

**Facultad de Ingeniería**

## **Diseño y documentación**

Entregado como requisito del segundo obligatorio de Diseño de Aplicaciones 2

<https://github.com/ORT-DA2/194257-205110>

**Nahuel Kleiman - 205110**

**Federico Jacobo - 197254**

**Docentes: Gabriel Piffareti y Nicolas Fierro**

**2020**

**Índice**

<b>Descripción general del trabajo</b>	<b>4</b>
<b>Descripción del diseño</b>	<b>5</b>
<b>Funcionalidades</b>	<b>7</b>
Errores conocidos	8
<b>Mejoras del diseño</b>	<b>9</b>
<b>Vista de Componentes o de Implementación</b>	<b>10</b>
Diagrama de descomposición de los namespaces del proyecto	10
Diagrama general de paquetes (namespaces)	11
Diagrama de implementación (componentes)	12
<b>Vista Lógica o de Diseño</b>	<b>13</b>
Diagrama de clases: BusinessLogic	13
Diagrama de clases: DataAccess	14
Diagrama de clases: Entities	16
Diagrama de clases: Controllers	17
Diagrama de clases: IDataImporter	17
<b>Vista de Procesos</b>	<b>19</b>
Diagrama de secuencia: Alta de solicitud	19
Diagrama de secuencia: Método de importación	20
Diagrama de secuencia: Reporte B	21
<b>Vista Física o de Despliegue</b>	<b>22</b>
Diagrama de despliegue/deployment	22
<b>Informe de métricas</b>	<b>24</b>
Cohesión	24
Inestabilidad	25
Abstracción	26

Abstracción vs. Inestabilidad	27
<b>Anexo I: Modelo de tablas de la estructura de la base de datos</b>	<b>29</b>
<b>Anexo II: Análisis de cobertura de pruebas</b>	<b>30</b>
IMMRequest.BusinessLogic	30
IMMRequest.DataAccess	31
IMMRequest.Entities	31
IMMRequest.WebApi	32
<b>Anexo III: Documento actualizado de la API</b>	<b>33</b>
Criterios utilizados para cumplimiento de REST	33
Descripción del mecanismo de autenticación de requests	34
Descripción general de los códigos de error	34
Especificación de Resource Session de la API	36
Especificación de Resource User de la API	37
Especificación de Resource Request de la API	39
Especificación de Resource TypeReq de la API	44
Especificación de Resource AdditionalField de la API	46
Especificación de Resource Import de la API	48
<b>Anexo IV: Capturas de pantalla del Frontend</b>	<b>49</b>
Reportes A y B	49
Ingreso de nueva solicitud	50
Listado de solicitudes	51
Modificación de usuario	53
Cambio de estado de solicitud	53
Alta de tipo de solicitud con campos adicionales	54

## Descripción general del trabajo

Para la realización de la segunda entrega de IMMRequest, se utilizó nuevamente un repositorio distribuido con la herramienta GitHub. En este repositorio se encontrarán dos aplicaciones: la primera es una WebApi REST, como la que entregamos en la primera entrega, pero contiene varias mejoras basadas en los cambios solicitados en la corrección. La segunda es una aplicación frontend que se encarga de consumir esta API.

El backend se realizó con .NET Core y el frontend con Angular, utilizando las herramientas que nos fueron brindadas en clase. Dado que para la primera entrega ya teníamos hecha la mayor parte de nuestro desarrollo del backend, se continuó realizando la técnica Code First.

Todo el desarrollo del backend fue realizado siguiendo los pasos estipulados de una estrategia de Test-Driven Development (TDD):

- Se codifica un conjunto de pruebas para un determinado módulo
- Commit [RED]
- Se escribe el código para que el conjunto de pruebas pase y se logre hacer un build de la solución
- Commit [GREEN]
- Se refactoriza código si es necesario
- Commit [REFACTOR]

Además, utilizamos git flow como técnica de dirección y gestión de versiones. Nos informamos bastante sobre este tema para poder cumplirlo de manera de adaptarlo a las condiciones en las que nos encontrábamos. Por un lado, dejamos la rama master únicamente para realizar las releases (nuestra rama de producción), y creamos la rama develop, sobre la que ocurre la mayoría del trabajo y es nuestra rama de desarrollo. Luego, tenemos todas las que llamamos feature branches que facilitan que hayamos podido trabajar en distintas funcionalidades simultáneamente, siempre y cuando trabajemos en distintas feature branches. Cada una de estas branches son luego mergeadas con develop, donde se concentra el flujo central del trabajo.

## Descripción del diseño

La aplicación permite realizar solicitudes a la Intendencia de Montevideo. Estas solicitudes corresponden a un área, que corresponde a un tema y que este último tiene un tipo.

Para realizar el diseño de este sistema, comenzamos por pensar cómo iba a ser nuestro modelo de dominio. Dado que existen dos tipos de usuarios pero que solamente difieren en administradores y ciudadanos y a partir de esta calificación se los divide en las funciones que realizan, tuvimos dos opciones a la hora de diferenciarlos: mediante una jerarquía (utilizando herencia), creando por ejemplo una clase Persona, y dos clases hijas Ciudadano y Administrador, o por medio de una única clase Usuario, que diferencie a ciudadanos y administradores con una bandera booleana. Dada la poca complejidad que vimos entre ambos tipos de usuarios, ya que ambos contaban con exactamente las mismas propiedades, decidimos seguir con la segunda opción, para simplificar el sistema. Es por esto que utilizamos la bandera IsAdmin en la clase UserEntity para diferenciar los roles de un usuario. Esta fue la única oportunidad en la que vimos como una opción la utilización de herencias, por el simple hecho de que no creímos necesario que la complejidad de este sistema las requiriera, y nos pareció más simple manejarlos sin la utilización de las mismas en esta oportunidad.

La clase RequestEntity es la que tiene la información de una solicitud realizada por un usuario, sea éste un usuario registrado o no. Dentro de esta clase, hay dos atributos que vale la pena explicar cómo manejamos. Por un lado, el Status representa al estado de una solicitud, y lo manejamos con un string, pero para validar que éste string es correcto (es decir, que pertenece a uno de los estados posibles) lo chequeamos con un enum llamado RequestStatusEnum, que tiene seteados los estados posibles de una solicitud. Por otro lado, el atributo que representa a los valores de campos adicionales de una solicitud se llama AdditionalFieldValues y también lo manejamos con un string. Cuando un usuario va a setear los valores de los campos adicionales de una solicitud, tiene que primero saber cuáles y de qué tipo son estos campos adicionales, para luego escribirlos separados por comas. Nuestra lógica de negocio se encarga de realizar las validaciones necesarias y de “splitear” este string y adjudicar los valores ingresados a los campos adicionales correspondientes.

Además, a una solicitud le corresponde un tipo de solicitud (lo que en el código se llama TypeReqEntity). Este atributo lo manejamos con un entero, haciendo que cada solicitud tenga el identificador numérico del tipo al que corresponde.

La estructura de entidades AreaEntity - TopicEntity - TypeReqEntity está conformada por dichas clases y se comporta de manera que existen dos relaciones one to many. Una entre Áreas y Topics, y otra entre Topics y TypeReqs:

- Un área tiene muchos temas, y un tema pertenece a una sólo área.
- Un tema tiene muchos tipos, y un tipo pertenece a un sólo tema.

A su vez, un tipo tiene una lista de campos adicionales, lo que en nuestro código se puede encontrar como la entidad `AdditionalFieldEntity`. Esta entidad tiene dos atributos que debemos explicar para que se entienda su funcionamiento. Por un lado tenemos una propiedad `Type` de tipo `string`, que corresponde al tipo del campo adicional, que puede ser fecha, texto, entero o `bool`. Al igual que con el `Status` de las solicitudes, es una propiedad que manejamos con un `string` pero que validamos su correctitud con un `enum` que se llama `AdditionalFieldType`. Seguidamente, tenemos una propiedad `Range` que hace referencia al rango que puede tener el valor de este campo adicional. Por ejemplo, si es una fecha, su rango estará entre dos fechas; si es un entero, su rango será entre dos números; si es un booleano, su rango será `true` o `false`; etc. Este campo también lo manejamos con un `string`, y a través de nuestra lógica de negocio se realiza un “parseo” dependiendo del tipo de campo adicional del que se trate.

