Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de Aplicaciones 2

Obligatorio 2

Link a Repositorio - BlogsApp

Fernando Spillere - 274924

Gimena Alamón - 243518

Carmela Sotuyo - 186554

Junio 2023

Índice

Introducción

1.1 Descripción General de la solución

La solución que presentamos en este documento se basa en los requerimientos identificados tanto en la primera como segunda letra. El código en general, así como nombres de entidades y componentes fue realizado en inglés. Se siguen los estándares, buenas prácticas, uso de patrones y recomendaciones que entendemos hacen a un código más legible, re utilizable y sencillo de entender. A continuación se detallan algunas consideraciones relevantes.

1.2 Bugs conocidos y resoluciones

1.2.1 Bugs pendientes de entrega anterior

En la entrega anterior quedaron dos bugs que fueron corregidos en la presente. Los mismos son:

1. Anteriomente un usuario con rol Blogger era capaz de auto asignarse el rol de Administrador. En esta versión esta acción no está permitida. Es decir, si un usuario trata de ejecutarlo, con las condiciones descritas, el sistema le brindará un error expresando que ”No puede autoasignarse roles”

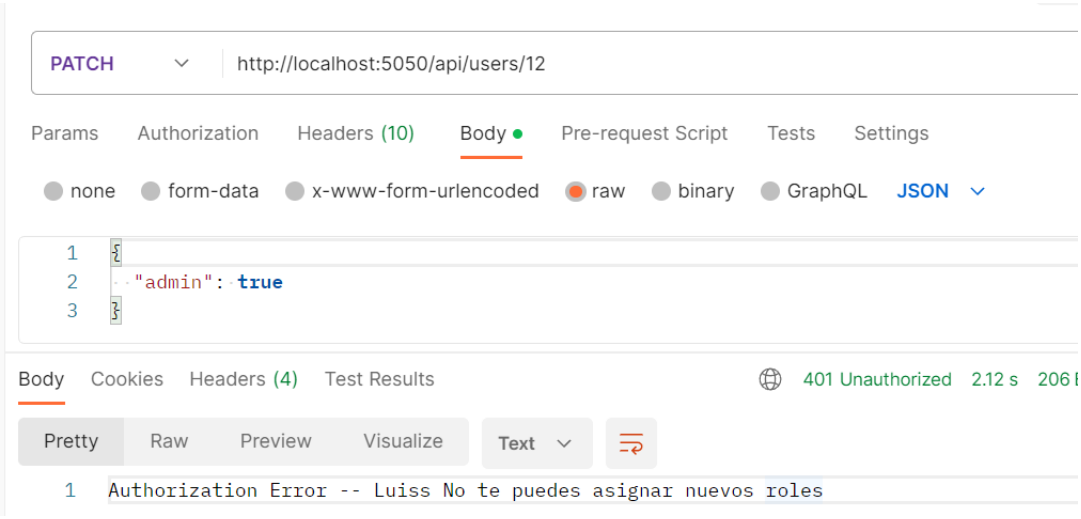

Figure 1.1: Ejecución Patch User

2. El ranking de usuarios tenía algunos errores de implementación, por lo cual no estaba fucionando correctamente en la entrega anterior. En esta versión el endpoint del ranking se ejecuta sin inconvenientes, brindando la información que se solicita.

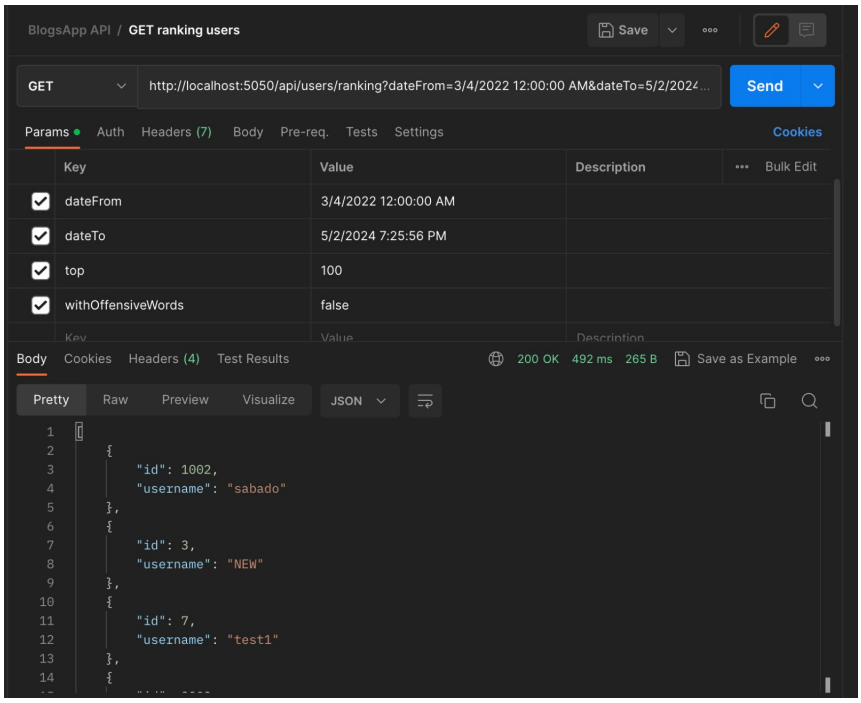

Figure 1.2: Ejecución Ranking

A su vez, para implementar la nueva funcionalidad del nuevo ranking, se agrega el parámetro WithOffensiveWords, con valores True o False, para determinar si se desea agregar el parámetro al mismo.

1.2.2 Bugs pendientes de resolución

- Como bug conocido, los sub comentarios se muestran duplicados en los artículos, una vez en su ubicacion correcta (debajo del comentario al que pertenece) y otra vez como si fueran comentarios directos sobre el articulo.

- Si se cierra el navegador con una sesion activa, se pierde la session por el sessionStorage.

Arquitectura de la solución

2.1 Descripción General

La solución costa en los siguientes paquetes:

El paquete WebAPI contiene todos los controladores (Controllers), los filtros (Filters), Objetos de transferencia de datos (DTOs) y una carpeta con los .dlls de los importadores que se utilicen. En la entrega se incluye un .dll para un objeto .json.

Este paquete se comunica con la lógica mediante el paquete de Interfaz de lógica (IBusinessLogic), el cual expone interfaces dependiendo de la entidad del dominio con la cual se trabaje.

Luego la lógica contiene la implementación de los métodos expuestos por las interfaces, y se comunican con la base de datos mediante las interfaces expuestas por el paquete de IDataAccess/Interfaces.

En Domain se encuentran las entidades que componen la solución, y ServiceFactory se encarga de la inyección de dependencias para la lógica y la API. A su vez, también existen 3 proyectos de tests que corresponden a la API, Lógica, Base de datos y Dominio.

A su vez, y con motivo de hacer este módulo independiente por futuros cambios que puedan surgir, se crean componentes para gestionar el sistema de Logs. Se dispone entonces de la clase Logging donde se define la entidad, Logging.Logic.Services para la lógica y Logging.DataAccess para el acceso a datos.

2.2 Diagramas

2.2.1 Diagrama de componenes

En el mismo se logra visualizar como se dividen y organizan los componentes del sistema, cómo interactúan entre sí y cómo se comunican.

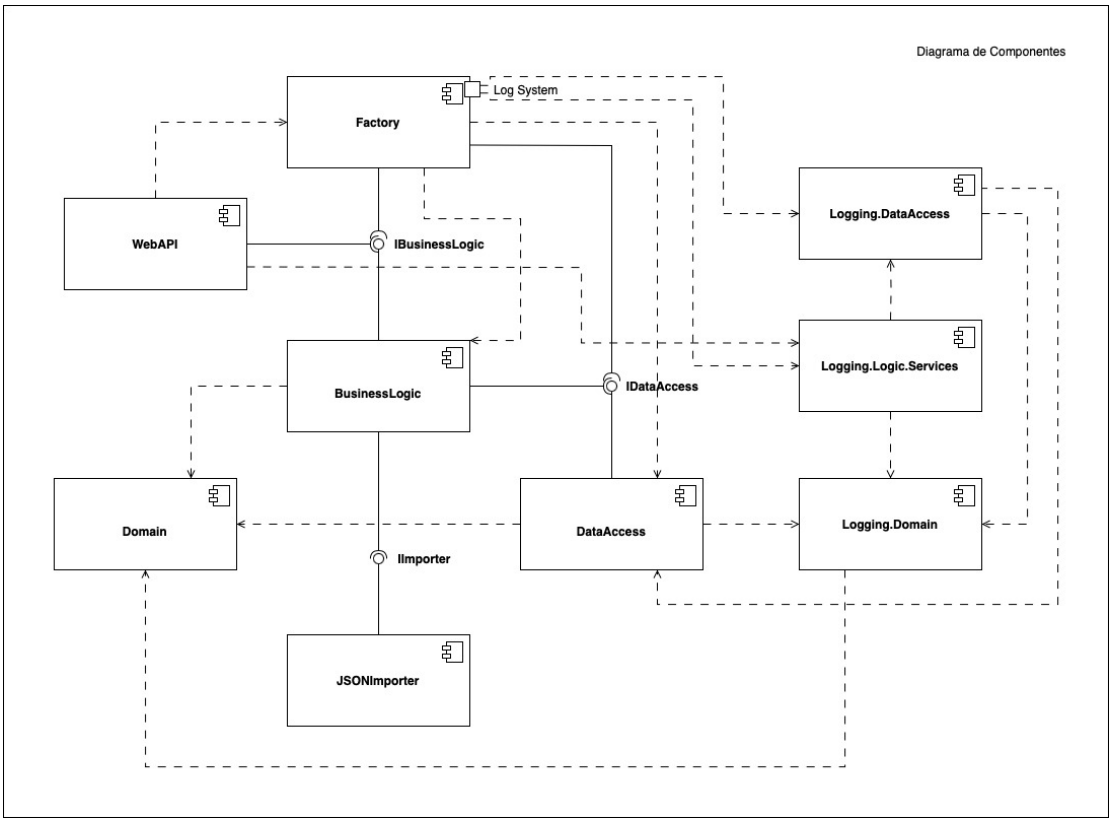

Figure 2.1: Diagrama de componentes

2.2.2 Dominio

En el paquete de Dominio se definen:

- Las entidades clave de la solución, como ser las entidades de negocio.

- Las excepciones relacionadas a errores de datos y lógica. Aquí se definieron BadInputException, InterruptedActionException y ExistenceException y NonExistantImplementationEx-

ception.

