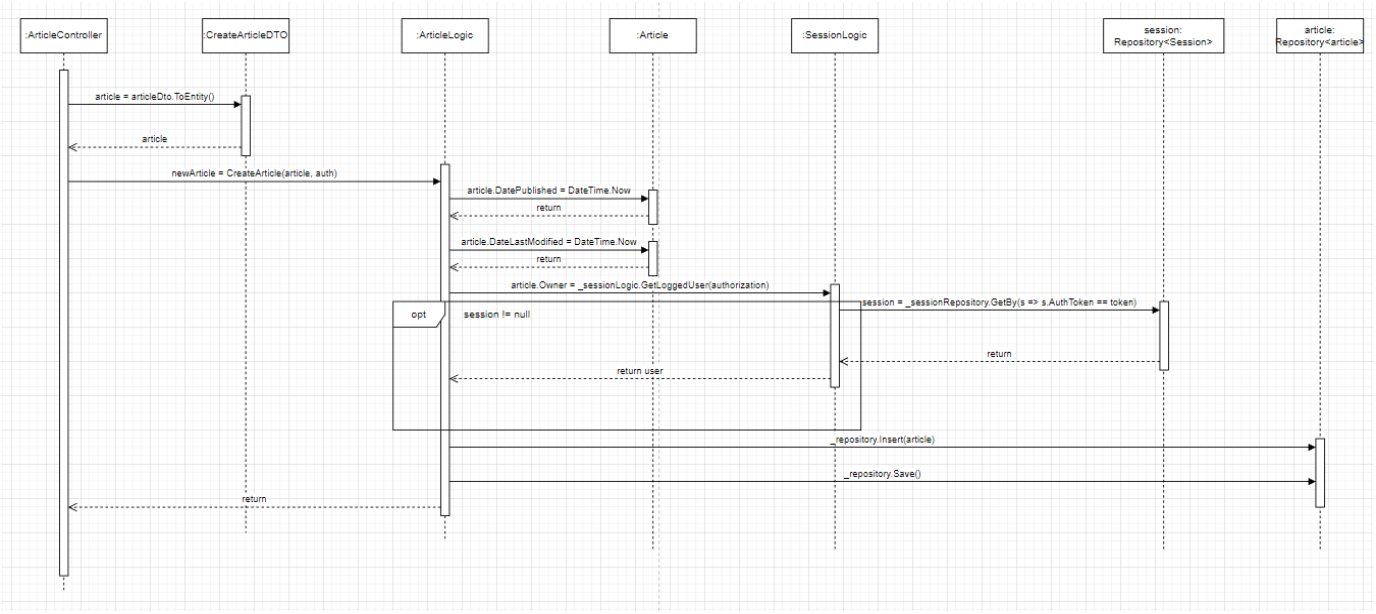
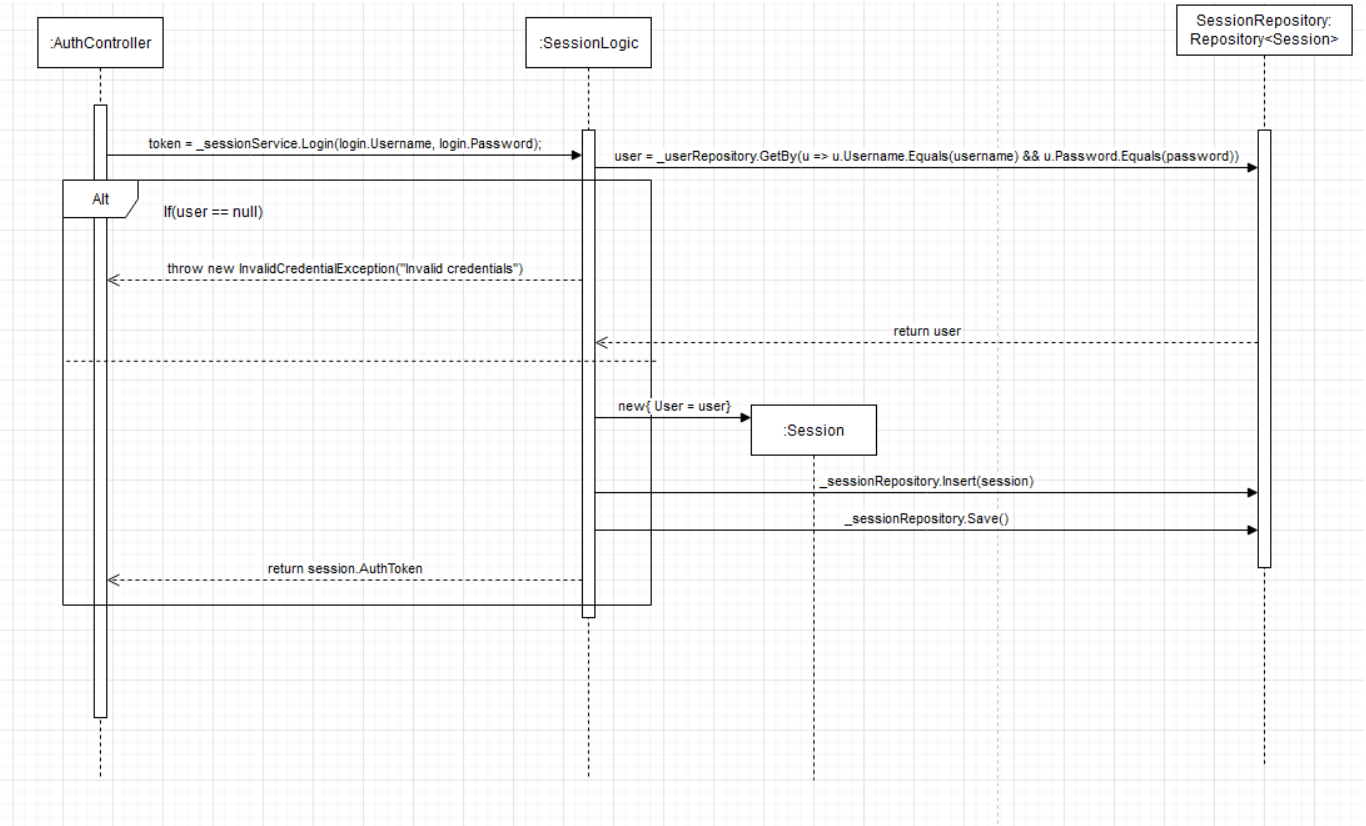