Entre las entidades `AdditionalFieldEntity` y `TypeReqEntity` también existe una relación `one to many`, en la que un tipo tiene varios campos adicionales, pero un campo adicional corresponde a un único tipo.

Todas estas relaciones `one to many`, las manejamos teniendo en cuenta que `.NET Core` es capaz de identificarlas de varias formas. Nosotros optamos por diseñar la que nos parece que tiene un modelo más intuitivo, en la que, por ejemplo, del lado de las Áreas tenemos un atributo con una lista de temas, y del lado de `Topics` tenemos un atributo de identificador numérico del área al que corresponde. Seguimos este modelo con el resto de las relaciones que ya mencionamos.

## Funcionalidades

La principal funcionalidad de un usuario ciudadano es la de realizar una solicitud sin necesidad de estar registrado en el sistema ni autenticado en el mismo. Para realizar esta funcionalidad, lo hicimos a través de la lógica de las solicitudes. La decisión de ubicar esta funcionalidad en esta clase se debe a que una solicitud puede estar ligada a un usuario propiamente dicho o no, dependiendo si éste está registrado. Al ingresar los datos de una solicitud, se especifica un email, y a través de la lógica es que se verifica en la base de datos si existe algún usuario con ese email. Si existe, entonces se le adjudica esta solicitud al usuario, y si no existe el usuario, se crea la solicitud sin usuario asociado. Por el contrario, si el email introducido sí pertenece a un usuario registrado en el sistema, entonces nuestra solución va a crear la relación entre estas entidades.

Las funcionalidades de un usuario ciudadano las puede realizar cualquier tipo de usuario, ya que en nuestro sistema la única distinción que hicimos fue ver si un usuario es o no administrador. Un usuario administrador puede realizar todas las funcionalidades que especifica la letra del obligatorio. Y es por esto, que aquellas funcionalidades que no se permiten para usuarios ciudadanos, cuando se intente realizarlas se mostrará un mensaje de error aludiendo a que se necesita ser administrador para realizar tal operación. Para lograr esto utilizamos un Authorization Filter.

Además de estas funcionalidades y de las ya entregadas en la primera entrega, se implementaron las siguientes nuevas funciones que fueron pedidas para esta entrega.

Los administradores pueden acceder a dos tipos de reportes:

- El Reporte A, permite que dado el mail de un usuario, se obtienen todas las solicitudes que éste realizó en un determinado período de tiempo (fecha y hora en los que la solicitud fue creada). El reporte obtenido contendrá las solicitudes que se hayan creado entre las fechas ingresadas, y serán agrupadas por su estado y especificando la cantidad por estado.
- El Reporte B, permite obtener los Tipos más usados por todos los usuarios para crear solicitudes dentro de un período de tiempo (fecha en que la solicitud fue creada).

Existe un nuevo tipo de datos para un campo adicional. Además de los que había para la primera entrega (fecha, texto y entero), ahora se suma el tipo bool que admite un rango de valores entre true y false.

Posibilidad de agregar campos adicionales con múltiples valores. Nuestra aplicación soporta el ingreso de campos adicionales de un tipo que acepta más de un valor, y dichos valores pueden tener un rango definido o no.

Se ofrecen mecanismos de importación de estructuras de áreas, temas y tipos. Esta funcionalidad viene con dos tipos de lecturas, para archivos xml y json, pero la

realizamos con tal extensibilidad que permite dar soporte a otros tipos de archivo o mecanismos de importación.

## **Errores conocidos**

En relación al manejo de excepciones, utilizamos bloques try-catch para evitar que nuestra solución “cayera” ante errores de ingreso de datos u otro tipo de errores. De cualquier manera, hay un uso injustificado de la excepción `ArgumentException` ya que hay casos en los que no se tiene nada que ver con argumentos, y el no uso de excepciones específicas para el manejo de errores, no es una buena práctica. Lo que sí hicimos fue una distinción en el mensaje de error que se le muestra al usuario, para que éste entienda qué está sucediendo. Sabemos que la solución no tiene un buen manejo de excepciones distintivas con errores específicos, lo que ayudaría también a respetar estándares de Clean Code, pero en su momento priorizamos otros aspectos del diseño.

Otro error o funcionalidad no implementada en esta segunda entrega, es que no realizamos la implementación del frontend para la extensibilidad de mecanismos de importación. Es decir, la funcionalidad está implementada en el backend y es funcional, pero no nos dio el tiempo para implementar un componente en Angular que se conecte con la lógica del backend y se permita interactuar con la misma a través del frontend. De cualquier manera, para probar esta funcionalidad, se puede recurrir a Postman. Se dejaron dos archivos (`jsonFile.json` y `xmlFile.xml`) con una estructura de áreas, temas y tipos cargada para que se puedan probar ambos importadores.

El ingreso de múltiples valores en los campos adicionales al momento de crear una request, es otra funcionalidad que está implementada y su funcionalidad es correcta en el backend, pero que no la pudimos plasmar en el frontend.



## Mejoras del diseño

Para esta segunda entrega se realizaron algunas mejoras en el diseño del sistema. Estas mejoras no fueron muchas en cantidad, aunque sí algunas fueron drásticas ya que mejoraron enormemente la calidad de diseño de la solución si la comparamos con nuestra primera entrega. Principalmente nos enfocamos en cumplir con la implementación de las nuevas funcionalidades, sin perder el foco en posibles mejoras.

En la primera entrega, contábamos con dos paquetes muy parecidos, Entities y Domain, e hicimos mención de esta repetición en la pasada documentación. Para esta segunda entrega, entendimos que esta separación entre dos paquetes es más indicada cuando se tiene un modelo de dominio del que no se desea guardar toda su información, o se la quiere guardar de una forma distinta, y por eso se termina realizando un mapeo a otro paquete con el fin de guardar éste último en la base de datos.

En conclusión, decidimos borrar el paquete Domain, y quedarnos únicamente con Entities para que sea éste el que represente nuestro modelo de dominio. Esto llevó un gran trabajo de refactorización de código, pero permitió reducir el acoplamiento entre las capas y darle más simplicidad al código.

El segundo cambio que realizamos, que fue conllevado por el borrado del paquete que mencionamos recién, fue el de borrar el paquete Mapper, ya que al no tener más los dos paquetes (Domain y Entities), el Mapper ya no tenía sentido.

De la manera en la que habíamos diseñado la solución en la primera entrega, el mapeo se realizaba dentro de la capa de la lógica de negocios en lugar de hacerlo en la capa de acceso a datos, lo que generaba una dependencia entre el componente BusinessLogic y el componente IMMRequest.Mapper que no estaba del todo justificada. Esto ahora ya no sucede, gracias a que no es necesario un mapeo.

Otro de los cambios y mejoras que implementamos para esta segunda entrega fue corregir esto el action filter que habíamos utilizado para la primera entrega y hacer la autenticación de usuarios con un authorization filter, que son específicos para políticas de autorización y seguridad. Estos filtros tienen la ventaja de que se ejecutan antes que cualquier otro filtro y permiten evitar llegar al controller en caso de que no se cumpla con las políticas de seguridad.

Finalmente, el diseño de la nueva funcionalidad de mecanismos de importación también aporta extensibilidad a la solución, de manera que utilizando Reflection pudimos encontrar la forma de que se pueda brindar a desarrolladores terceros una interfaz. Este paquete que contiene a la interfaz se denomina IMMRequest.IDataImporter y está explicado con detalle más adelante, pero la decisión de diseño de ubicar la interfaz en un nuevo paquete, y no hacerlo en un paquete ya existente, permite no violar el principio de reuso común y el principio de equivalencia de reuso.