Diagrama de paquetes

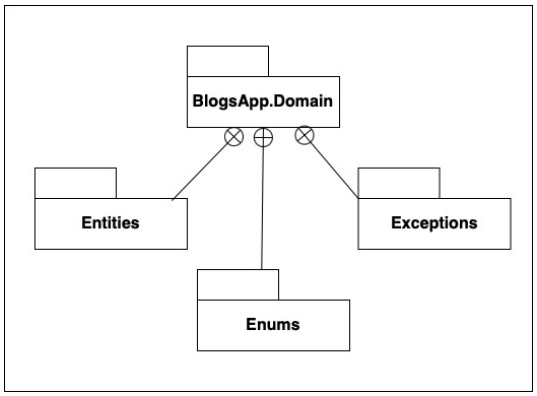

Figure 2.2: Dominio - Dagrama de paquetes

Diagrama de clases

En el mismo se logra visualizar el detalle de la definición de cada clase que compone a los objetos de la realidad y como se comunican entre ellos. Respecto a la entrega anterior, se visualiza una nueva clase Offensive Word, la cual tiene la responsabilidad de gestionar las palabras ofensivas que serán agregadas por los moderadores, y sus relaciones con artículos y comentarios. Otra diferencia es que ya no se cuenta con la clase ”Reply”, sino que los comentarios de comentarios son gestionados bajo la misma entidad Comment. A su vez, en la clase User se logra visualizar el nuevo atributo Moderador de tipo booleano, el cual llevará el registro de si el usuario dispone o no de ese rol.

Adicionalmente, se incluyó la clase abstracta Content, de la cual heredan Article y Comment. Esto se realizó para poder manejar ambas entidades como una sola a la hora de evaluar el contenido ofensivo.

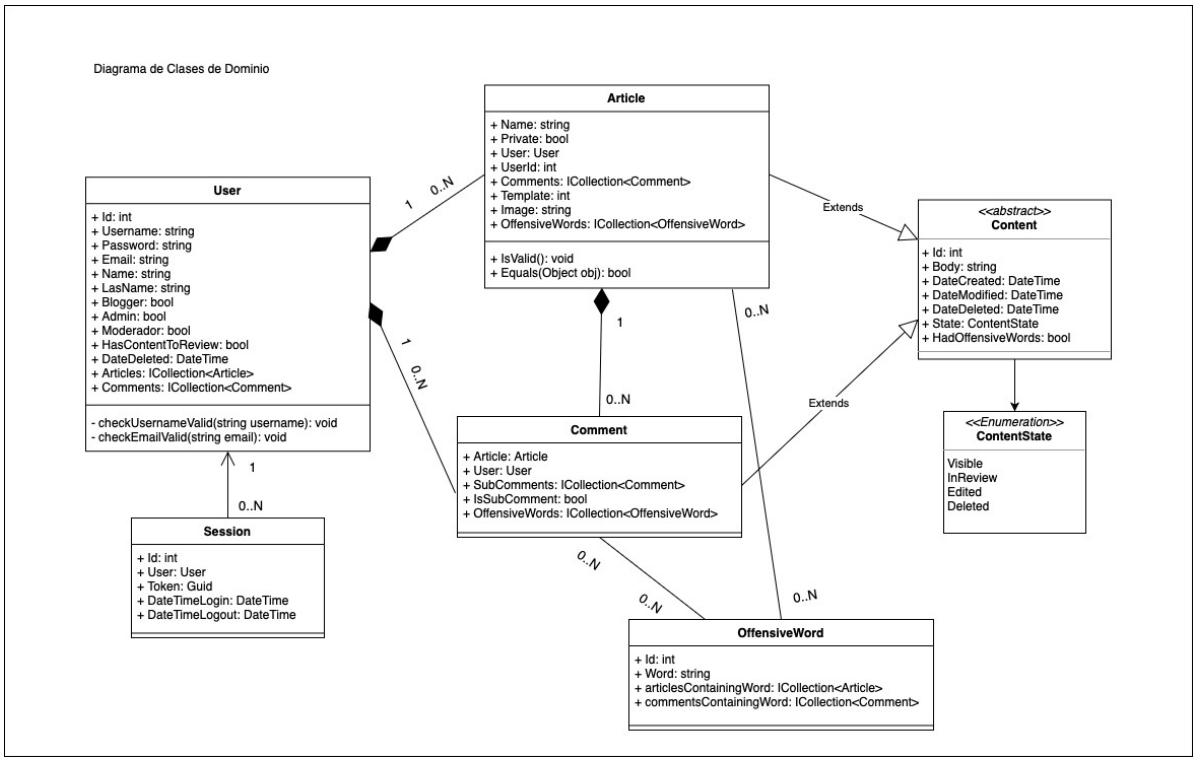

Figure 2.3: Diagrama de clases - Dominio

2.2.3 Lógica de negocio

Diagrama de paquetes

Se logra visualizar como están separados los paquetes de implementación de las interfaces para cumplir con la inversion de dependencias, y de este modo otros paquetes como WebApi dependan de abstracciones estables.

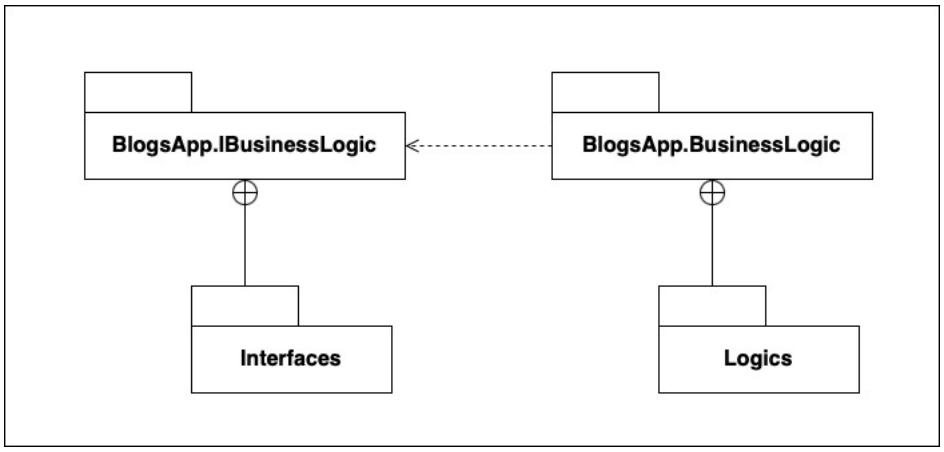

Figure 2.4: Diagrama de paquetes - BusinessLogic

Diagramas de clases

Diagramas de clases pertenecientes a la lógica de negocio, con sus respectivas relaciones:

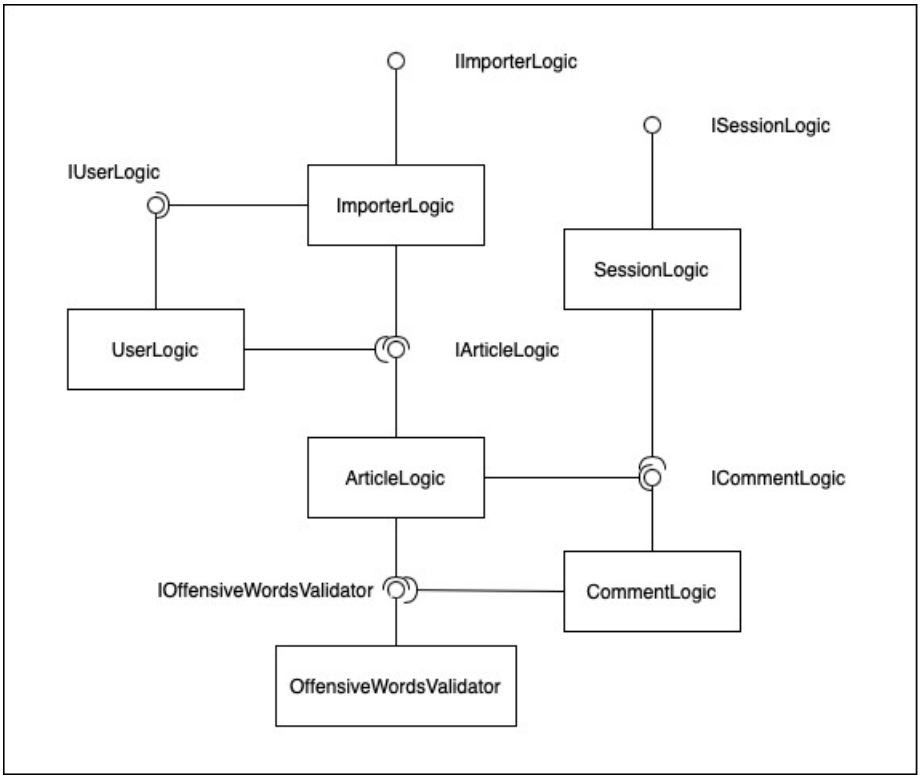

Figure 2.5: Diagrama de clases - BusinessLogic

Detalle de cada clase

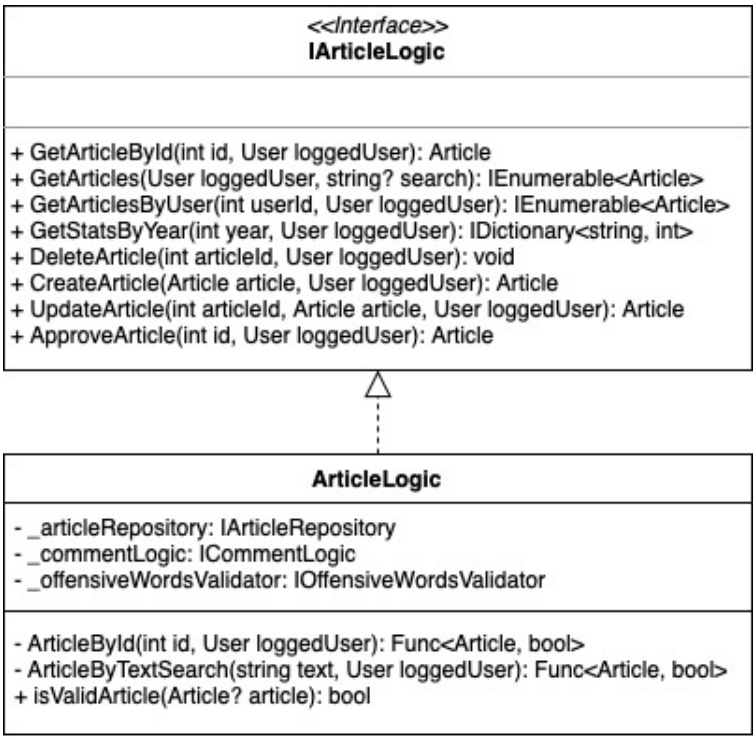

Figure 2.6: Diagrama de clases - ArticleLogic

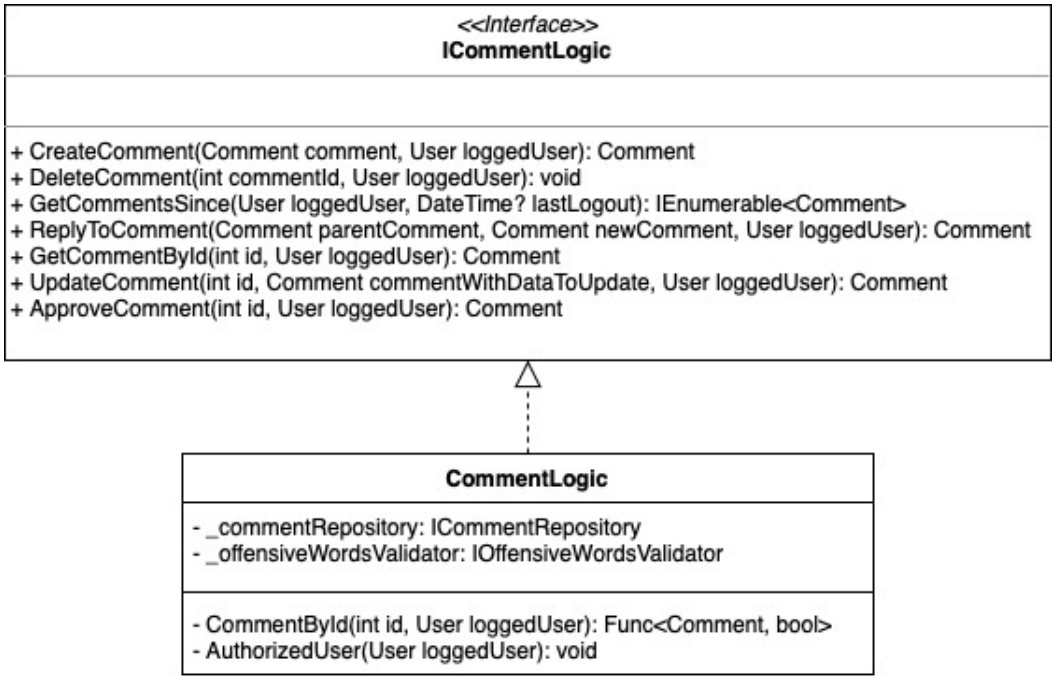

Figure 2.7: Diagrama de clases - Commentlogic

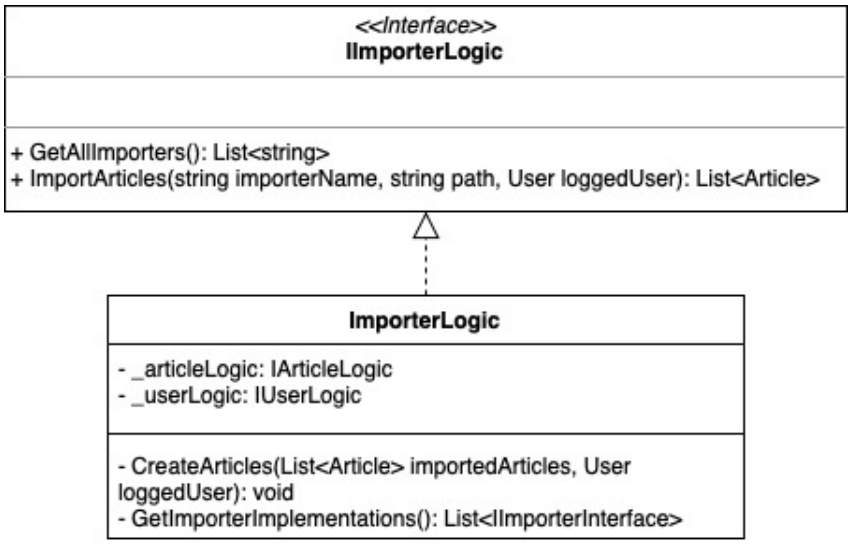

Figure 2.8: Diagrama de clases - ImporterLogic

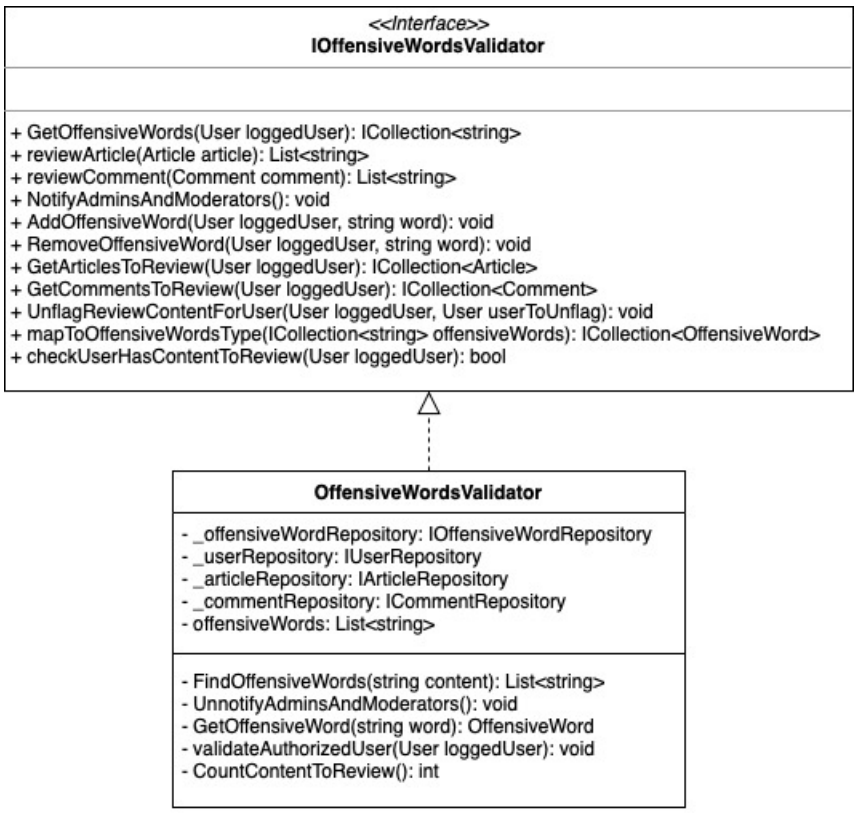

Figure 2.9: Diagrama de clases - OffensiveWordsValidator

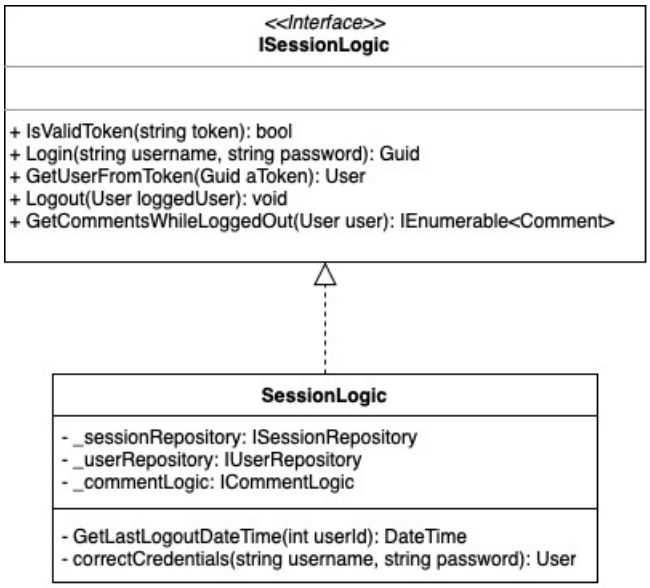

Figure 2.10: Diagrama de clases - SessionLogic

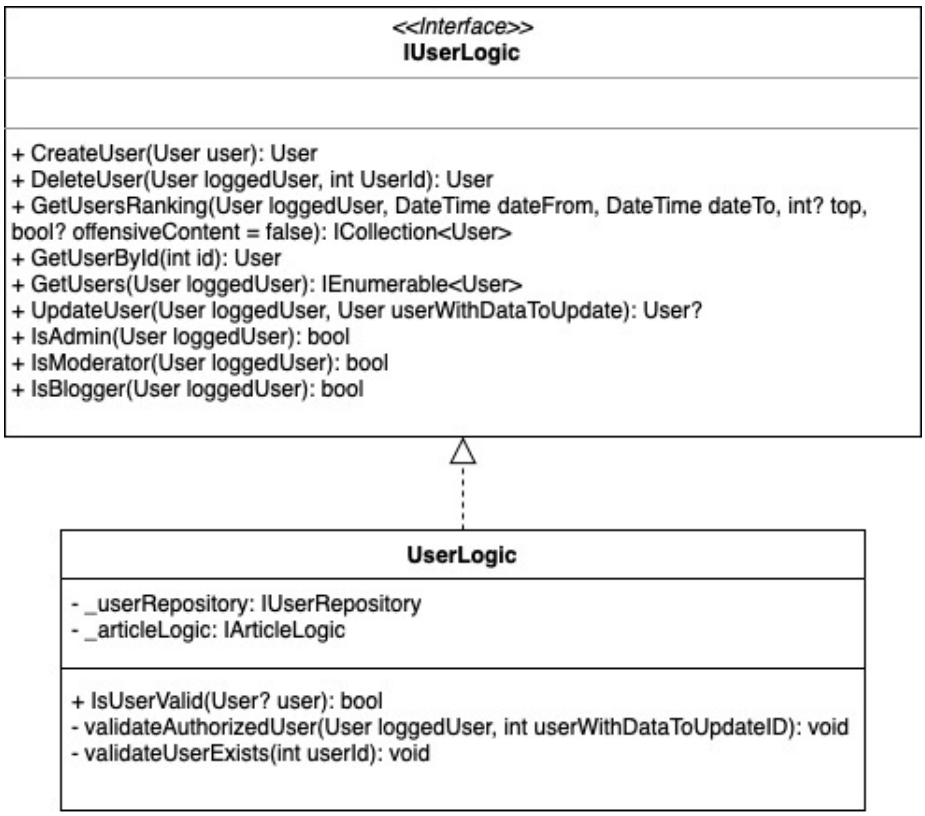

Figure 2.11: Diagrama de clases - UserLogic

2.2.4 Logs

Para llevar a cabo la funcionalidad de registro de acciones del sistema y que el mismo tenga independencia del sistema, teniendo en cuenta que en un futuro puede surgir una nueva forma de mantener el log, se realizó una implementación por separado de sus componentes, como se detalla a continuación:

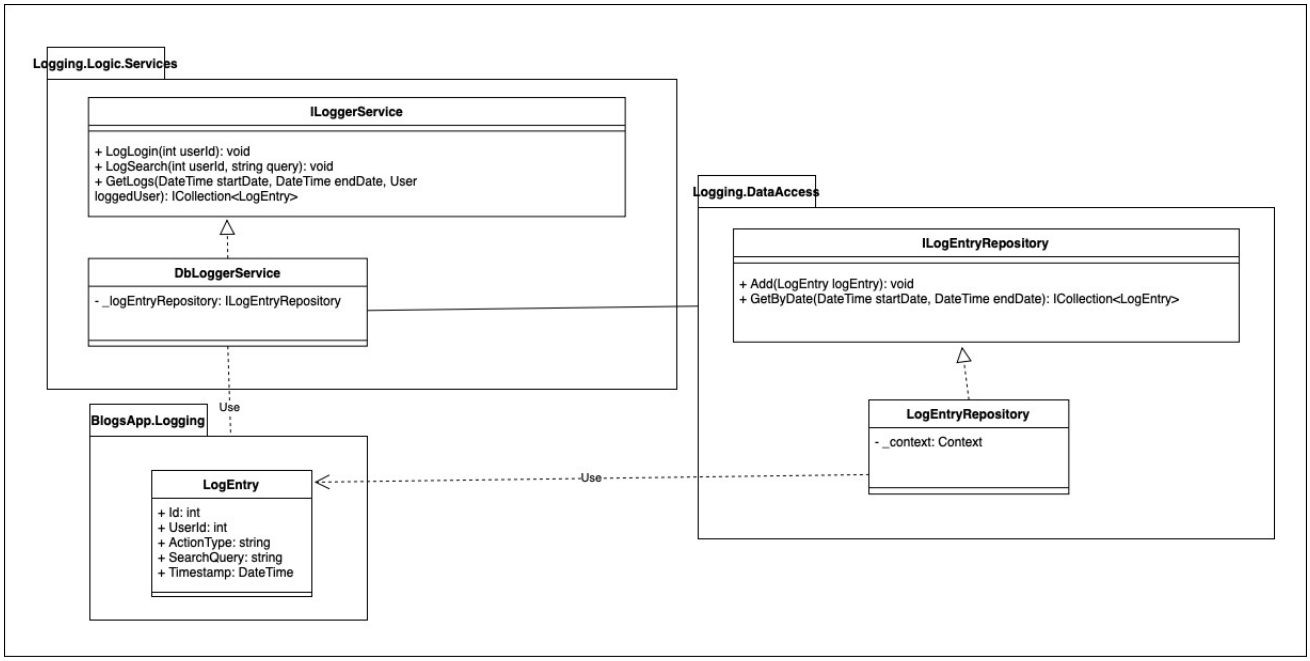

Figure 2.12: Diagrama de clases - UserLogic

2.2.5 DataAccess

Diagrama de paquetes

Los paquetes de implementación de DataAccess se encuentran separados de las interfaces que implementan, con el objetivo de desacoplar y cumplir con la inversion de dependencias, y de este modo otros paquetes puedan depender de abstracciones más estables.

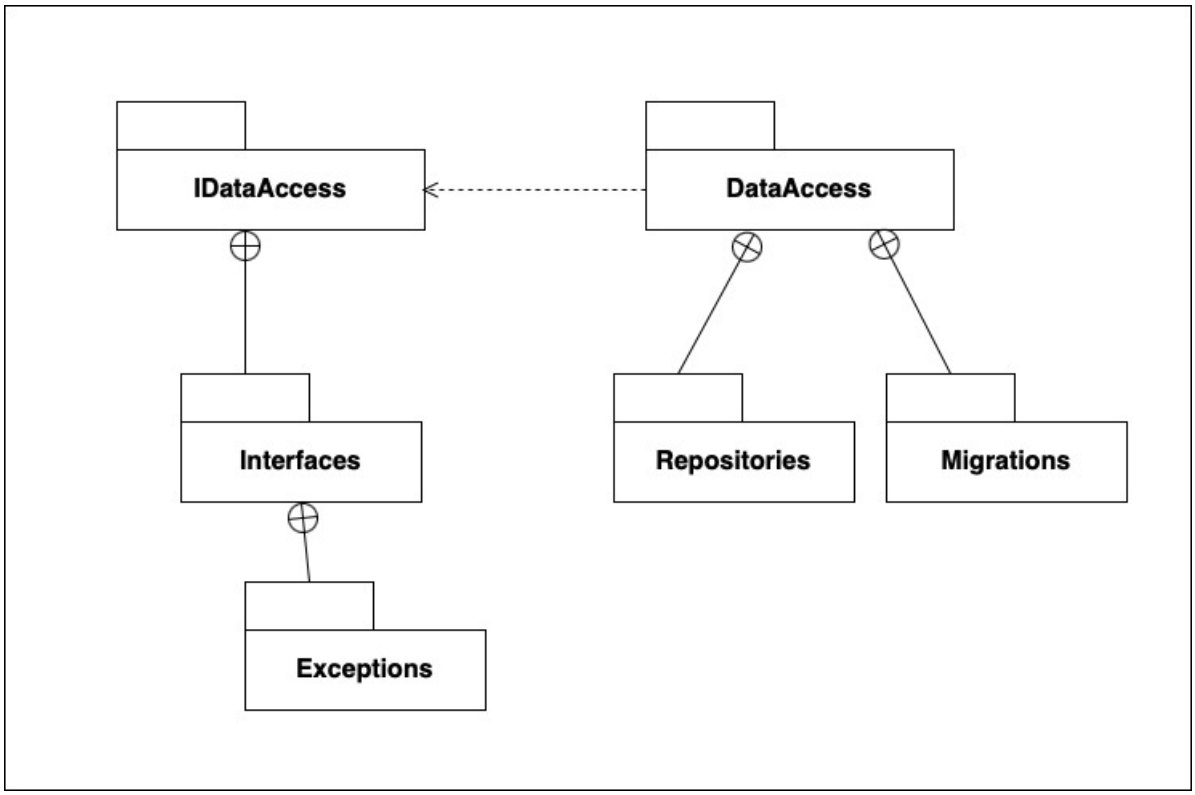

Figure 2.13: Diagrama de paquetes - DataAccess

En el diagrama de clases se puede visualizar cómo los Repositories implementan sus respectivas interfaces, a la vez en la que heredan de un Repository mayor, el cuál contiene métodos comunes a todos.

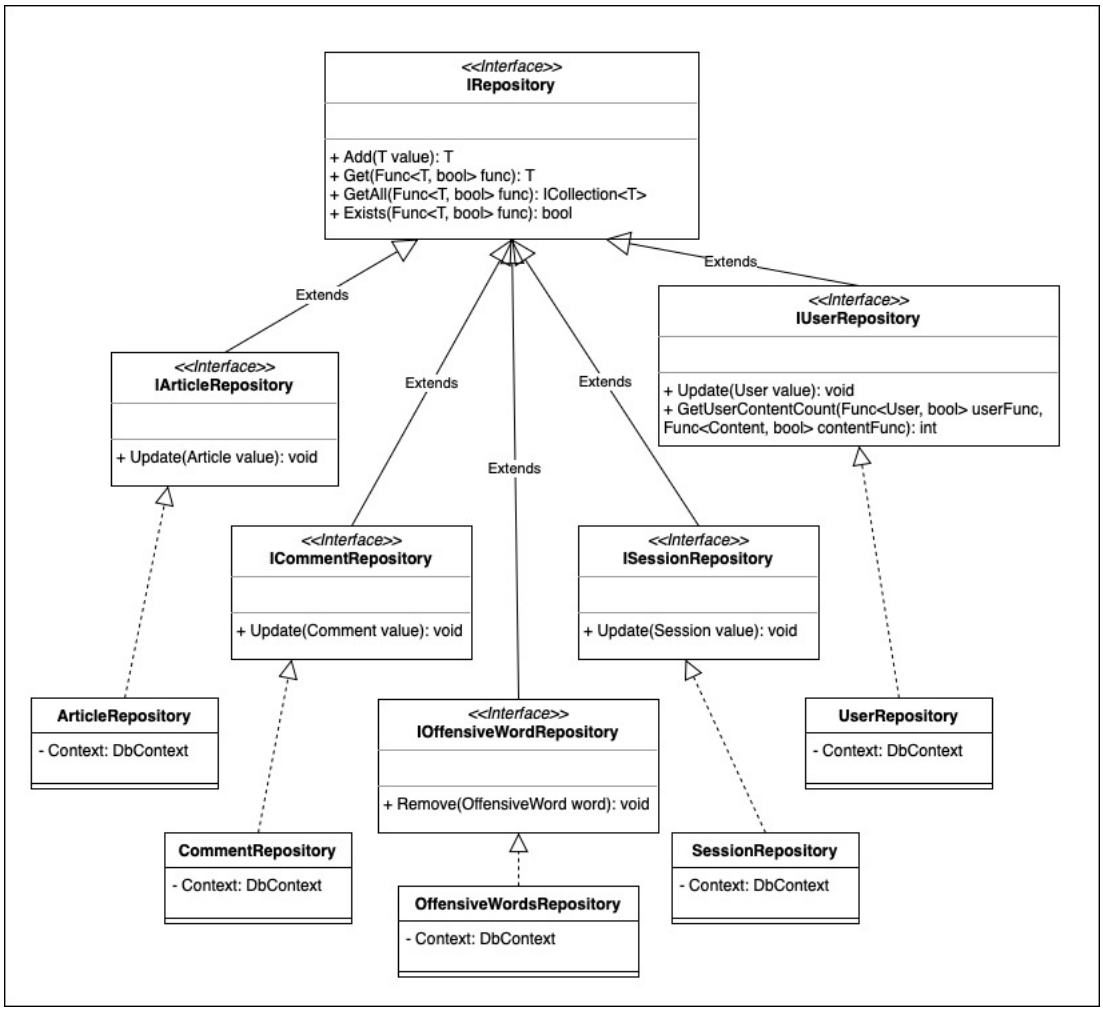

Figure 2.14: Diagrama de paquetes - DataAccess

2.2.6 WebApi

Diagrama de paquetes

En cuanto a la arquitectura de la API se define un proyecto WebAPI, el cual contiene los siguientes paquetes:

- Controllers

- DTOs

- Filters

- Importers

Existe un controller por cada recurso, el cual se encarga de gestionar las peticiones relacionadas al mismo, utilizando las interfaces de la lógica para cumplir con las funcionalidades.

Los DTOs se utilizan con el objetivo de proteger al dominio en la comuniciación hacia y desde los controllers, pero también sin depender de información innecesaria al momento de crear o actualizar los recursos. Para hacer conversiones de objetos a DTOs y viceversa se definen Converters, los cuales son clases con métodos que nos permiten realizar dichas conversiones.