Diagrama de secuencia Login

Se puede encontrar una copia del diagrama en github y junto a la documentación por si no se puede apreciar al 100% todo



Justificación del diseño

Mecanismos de inyección de dependencias

Decidimos la utilización de un proyecto aparte para el manejo de la inyección de dependencias ya que nos parecía que ponerlo en la clase program del paquete de Blog.WebApi hubiese sobrecargado el paquete de dependencias que no eran necesarias para ese paquete como por ejemplo las interfaces de Blog.IDataAccess y las clases de DataAccess.

Filtros

Los filtros también decidimos ponerlos en un paquete aparte para que se cumpla con una sola responsabilidad en el y cumplir con las reglas vistas en clase sobre la agrupación de clases en paquetes a partir de responsabilidades, tratamos de que

los diferentes paquetes están agrupados por responsabilidades y que las clases dentro de un paquete tengan responsabilidades similares.

Clase de contextFactory

Decidimos crear una clase dentro del paquete de Blog.DataAccess para que nos facilite al momento de los test unitarios, ya que para los mismos utilizamos una base de datos de SqlLite y para la base de datos del codigo fuente utilizamos una base de datos de SQLServer.

Repository

Utilizamos una clase llamada Repository ubicada en el paquete DataAccess, la cual implementa la interfaz IRepository, Los métodos se implementan en esta clase y a la vez cada repositorio en particular hereda estos métodos, donde algunos los sobreescriben para casos particulares, para así de esta manera poder reutilizar la mayor cantidad de código posible.

Manejo de excepciones

Para el manejo de excepciones utilizamos un filtro dentro del paquete Blog.Filters, este filtro fue utilizado en los diferentes controladores en caso de que una excepción salte, en dicho filtro utilizamos los códigos de error: 400, 401, 404 y 500. Siendo el código de error 500 el código por defecto osea el que salta en caso de excepciones que no manejamos en nuestro sistema. También creamos una excepción custom para cuando no se encontraba un elemento la cual llamamos NotFoundException.

Principios aplicados

Como se podrá ver en nuestra solución buscamos acoplarnos lo máximo posible algunos de los principios SOLID, como por ejemplo el de responsabilidad única, ya que la gran mayoría de nuestras clases cumplen con una responsabilidad, por ejemplo la clase ArticleLogic cumple solamente con la responsabilidad de la lógica de los artículos.
También buscamos cumplir con el quinto principio SOLID y hacer que las clases concretas dependan de clases abstractas, como por ejemplo en los controladores

los cuales usan las interfaces de IBusinessLogic y en la lógica de negocio que se utilizan las interfaces de IDataAccess.

Diagrama de componentes

La solución se dividió en los componentes de notification, comments, article, user, session y repository ya que son las partes fundamentales del blog, todos los componentes menos el article son componentes propios de un blog con autenticación y administración de usuarios, y luego se decidió agregar el componente repositorio ya que todos los componentes son guardados allí y los diferentes componentes necesitan de ella.