## Diagrama general de paquetes (namespaces)

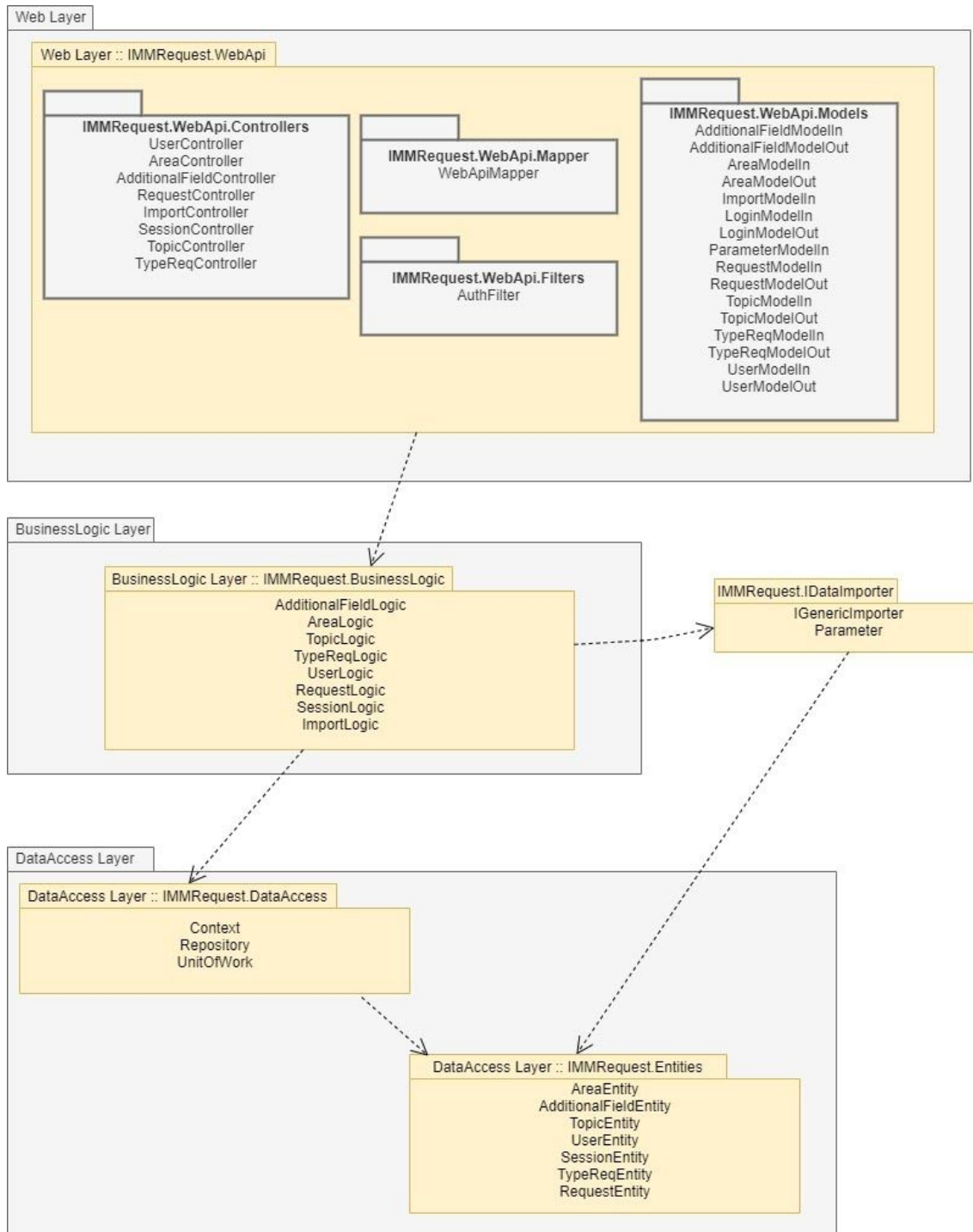


Figura 2: Diagrama general de paquetes (namespaces)

En la Figura 2 vemos un diagrama de paquetes general de la solución, en donde se muestran los namespaces organizados dentro de sus capas (layers). Como se puede observar, pudimos lograr un bajo acoplamiento entre las capas de nuestro sistema.

También mostramos las clases que se pertenecen a cada uno de estos paquetes para que el lector logre una mayor comprensión.

Como se puede observar en el diagrama, cada paquete contiene clases que son pertinentes únicamente a la funcionalidad de ese paquete en sí mismo. Es decir, intentamos agrupar todas las clases que tuvieran que ver con un mismo tipo de funcionalidad a un mismo paquete, y de esta manera también se puede notar que se cumple con el principio de clausura común (CCP), ya que todas las clases pertenecientes a un paquete tienen el mismo tipo de cambio.

Tuvimos este enfoque por el hecho de tener cuidado de no generar tensión entre los principios de paquetes, entonces primero nos enfocamos en el mantenimiento y cumplir CCP y luego analizar si podíamos reusar.

### Diagrama de implementación (componentes)

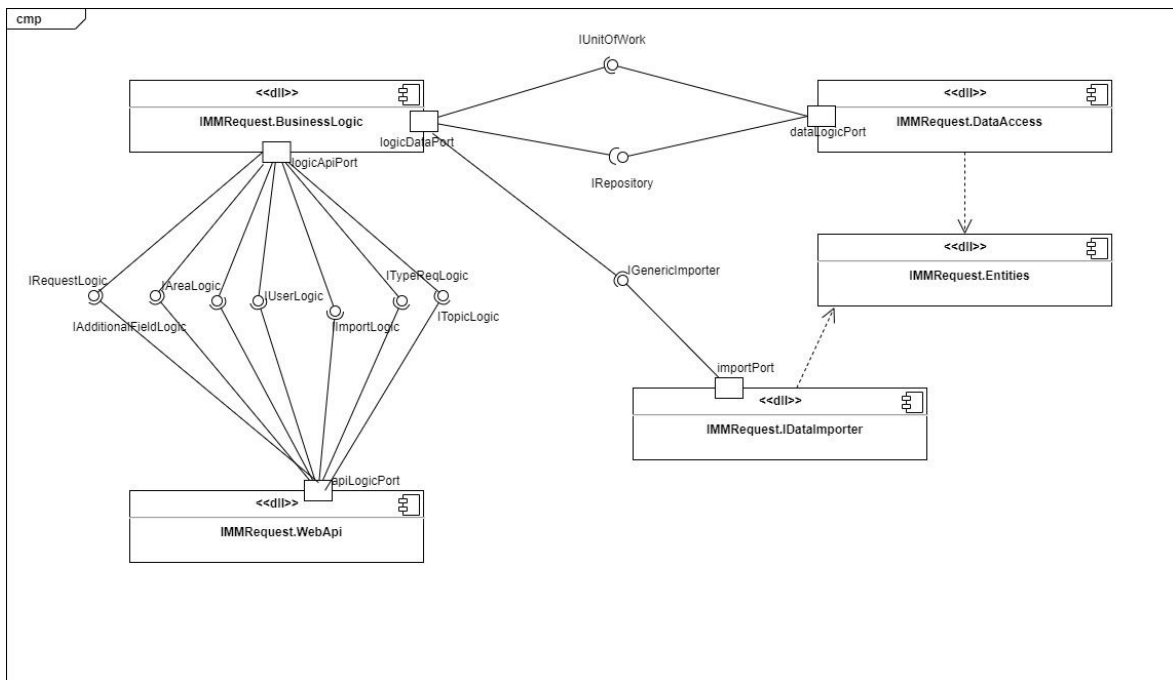


Figura 3: Diagrama de componentes del sistema

En la figura 3 se puede ver el diagrama de componentes del sistema, donde se describen los mismos y cómo dependen unos de otros. En este diagrama se puede ver cómo la api depende mayoritariamente de interfaces. En este diagrama se muestran 5 componentes que son los que nos parecieron pertinentes para ilustrar el comportamiento del sistema. Además hay componentes de testing que no son necesarios para el diagrama.

Con respecto a la primera entrega, vemos una primera diferencia en cuanto al diseño cuando se nota la desaparición de los componentes **IMMRequest.Domain** y **IMMRequest.Mapper**, que luego entraremos más en detalle sobre el por qué de esta

decisión. Otra diferencia es la aparición de un componente IMMRequest.IDataImporter, cuya función es almacenar el contrato que deban cumplir aquellos desarrolladores externos que deseen atribuir (importar) datos al sistema. La decisión de diseño de realizar un componente separado para esto, se toma en base a que cuando se entregue a desarrolladores externos este contrato (interfaz) que con una implementación deben cumplir, únicamente se les puede dar a conocer este componente, y el componente IMMRequest.Entities, ya que es utilizado por el que tiene el contrato. Entonces, al entregar el contrato a desarrolladores externos, no tendremos que enseñarles todo nuestro código o sistema, sino solamente aquellas porciones pertinentes a los asuntos que ellos deberán implementar.