En los filtros se encuentran ”AuthorizationFilter”, el cual tiene la responsabilidad de gestionar las sesiones de los usuarios, es decir, otorga Tokens de inicio de sesión a cada usuario , y evalúa en cada petición que el mismo se encuentre logueado en el sistema; y ”Exceptionfilter”, el cual se encarga del manejo de excepciones que surjan resultado de la ejecución de peticiones. Dependiendo del error que surja, retorna un mensaje y código de error.

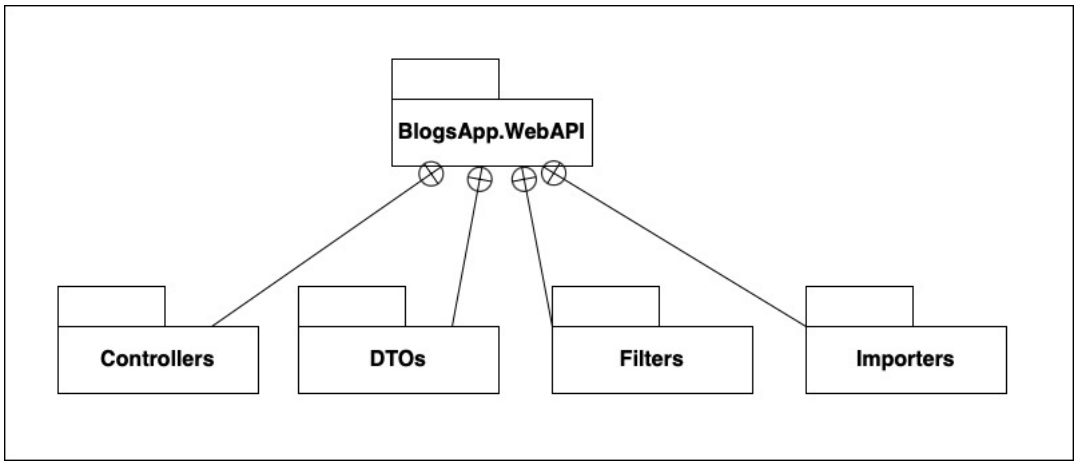

Figure 2.15: Diagrama de paquetes - WebApi

En el siguiene diagrama se puede visualizar como los controllers (Web Api) usan las interfaces de la lógica (BusinessLogic). Además, los mismos heredan de Base controller, cual contiene el método GetUserFromToken que utilizan todos los demás.

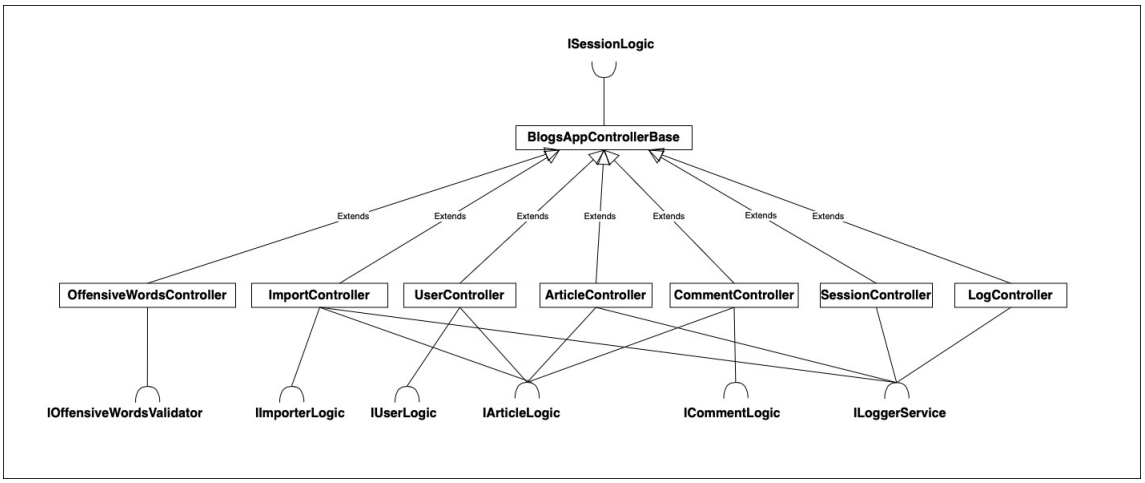

Figure 2.16: Diagrama - Comunicación Controllers-Lógica

Si se visualiza el detalle de la implementación de cada controller, se entiende que cada método implica un endpoint, y que en el mismo se define el verbo HTTP utilizado, los parámetros y body que se reciben, la comunicación con la lógica del sistema para ejecutar las acciones, y la respuesta que se brinda al usuario. A continuación se detalla cada uno:

como se comentó anteriormente, el BaseController contiene la implementación del método que resuelve el token del usuario que realiza la petición, por lo cual el resto hereda del mismo.

Figure 2.17: Diagrama - BaseController

En el controller de usuarios se incluyen todos los endpoints referidos exclusivamente a esta entidad, pero también a algunos que tienen relación con artículos, como GetRanking y GetUserArticles.

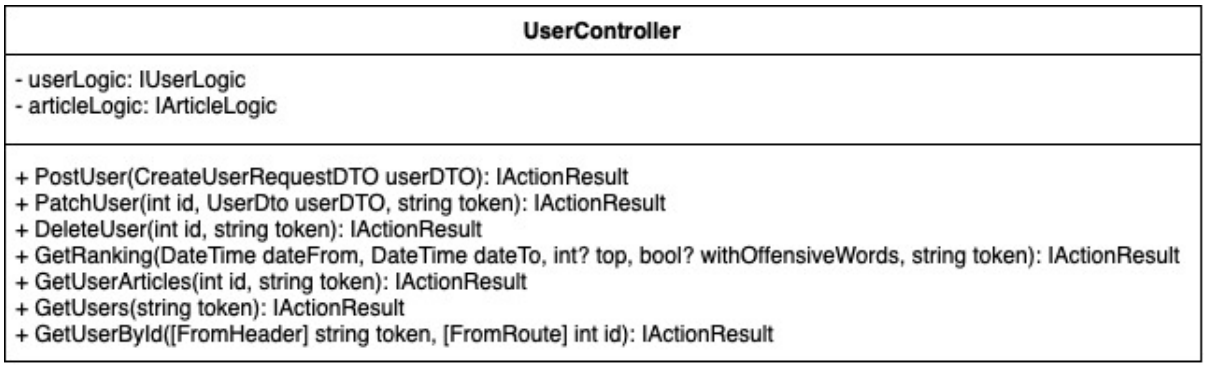

Figure 2.18: Diagrama - UserController

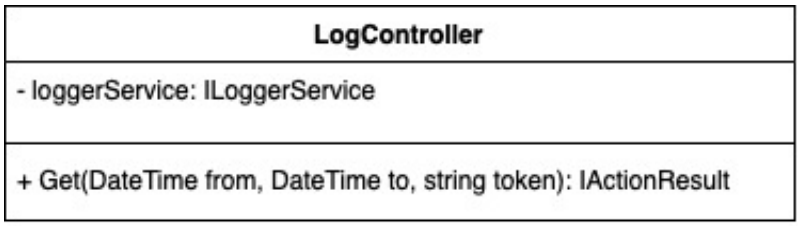

Figure 2.19: Diagrama - LogController

Este controller es nuevo para la presente entrega. Se encarga de los endpoints relacionados a la importación de artículos. Independientemente de cuál sea el formato elegido.

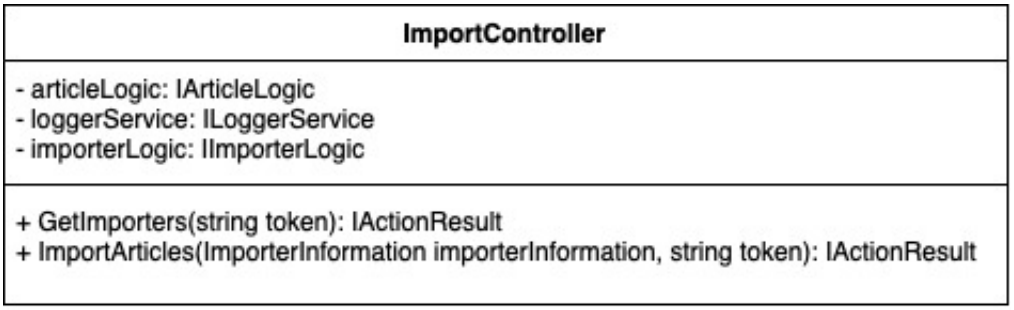

Figure 2.20: Diagrama - ImportController

Este contolador sufrió varios cambios respecto a la entrega anterior, dado que en la actual se necesitó incluir el manejo de palabras ofensivas, lo cual se relaciona con la entidad. Surgen endpoints tanto para modificar un comentario como para aprobarlo en caso de no disponer de palabras ofensivas.

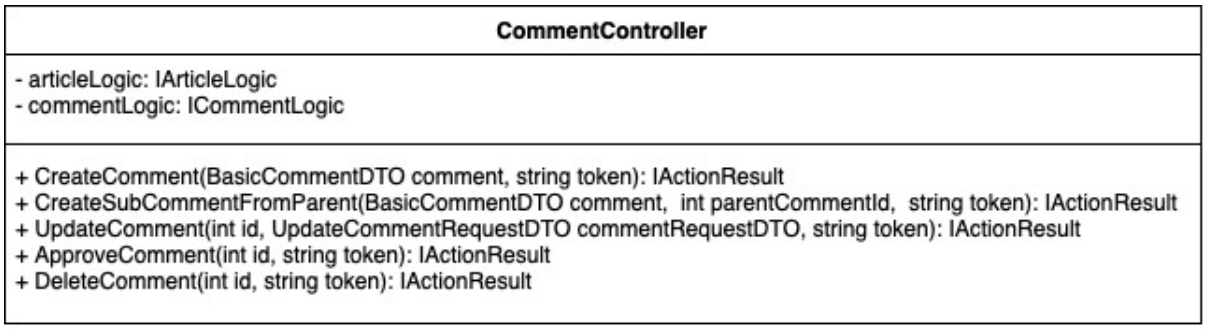

Figure 2.21: Diagrama - CommentController

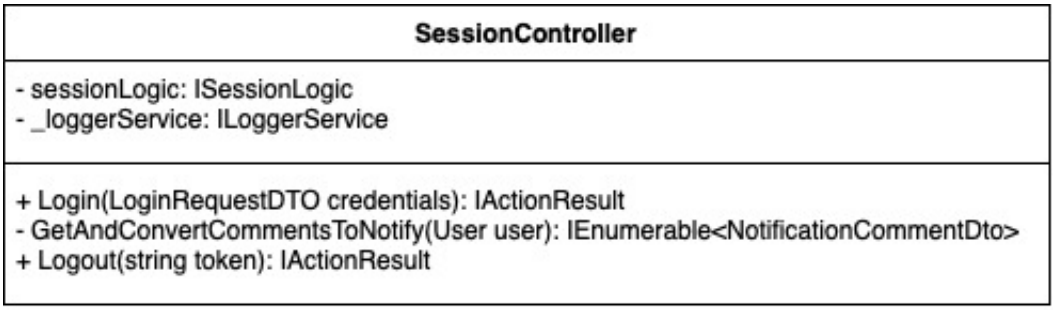

Figure 2.22: Diagrama - SessionController

Este controller es completamente nuevo. Se creó con el objetivo de gestionar las peticiones relacionadas a las palabras ofensivas, su control y aprobación.