## Vista Lógica o de Diseño

### Diagrama de clases: BusinessLogic

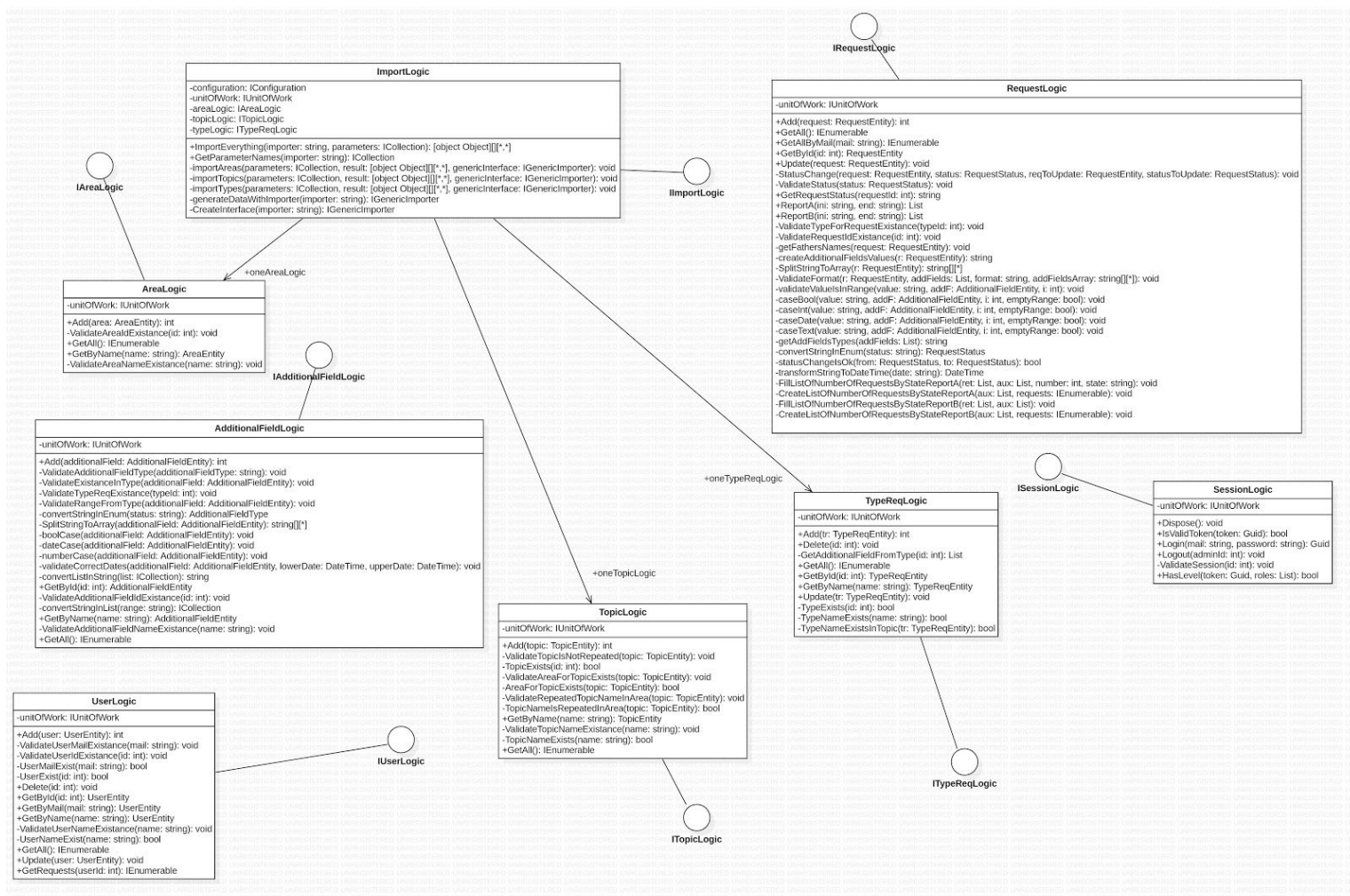


Figura 1.3: Diagrama de clases del paquete IMMRequest.BusinessLogic

En la Figura 1.3 se aprecia el diagrama de clases del paquete IMMRequest.BusinessLogic. Este paquete se encarga de manejar toda la lógica de negocio de nuestra solución, por lo que tiene clases pertinentes para implementar las funcionalidades que ofrece el sistema. Cada una de estas clases que se muestran en el diagrama, están implementando su interfaz que se encuentra en el namespace IMMRequest.BusinessLogic.Interfaces.

En este diagrama UML incluimos las clases de IMMRequest.BusinessLogic.Interfaces y IMMRequest.BusinessLogic para representar claramente cómo se comportan las clases de los mismos. Es en este paquete donde se encapsula la mayor parte de la lógica de negocios, por lo que lo consideramos el paquete de mayor nivel.

Otro punto a destacar, es que cada clase Logic implementa una interfaz distinta que no posee ningún método público específico para la implementación, lo que resulta beneficioso cuando luego estas dependencias son inyectadas en los controllers como interfaces, lo que permite un gran desacoplamiento de capas.

Un punto no tan favorable, es que en la clase ImportLogic se tiene asociaciones con otras tres clases de este paquete, lo que probablemente provoque que no se cumpla completamente el principio de responsabilidad única.

### Diagrama de clases: DataAccess

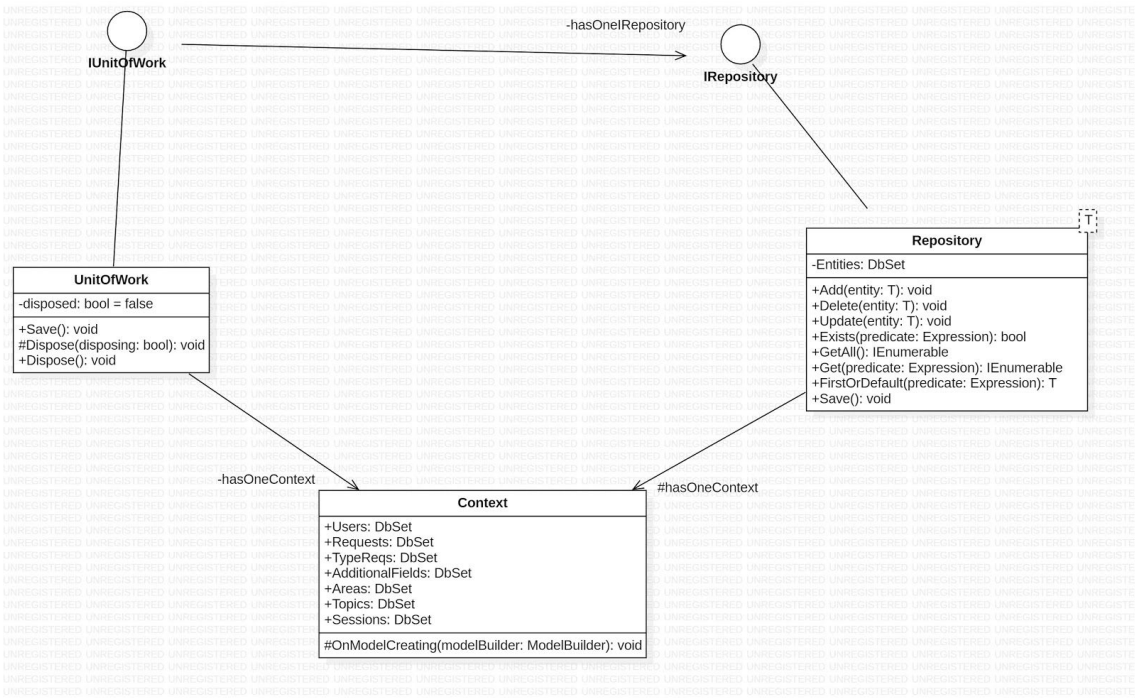


Figura 4: Diagrama de clases del paquete IMMRequest.DataAccess

En la Figura 4, vemos el diagrama de clases del namespace IMMRequest.DataAccess. Cabe destacar que este paquete, al igual que el anterior, también posee un subpaquete (o namespace) llamado IMMRequest.DataAccess.Interfaces, donde están las interfaces IRepository e IUnitOfWork. Ambas interfaces se encuentran implementadas en las clases Repository y UnitOfWork respectivamente, y están diagramadas en la figura.

Este paquete, se encarga como su nombre lo indica, del acceso a datos, por lo que tiene únicamente la responsabilidad de realizar la conexión con la base de datos y todas las operaciones con la misma. Es por esto que tiene una única dependencia y es claramente con el paquete donde se almacenan las entidades, IMMRequest.Entities.

Además, se observa la implementación de los patrones Repository Pattern y Unit Of Work, ambos utilizados con la intención de formar una capa de abstracción entre la lógica de negocio y la persistencia de datos.

El patrón Unit Of Work nos permite coordinar las transacciones sobre la base de datos que necesitan trabajar con un mismo contexto. Habiendo utilizado este patrón, utilizamos siempre la instancia de nuestra unidad de trabajo para acceder a la base de datos, indicando el repositorio concreto que deseamos utilizar. Especificamos una interfaz de repositorio genérico en la que se encuentran los métodos que utilizamos para todos los repositorios de todas las entidades. Dado que la complejidad de estructuras de nuestro modelo dominio no requería distintas implementaciones de esta interfaz, existe una única implementación llamada Repository, que implementa los métodos de IRepository con un tipo T genérico.

También, con el fin de poder realizar inyección de dependencias en las siguientes instancias de la implementación del sistema, fue que creamos la interfaz IUnitOfWork, dentro de la cual se encuentran todas aquellas instancias de IRepository necesarias para el acceso a datos de cada entidad. Luego, la clase que implementa es UnitOfWork, que tiene los métodos que devuelven las instancias de estos repositorios.

## Diagrama de clases: Entities

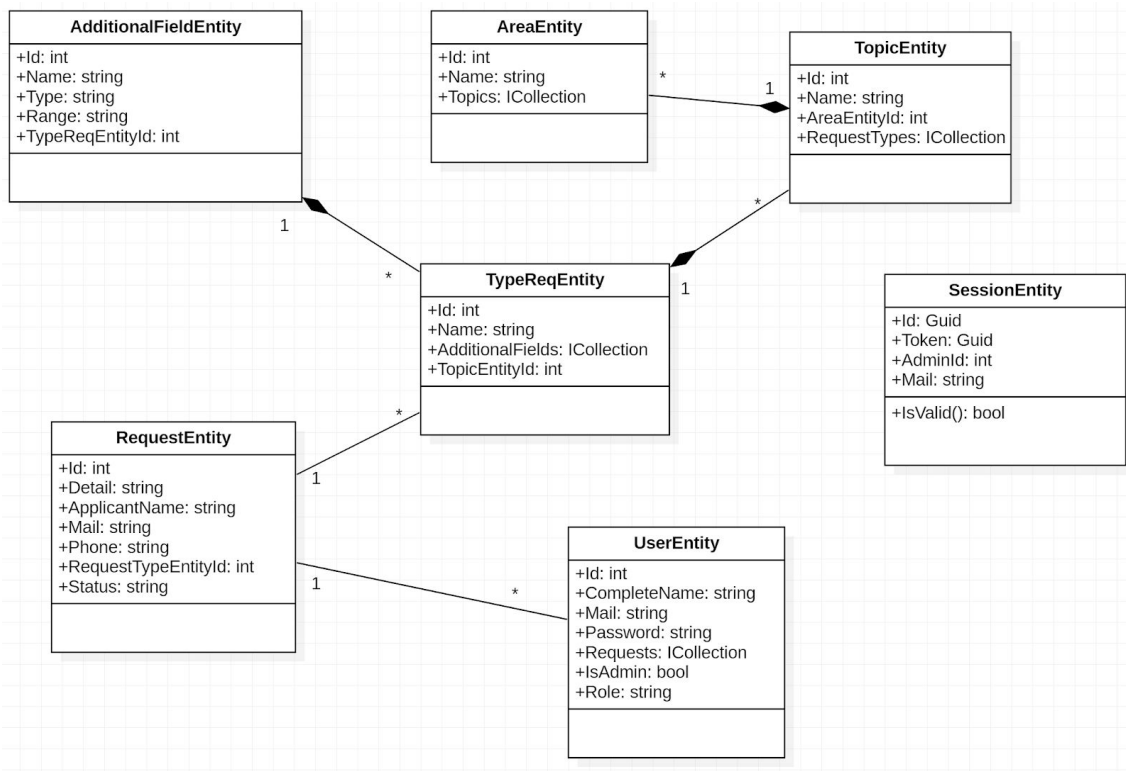


Figura 5: Diagrama de clases del paquete IMMRequest.Entities

En la Figura 5 se muestra el diagrama de clases del namespace IMMRequest.Entities. Este paquete tiene la responsabilidad de modelar el dominio de nuestro negocio o sistema. Tiene las clases que representan los elementos de la realidad en nuestro código y que nos permiten manejarlos. Este modelo de entidades son lo que luego es llevado a nuestra capa de negocios y guardado en base de datos.



## Diagrama de clases: Controllers

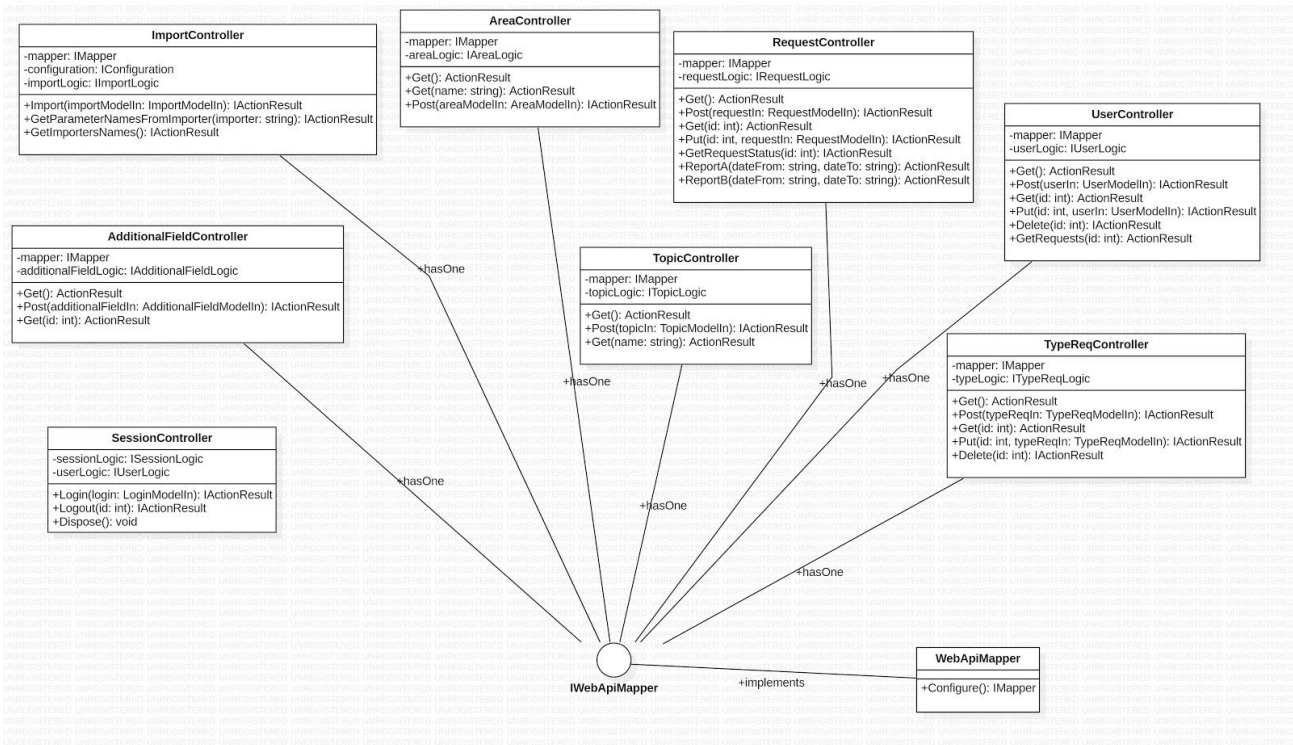


Figura 6: Diagrama de clases del paquete IMMRequest.WebApi.Controllers

En la Figura 6 se ve el diagrama de clases del paquete `IMMRequest.WebApi.Controllers`, y su dependencia con el paquete del mapper de la api, cuya funcionalidad es la de realizar un mapeo entre los modelos. Además, un punto a destacar es la independencia entre los distintos controllers y cómo todos utilizan interfaces para instancias a las clases Logic y al propio mapper.