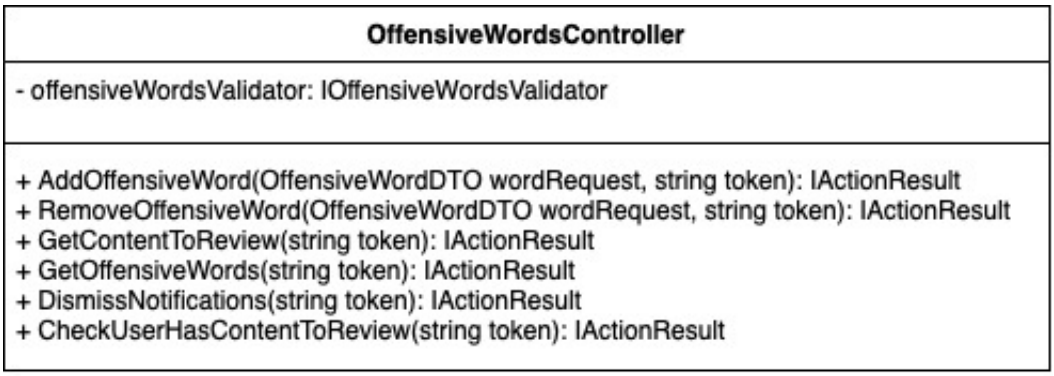

Figure 2.23: Diagrama - OffensivWordsController

En el controller de Artículos también surge un endpoint nuevo, con el fin de aprobarlos. También relacionado a la nueva funcionalidad de palabras ofensivas.

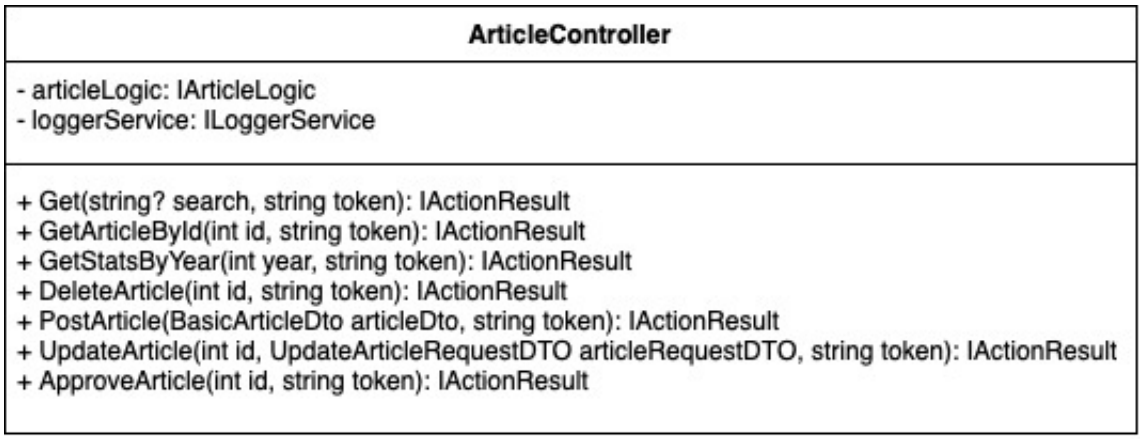

Figure 2.24: Diagrama - Articlecontroller

2.3 Base de Datos

Se utilizó Entity Framework Core, modalidad Code-First para la definición de la Base de datos, así como para el manejo de los mismos.

Teniendo como base la defínción de las clases que representan el dominio, Entity Framework creó las tablas que contendrán la información de los objetos instanciados a partir de las clases del dominio.

Se detalla la estructura de la base de datos y la relaciones entre las tablas:

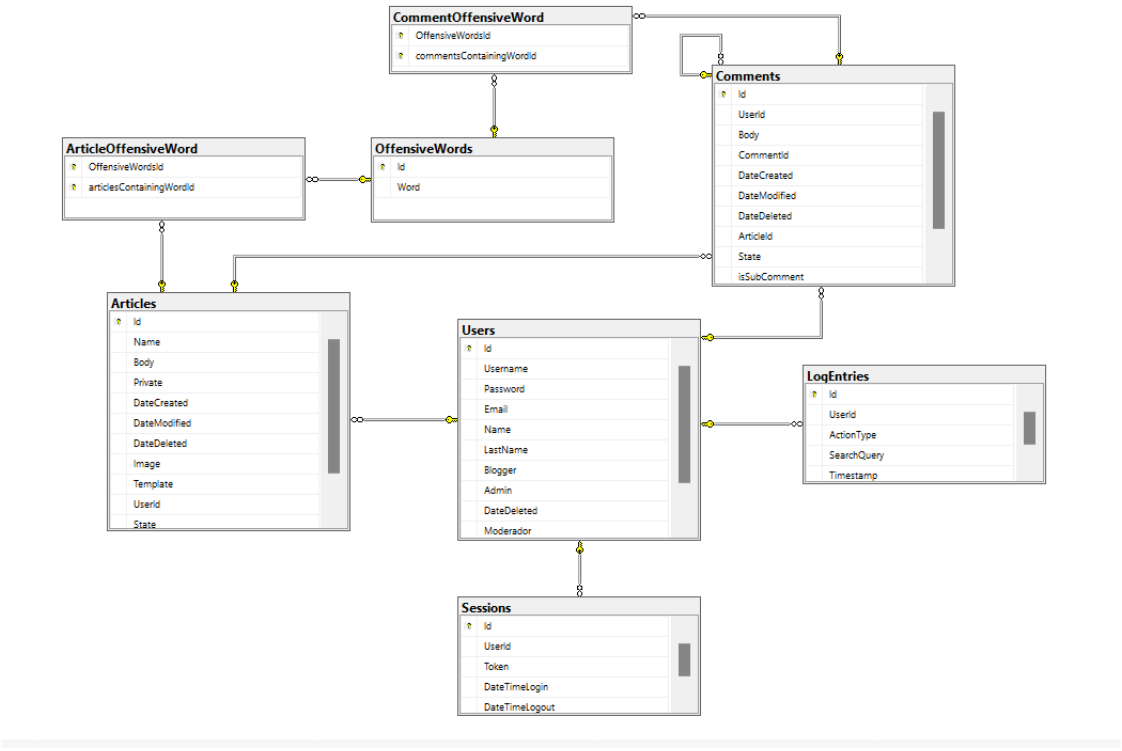

Figure 2.25: Modelo de tablas

Se puede observar la relación de Comment con si mismo, lo cual representa que un comentario puede estar asociado a otro comentario, y así conformar el hilo de comentarios de un artículo. La tabla OffensiveWords también es nueva, se encarga de guardar todas las palabras que los administradores consideren como ofensivas. Por otro lado tenemos las tablas ArticleOffensiveWord y CommentOffensiveWord, también nuevas, las cuales surgen de la relación N-N entre Article-OffensiveWord y Comment-OffensiveWord respectivamente. Esto quiere decir, que tanto los artículos como los comentarios pueden tener de 1-N palabras ofensivas, y que las mismas se pueden repetir en otros.

Justificación y explicación del diseño

3.1 Métricas

Assemblies	Relational Cohesion	Instability	Abstractness	Distance
BlogsApp.WebAPI	3.21	1	0	0
BlogsApp.Logging.Logic.Services	1.5	0.76	0.17	0.05
BlogsApp.Logging.DataAccess	1.4	0.95	0.2	0.1
BlogsApp.Logging	1.25	0.65	0	0.25
BlogsApp.IDataAccess	1.5	0.62	0.5	0.08
BlogsApp.IBusinessLogic	1.29	0.6	0.57	0.12
BlogsApp.Factory	1.25	0.97	0	0.02
BlogsApp.Domain	2.1	0.38	0	0.44
BlogsApp.DataAccess	1.75	0.98	0	0.01
BlogsApp.BusinessLogic	1.67	0.98	0	0.01

Figure 3.1: Métricas

En base al análisis de los valores podemos obtener algunas conclusiones: La cohesión relacional se puede definir como buena si se encuentra dentro de un rango 1.5 a 4.0. Se puede visualizar que DatAccess, IBusinessLogic y Factory se encuentran mínima aunque por debajo de la línea, por lo cual se deberá analizar sus relaciones internas y evaluar el motivo de dicho valor. El resto de los valores se puede decir que tienen un valor de alta cohesión relacional por cada assembly. En cuanto a la Inestabilidad, en este caso, el assembly BlogsApp.WebAPI tiene un valor de 1, lo que indica que no depende de otros assemblies externos, y en otro extremo tenemos el assembly IBusinessLogic siendo el que más posee dependencias externas. Por último, la abstracción es una medida de la proporción de tipos abstractos (clases e interfaces) en un assembly. Un valor de 0 indica que no hay tipos abstractos, mientras que un valor de 1 indica que todos los tipos son abstractos. En este caso, los assemblies WebAPI, BusinessLogic, Domain, Factory, DataAccess y Logging tiene un valor de 0, lo que indica que no contienen tipos abstractos.

Distancia relativa de un assembly desde el punto óptimo en el gráfico de estabilidad-abstractividad.

Assemblies	Distance
BlogsApp.WebAPI	0
BlogsApp.Logging.Logic.Services	0.05
BlogsApp.Logging.DataAccess	0.1
BlogsApp.Logging	0.25
BlogsApp.IDataAccess	0.08
BlogsApp.IBusinessLogic	0.12
BlogsApp.Factory	0.02
BlogsApp.Domain	0.44
BlogsApp.DataAccess	0.01
BlogsApp.BusinessLogic	0.01

Figure 3.2: Distance

Se observa que Domain está en la zona de dolor, pero es esperable, ya que contiene los modelos de la solución, y en la misma línea vemos a Logging, que también aplica porque es como como una representación de Dominio del proyecto de Logs.

Diagrama de ayuda para visualizar el puntaje de inestabilidad de cada paquete y sus dependencias:

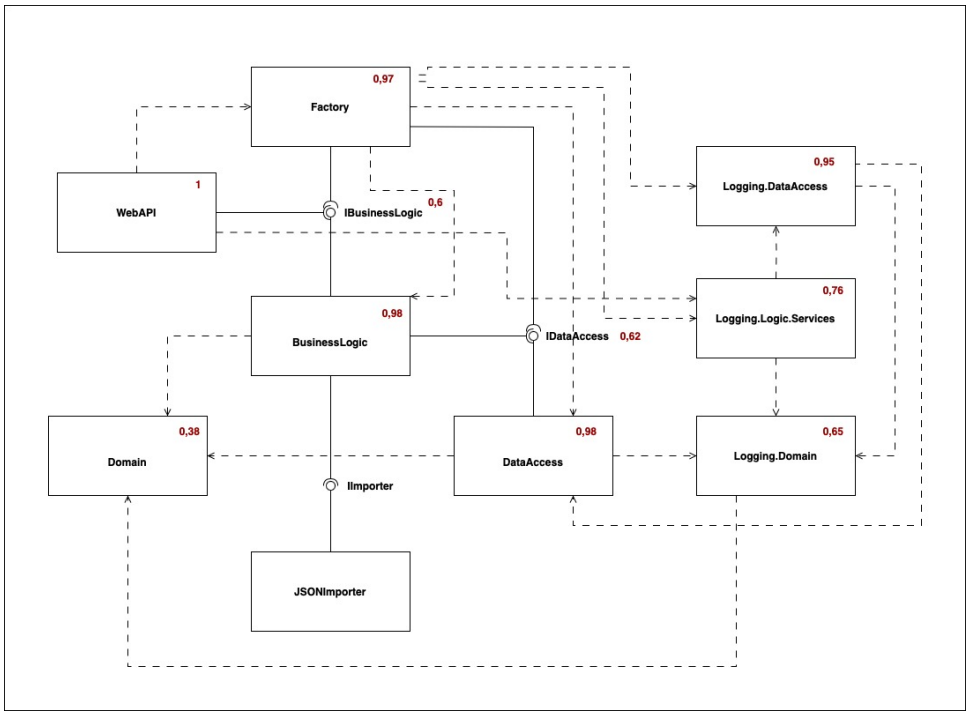

Figure 3.3: Diagrama - Métricas.

Se tiene en cuenta que se podría mejorar por ejemplo Logging Logic services que depende de DataAccess, el cual es más inestable. Se debería sacar la interfaz de lógica para afuera, y así hacer una inversión de dependencias que hiciera que todos dependan de paquetes mas estables. El resto tiene valores aceptables.

3.2 Extensibilidad

Importadores

Se añade al sistema un nuevo componente que permite a los bloggers poder importar sus artículos, sin necesidad de hacer la creación uno a uno mediante la web. Esta opción fue desarrollada internamente para que se puede realizar la importación de uno o más artículos en formato .json, pero brinda la flexibilidad de que se puedan agregar nuevos importadores, desarrollados internamente o por terceros, de forma dinámica y desde cualquier tipo de fuente.

El sistema está preparado para soportar diversos formatos, como por ejemplo: .xml, un .csv,

entre otros.

Es importante destacar que que los artículos importados pasan por el control de palabras ofensivas.

A continuación se detalla el funcionamiento:

De forma de poder cumplir con el requerimiento de importación de artículos se creó un proyecto IImporter, el cual contiene la interfaz a implementar por terceros. Esta solución implementa el patrón Strategy, donde se cuenta con la interfaz IImporterInterface, la cual implementarán los diversos importadores que sean desarrollados. Esto resulta en que el código sea flexible para añadir nuevos tipos de importadores, y que se puedan instanciar la implementación que es elegida por el usuario.

Al implementar Reflection, en nuestra solución se creó la clase ImporterLogic, la cual implementa la interfaz IImporterLogic, con el fin de contener la definición de los métodos básicos que se ejecutarán al realizar reflection.

El path de donde obtener los importers, se determinó relativo a la ubicación del proyecto WebApi, con el fin de tener fácil y flexible acceso.

A su vez se define un controller ImportController, el cual define los métodos que se encargarán de ejecutar las peticiones realizadas por el usuario. Estos endpoints son, uno para obtener los importadores disponibles, y otro para efectivamente importar. Se detalla su funcionamiento en el Anexo A - Especificación de la API.

Ejemplo de formato de archivo .json

[{

”name”: ”New article 1”,

”body”: ”Body ”,

”private”: false,

”userId”: 11,

”template”: 1,

”image”: ”image.png”

},

{

”name”: ”New article 2”,

”body”: ”Body”,

”private”: false,

”userId”: 11,

”template”: 3

}]

Es importante notar que, se incluya 1 o N artículos, el json debe estar incluído dentro de corchetes [].

3.3 Principios y patrones de diseño

Se aplica el patron de inyección de dependencias, para el cual se utilizó el framework que nos ofrece .Net para C#.

Para lograr implementarlo, se crea el módulo ServiceFactory (BlogsApp.Factory/ServiceFactory.cs), el cual será esencial en el funcionamiento. Esta ”fábrica de servicios” representa una capa de abstracción entre la aplicación y los servicios que utiliza, brindando flexibilidad en la definición de la misma.

En términos generales, se definen interfaces que describen las funcionalidades que se esperan de los diferentes servicios (y las clases implementan cada intefaz), y la aplicación utiliza la

fábrica de servicios para crear instancias de los servicios que necesita en tiempo de ejecución.

La principal ventaja de este enfoque es que la aplicación no está vinculada a ninguna implementación específica de los servicios que necesita. En cambio, la fábrica de servicios determina dinámicamente qué instancia de servicio se debe crear en función de la configuración de la aplicación. Esto permite que la aplicación sea más flexible y fácil de mantener.

- Uso de patrón Strategy para Reflection Este patrón permite agregar nuevas estrategias (en nuestro caso nuevos importadores) sin tener que modificar el código existente. Permite cargar dinámicamente las clases de los mismos en tiempo de ejecución y que de forma automática quede visible para el usuario y permita su uso, lo que brinda flexibilidad y extensibilidad a nuestra aplicación.

- Para la gestión de las notificaciones a los usuarios la cual debe realizarse de forma automática (es decir, en la UI la notifiación deberá mostrarse cuando suceda, sin la necesidad de que el usuario esté solicitando por la misma) se hace uso del patrón Observer, el cual facilita la comunicación y envío de cambios entre objetos, permitiendo que los observadores se registren y reciban notificaciones de cambios en el sujeto. Esto promueve la flexibilidad y la modularidad en el diseño del sistema.

3.4 Resumen de mejoras y cambios

- Destacamos el uso de inyección de dependencias en conjunción con el patrón de Service Factory, proporcionando una forma flexible de configurar los servicios. Esto permitió añadir nuevos sin mayor complejidad. A su vez en que modificar los existentes no es un problema.

- Se aplicó el principio de segregación de interfaces (Interface Segregation Principle, ISP) de los principios SOLID de diseño de software. Notamos que al haberlo aplicado anteriormente, la solución no tuvo mayor complejidad para enfrentarse a las nuevas funcionalidades y reestructuras implementadas. Al haber definido interfaces tanto para la Lógica de Negocio, como para el acceso a datos, se volvió sencillo el cambio de código cuando por ejemplo, métodos de lógica necesitaron reestructura, ya que los componentes que las usan solo conocen la firma de los

métodos que utilizan.

Se comprobó la modularización y por tanto flexibilidad c con la que cuenta la aplicación, disponiendo de módulos y clases independientes y abiertas al cambio.

- Se elimina la clase Reply, la cual buscaba representar un Comment, y se usa solo la última para gestionar estos objetos. Esto nos permitió reducir la cantidad de tablas en la base de datos, a la vez en la cual consideramos que es más acorde conceptualmente a la lógica de negocio, y sirve para definir correctamente la funcionalidad requerida de hilo de comentarios.

Apéndice A

A.1 Especificación de cambios de la API

A continuación se detallan aquellos endpoints que tuvieron cambios en esta segunda versión:

Comment

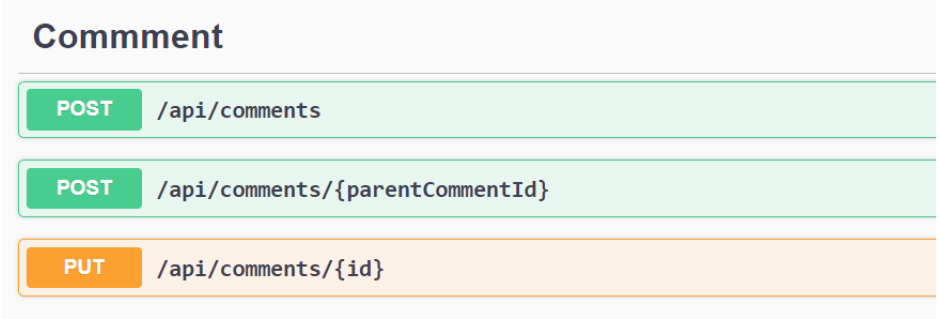

Figure A.1: Comment - Endpoint

POST /api/comments/parentCommentId

- Objetivo: Obtener los comentarios de un comentario padre.
- Parámetros:
  - Token
  - parentCommentID - Id de comentario padre
- Body: { ”body”: ”string”,

  ”articleId”: 0,

  ”state”: 0,

  ”message”: ”string”,

  ”offensiveWords”: [

”string”

] }

- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

PUT /api/comments/Id

- Objetivo: Modificar el texto de un comentario realizado
- Parámetros:
  - Token
  - Id - Identificador de comentario
- Body:

{ ”body”: ”string”

}

- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

User

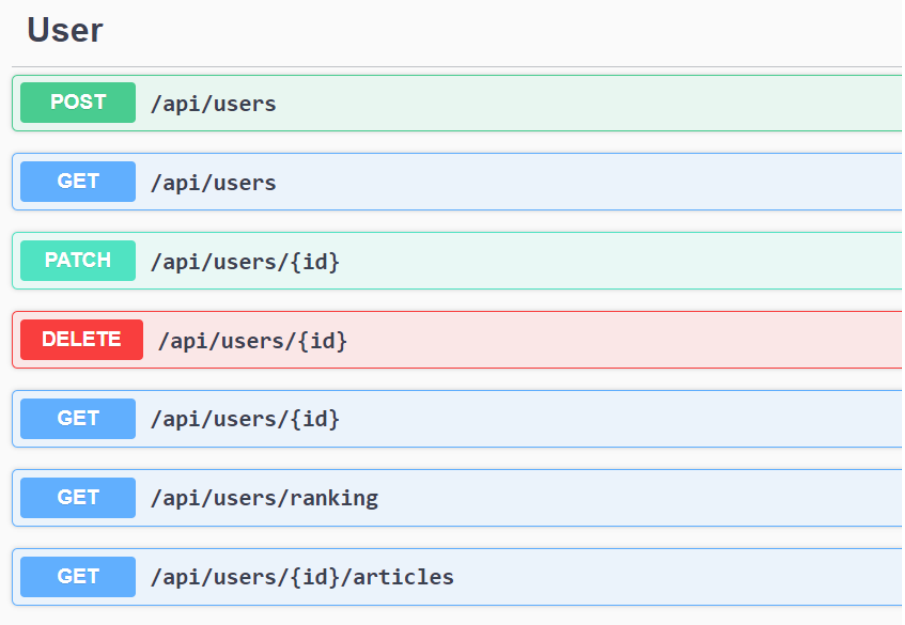

Figure A.2: User - Endpoint

GET /api/users

- Objetivo: Obtener un listado de todos los usuarios existentes. Solo lo puede realizar el administrador.
- Parámetros:
  - Token
- Body: No aplica.
- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