## Diagrama de clases: IDataImporter

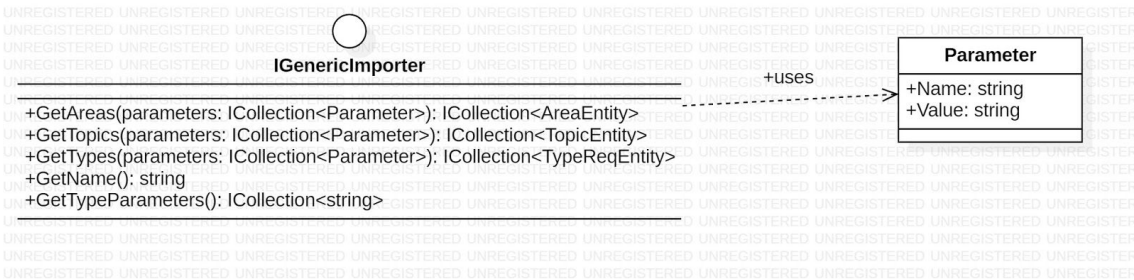


Figura 7: Diagrama de clases del paquete IMMRequest.IDataImporter

Este último paquete `IMMRequest.IDataImporter` contiene la interfaz y la clase utilizadas para brindar a los desarrolladores externos la posibilidad de crear métodos de importación propios. Los primeros tres métodos de la interfaz, `GetAreas`, `GetTopics` y `GetTypes`, son los métodos principales que el desarrollador tendrá que implementar para realizar la importación.

Dado que las variables en cuanto a métodos de extensibilidad de la aplicación y no las podemos prever, se nos ocurrió pasarles por parámetro a estos tres métodos una lista de `Parameter`. Esta clase es simplemente la representación de un parámetro genérico, que se compone de dos strings, uno para el nombre y otro para el valor. Esta clase genérica `Parameter`, nos permite que el contrato sea sumamente extensible.

Además, también se encuentran otros dos métodos: `GetName` es un método que permite obtener el nombre del importador (siendo éste `Json` o `Xml` para esta entrega) para poder seleccionar el tipo de importación que se quiere realizar. Por otro lado el método `GetTypeParameters` es un método que se utilizará en el front de la aplicación, donde se recibirá toda la información sobre los parámetros necesarios para cada importador en particular.

Nuestra aplicación está diseñada para poder agregar nuevas áreas, temas y tipos al sistema, utilizando importadores desarrollados por usuarios externos sin necesidad de que se tenga que recompilar la solución. Para lograr esto, es necesario agregar los componentes necesarios (dll) en una carpeta que definimos dentro del paquete `IMMRequest.WebApi` llamada `Assemblies`. Para la demostración de la funcionalidad, siguiendo la letra del obligatorio, fueron creados dos importadores que implementan el contrato especificado anteriormente, uno que importa una estructura de datos desde un archivo json y otro desde un archivo xml. Sus dll ya se encuentran en la carpeta mencionada, así como también para la entrega incluimos dos archivos, un `.json` y un `.xml` para la correcta demostración de esta funcionalidad.

Para mejorar la usabilidad, también desde el front existe una funcionalidad que permite obtener todos los nombres de los componentes que se encuentran en la carpeta `Assemblies`.

Todo este funcionamiento se logra gracias a `Reflection`, capacidad que nos permite inspeccionar un assembly en tiempo de ejecución. Concluyendo, teniendo la información del nombre de la dll del importador que se necesita, y el nombre del archivo json o xml (dependiendo del importador que se vaya a utilizar), se puede examinar las dll dinámicamente.

La devolución del método `POST` del `import` es un array de 3 posiciones de tupla de enteros. Las tres tuplas del array cumplen con la siguiente regla, el `item1` representa las cantidad de elementos agregados correctamente a la base de datos, y el `item2` representa la cantidad de datos que no se pudo agregar a la base de datos. Las tres posiciones se justifican porque la primera (posición 0) es para las áreas, la segunda (posición 1) es para los temas y la tercera (posición 2) es para los tipos.

## Vista de Procesos

### Diagrama de secuencia: Alta de solicitud

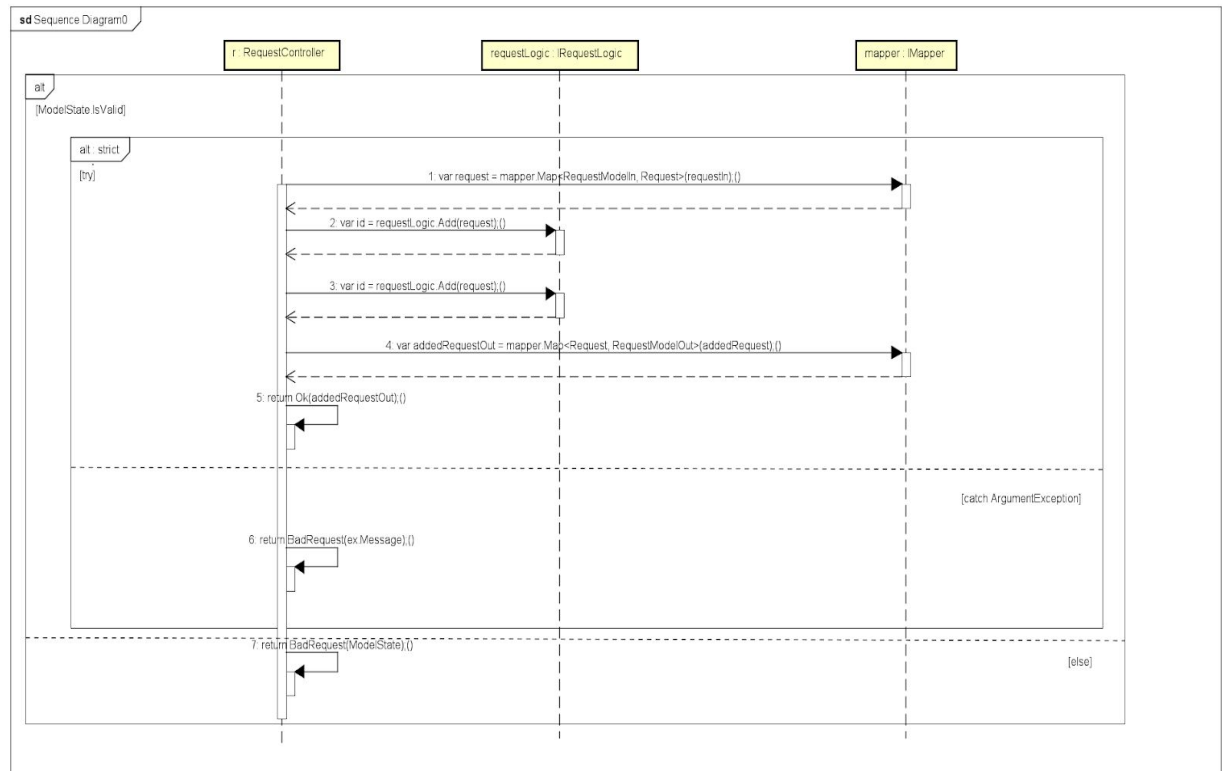


Figura 8: Diagrama de secuencia del método Post del RequestController

## Diagrama de secuencia: Método de importación

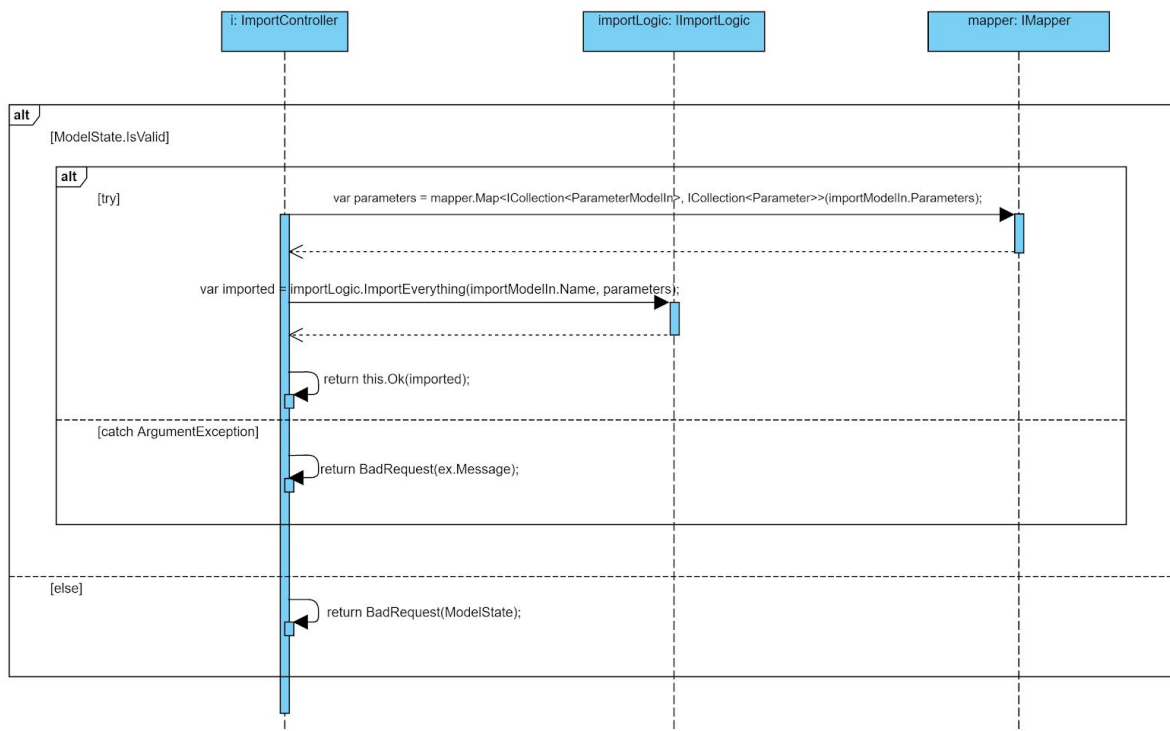


Figura 9: Diagrama de secuencia del método Post del ImportController

## Diagrama de secuencia: Reporte B

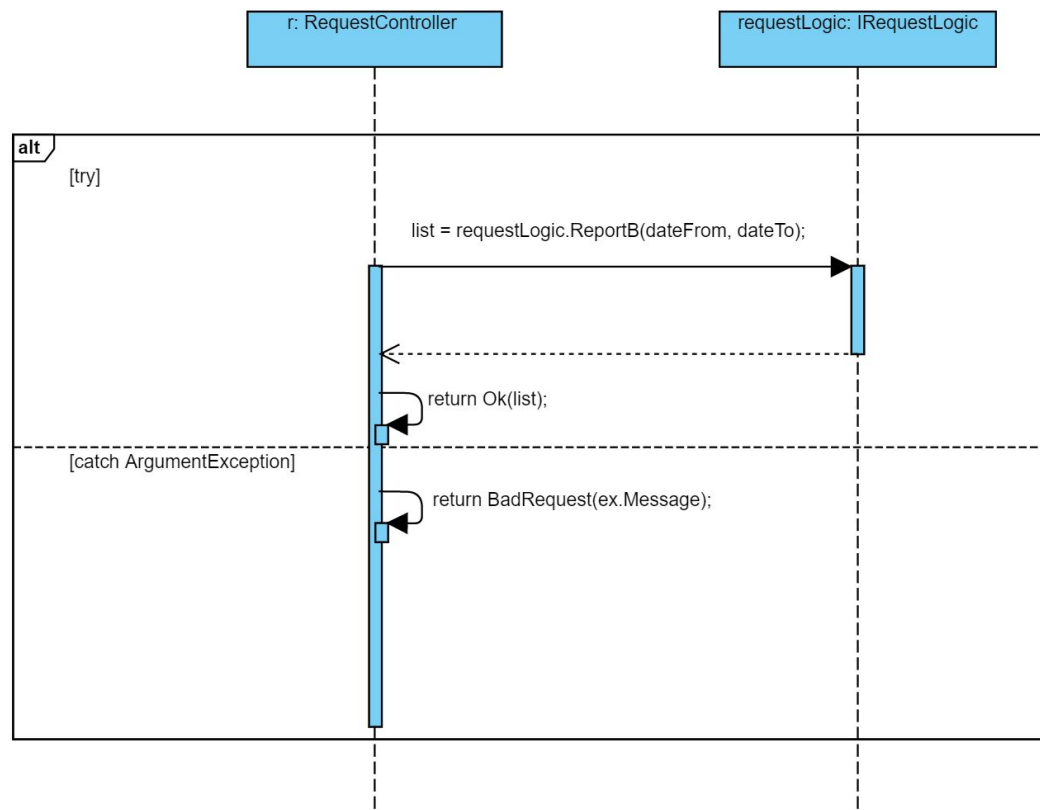


Figura 10: Diagrama de secuencia del método Get del Reporte B

## Vista Física o de Despliegue

### Diagrama de despliegue/deployment

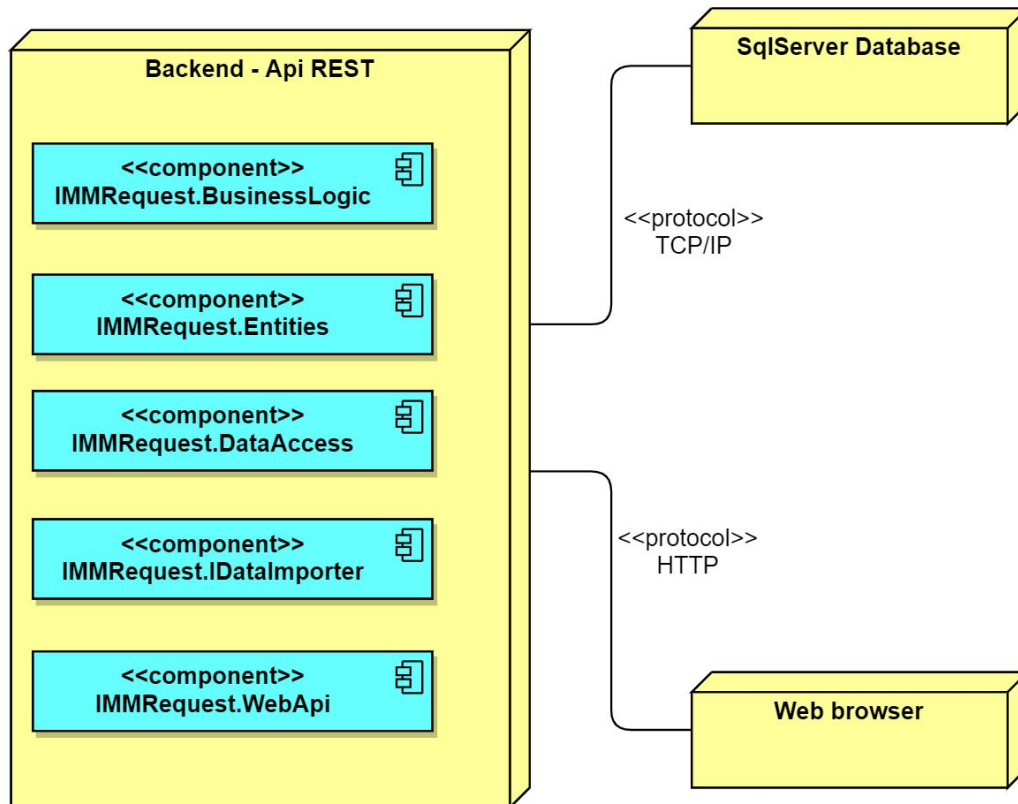


Figura 11: Diagrama de despliegue

En la Figura 11 se observa un diagrama de despliegue o deployment de nuestra aplicación. Este diagrama muestra los objetos de nuestra aplicación una vez que están en tiempo de ejecución, lo que nos permite también ver los protocolos que se utilizan para comunicarse entre los distintos nodos.