GET/api/users/Id

- Objetivo: Obtener información sobre el usuario consultado. El administrador puede obtener información de cualquier usuario, mientras que el resto solo puede obtener información de si mismo.
- Parámetros:
  - Token
  - Id - Identificador de usuario
- Body: No aplica.
- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

GET/api/users/ranking

- Objetivo: conocer el ranking de usuarios entre dos fechas que más publicaron artículos y comentarios con al menos una palabra ofensiva.
- Parámetros:
  - Token
  - Id - Identificador de usuario
  - DateFrom - DateTime
  - DateTo - DateTime
  - Top - cantidad de usuarios que se desea obtener en el resultado del ranking
  - WithOffensiveWords - True: Determina que se quiere incluir la opción de que se tengan en cuenta las palabras ofensivas. False: Determina que no se tienen en cuenta palabas ofensivas.

- Body: No aplica.
- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

Offensive Words

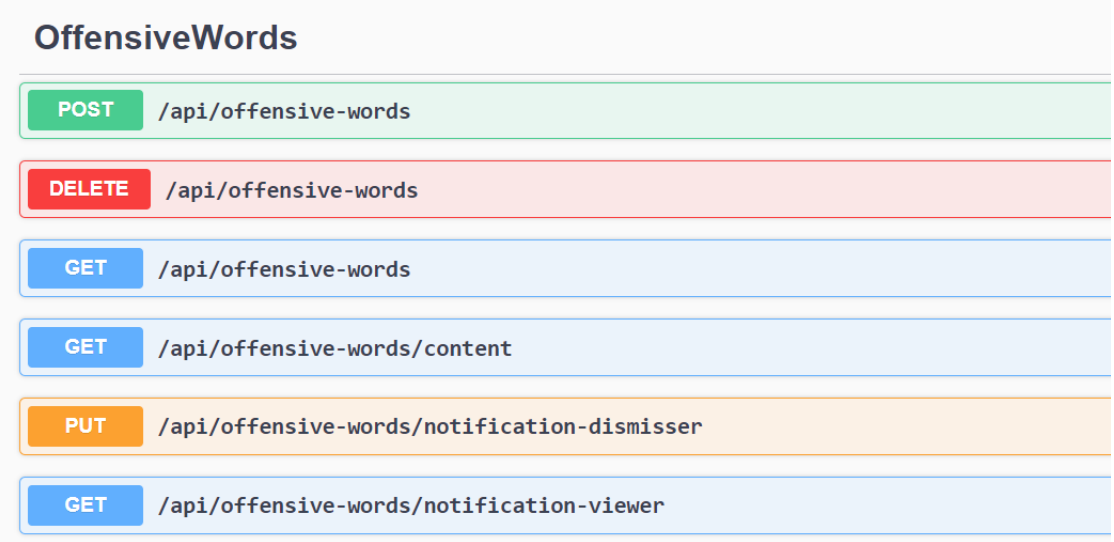

Figure A.3: Comment - Endpoint

POST api/offensive-words

- Objetivo: Añadir nuevas palabras a la lista de palabras ofensivas.
- Parámetros:
  - Token
- Body: { ”word”: ”string” }
- Responses:

  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

DELETE /api/offensive-words

- Objetivo: Eliminar una palabra de la lista de palabras ofensivas.
- Parámetros:
  - Token
- Body: { ”word”: ”string” }
- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

GET /api/offensive-words/content

- Objetivo: Obtener lista de contenido que se encuentra en revisión.
- Parámetros:
  - Token
- Body: No aplica.
- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

Import

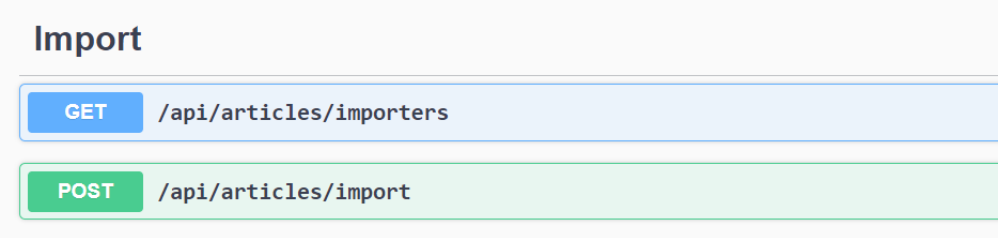

Figure A.4: Comment - Endpoint

GET/api/articles/importers

- Objetivo: Obtener lista de importers disponibles.
- Parámetros:
  - Token
- Body: No aplica.
- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

POST /api/articles/import

- Objetivo: importar uno o N artículos.
- Parámetros:
  - Token
- Body: {

”importerName”: ”string”,

”path”: ”string”

}

- Responses:
  - 200 Success
  - 404 Not Found
  - 401 Unauthorized

A.2 Informe de cobertura

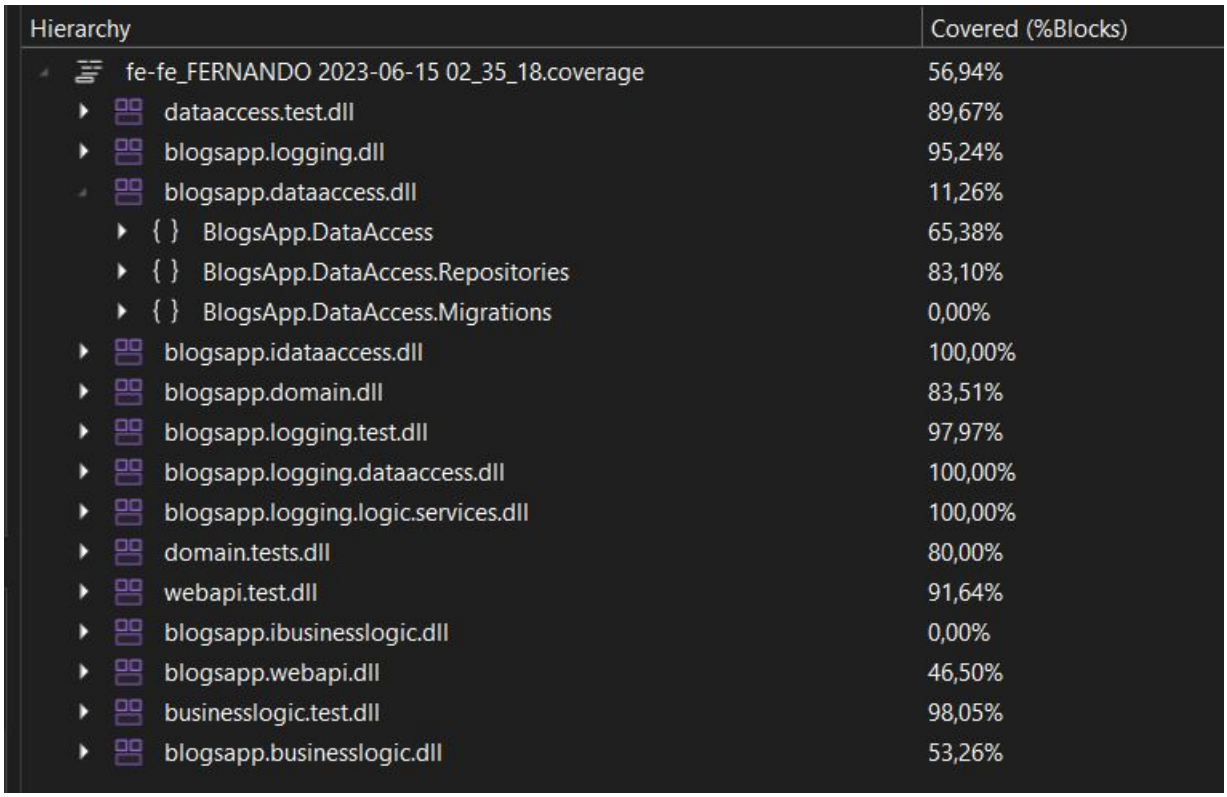

Figure A.5: Resultados cobertura

El porcentaje de cobertura de esta entrega en general es menor que la entrega anterior, ya que por un tema de tiempo se priorizó el desarrollo de funcionalidades que habían quedado pendientes, y no se realizaron algunos tests sobre algunas partes del código. Destacamos que la cobertura

total con 56,94 % no refleja la realidad del código, ya que en el mismo se incluye la cobertura de los propios tests, migraciones, y archivos que no hacen al código de la solución. Por otro lado notamos que en Domain la cobertura es de 82,51 %. La cobertura de Logging es de 100 % y la de DataAccess de 83,10%.

A.3 Datos de prueba

En la carpeta de la solución se adjunta un archivo .bak, el cual contiene una base de datos cargada con valores que servirán para las consultas a realizar.

Se detalla el contenido:

Usuarios:

Username	Password	Admin	Moderador	Blogger
admin	12345	si	no	no
Juan	12345	no	si	no
Pedro	12345	no	no	si
Juana	12345	si	no	si
Monica	12345	no	si	si

Palabras ofensivas:

Hola
Que
Tal

Articulos:

Nombre	Plantilla	Contiene palabras ofensivas	Está en revision	Fue editado	Autor	Está borrado	Es priva
Articulo1 Hola	1	Si	Si	No	Monica	No	No
Articulo2	2	Si	Si	Si	Juana	No	No
Articulo3	3	No	No	Si	Pedro	No	No
Articulo4	4	No	No	No	Pedro	No	Si
Articulo5	1	No	No	No	Pedro	Si	No

Comentarios en articulos:

Articulo	Comentario	Fue esditado	Esta en revision
Articulo3	Hola	No	si
Articulo3	Ey	Si	No
Articulo3	Bueno	No	No

Figure A.6: Datos de prueba