Comenzando con nuestra Api REST, es el artefacto que ofrece los componentes que brindan los servicios a la aplicación, y de dónde se consumen los mismos. Es decir, se encarga de exponer la API y estar “escuchando” las solicitudes que lleguen por parte del browser.

El SqlServer Database es el elemento encargado de almacenar la información del sistema, así como también es a donde la API va a buscar, a través de un protocolo TCP/IP, todos aquellos datos que precisa para completar las solicitudes.

Finalmente, el Web Browser es nuestro punto de contacto con el sistema, desde donde nosotros podemos interactuar con la API y la base de datos. A través del navegador, se realizan solicitudes HTTP a la API.

## Informe de métricas

Las métricas se pueden definir como una forma de medir y una escala definidas para realizar mediciones de uno a varios atributos. Sirven para brindar un análisis de nuestro diseño y a partir de las ellas evaluar y desprender conclusiones sobre los resultados que se obtengan. Se pueden dividir en tres categorías:

1. Orientadas al paquete (assembly)
2. Orientadas a la clase
3. Orientadas a métodos

Decidimos utilizar NDepend para analizar las métricas de nuestra solución, ya que es la herramienta que recomendó el curso. NDepend nos brinda un reporte completo y amigable de nuestra solución, en el que se nos da información sobre muchas métricas y otras variables, pero sólo nos centraremos en las que siguen.

### Cohesión

La cohesión relacional es la métrica que indica cuán fuertemente relacionadas están las clases dentro de un paquete. Lo más deseable es que las clases dentro de un paquete estén relacionadas fuertemente pero tampoco excesivamente, es buena si se encuentra entre los valores de 1,5 y 4,0. El cálculo de la cohesión se realiza de la siguiente forma:

$H = (R + 1) / N$  Siendo R el número de relaciones entre clases dentro del paquete (sólo se cuentan las relaciones de asociación, composición y agregación, y las bidireccionales cuentan doble). Y siendo N el número de clases e interfaces del paquete.

La tabla de cohesión relacional obtenida de NDepend fue la siguiente:

Assemblies	Relational Cohesion
IMMRequest.Entities v1.0.0.0	0.56
IMMRequest.BusinessLogic v1.0.0.0	0.75
IMMRequest.IDataImporter v1.0.0.0	1
IMMRequest.DataAccess v1.0.0.0	1.07
IMMRequest.WebApi v1.0.0.0	1.14



Figura 12: Tabla de cohesión de los assemblies de la solución

Como se puede ver en la Figura 12, hay paquetes que tienen mejor cohesión relacional que otros, aunque ninguno de nuestros paquetes llega a estar dentro de ese rango de valores aceptable del que hablamos anteriormente. Esto no quiere decir que tengamos un mal diseño. Por ejemplo, en los paquetes IMMRequest.BusinessLogic y IMMRequest.DataAccess lo que hicimos fue separar las responsabilidades de buena manera, juntando las mismas en respectivas clases y provocando una baja cohesión en el paquete. Por otra parte, en el paquete IMMRequest.Entities tenemos la cohesión más baja de los assemblies analizados, siendo este además el paquete que modela nuestro dominio del problema. Esto se debe a que a la hora de maquetar el diseño no utilizamos las entidades concretas, sino que nos manejamos con los identificadores numéricos de dichas entidades, dejando que sea la lógica de negocio quien se encargue de crear la relación propiamente dicha.

Habiendo hecho este análisis, podemos decir que en una gran parte de nuestros paquetes no estamos cumpliendo con el principio de clausura común (CCP), ya que no agrupamos clases juntas las clases que cambiarían juntas, y esto afecta al mantenimiento del sistema.

## Inestabilidad

Esta métrica se encarga de indicar cuán inestable es un paquete, refiriéndose a cuán fácil es cambiarlo dependiendo de cuántos paquetes lo utilizan. Se calcula de la siguiente manera:

$I = Ce / (Ce + Ca)$  Siendo  $Ce$  las dependencias salientes y  $Ca$  las dependencias entrantes. Sus valores varían entre 0 y 1. Cuanto más cercano sea el valor a 1 entonces el paquete es más inestable, lo que significa que depende de pocos paquetes y es un paquete fácil de cambiar por su poco impacto en los demás. En contraparte, cuanto más cercano sea el valor a 0 más estable será el paquete y dependerá de una gran cantidad de paquetes.

A continuación se muestra el resultado del análisis con NDepend:

Assemblies	Instability
IMMRequest.WebApi v1.0.0.0	1
IMMRequest.IDataImporter v1.0.0.0	0.5
IMMRequest.DataAccess v1.0.0.0	0.84
IMMRequest.BusinessLogic v1.0.0.0	0.73
IMMRequest.Entities v1.0.0.0	0

Figura 13: Tabla de inestabilidad de los paquetes de la solución

Como se puede ver en la tabla, el paquete IMMRequest.WebApi es sumamente inestable, lo que tiene sentido ya que es el paquete de más alto nivel de la solución por lo que no depende de muchos paquetes. Es un paquete al que se lo puede cambiar fácilmente y no impactará en el resto de los paquetes.

Por otra parte, el paquete IMMRequest.Entities que modela nuestro dominio o realidad, es el paquete más estable y sobre el que recaen más dependencias, lo que también tiene sentido. Si cambia este paquete, tendrá un impacto fuerte sobre el resto de los paquetes, incluyendo la capa de negocios y la capa de acceso a datos. Esto no es bueno para el principio de dependencias estables, ya que los paquetes deberían depender de paquetes más abstractos y en este caso las dependencias caen sobre IMMRequest.Entities que es un paquete sumamente estable.

## Abstracción

La métrica de abstracción indica cuán abstracto es un paquete. Se calcula de la siguiente forma:

$A = N_a / N_c$  Siendo  $N_a$  la cantidad de clases abstractas e interfaces en el paquete. Y siendo  $N_c$  la cantidad de clases total, incluyendo clases abstractas e interfaces. El resultado varía entre 0 y 1. Cuanto más cerca de 0, quiere decir que se trata de un paquete más concreto, y cuanto más cerca de 1 se trata de un paquete más abstracto.

La tabla obtenida con NDepend fue la siguiente:

Assemblies	Abstractness
IMMRequest.WebApi v1.0.0.0	0.03
IMMRequest.IDataImporter v1.0.0.0	0.5
IMMRequest.DataAccess v1.0.0.0	0.13
IMMRequest.BusinessLogic v1.0.0.0	0.5
IMMRequest.Entities v1.0.0.0	0

Figura 14: Tabla de abstracción de los paquetes de la solución

Como se puede observar en la tabla, el resultado obtenido es bastante dentro de lo esperado. Tenemos en nuestra solución un paquete sumamente definido o concreto que es IMMRequest.Entities. Por otro lado, tenemos paquetes más abstractos como lo son IMMRequest.BusinessLogic e IMMRequest.IDataImporter, que contienen más abstracciones, sobre todo el primero de estos. En la mayoría de nuestros paquetes manejamos abstracciones pero creemos que esto es un aspecto a mejorar a futuro, ya que en ninguno llegamos a superar la barrera del 0,5 de abstracción.

## Abstracción vs. Inestabilidad

La gráfica de abstracción en función de inestabilidad relaciona ambas métricas de los paquetes. Esta métrica sugiere que los paquetes se encuentren lo más cerca posible de la zona verde (línea punteada o secuencia principal). Luego se encuentra lo que se muestra como una zona naranja que puede llegar a ser problemática, y los paquetes que queden ubicados en dicha zona probablemente deban ser modificados. Finalmente, la zona roja contiene la zona de dolor y la zona de inutilidad, y no es bueno tener paquetes que se encuentren en dichas zonas. Los paquetes que se encuentren en la zona de dolor, quiere decir que será muy difícil modificarlos, por lo que probablemente por el análisis que venimos haciendo veremos a IMMRequest.Entities en esta zona.

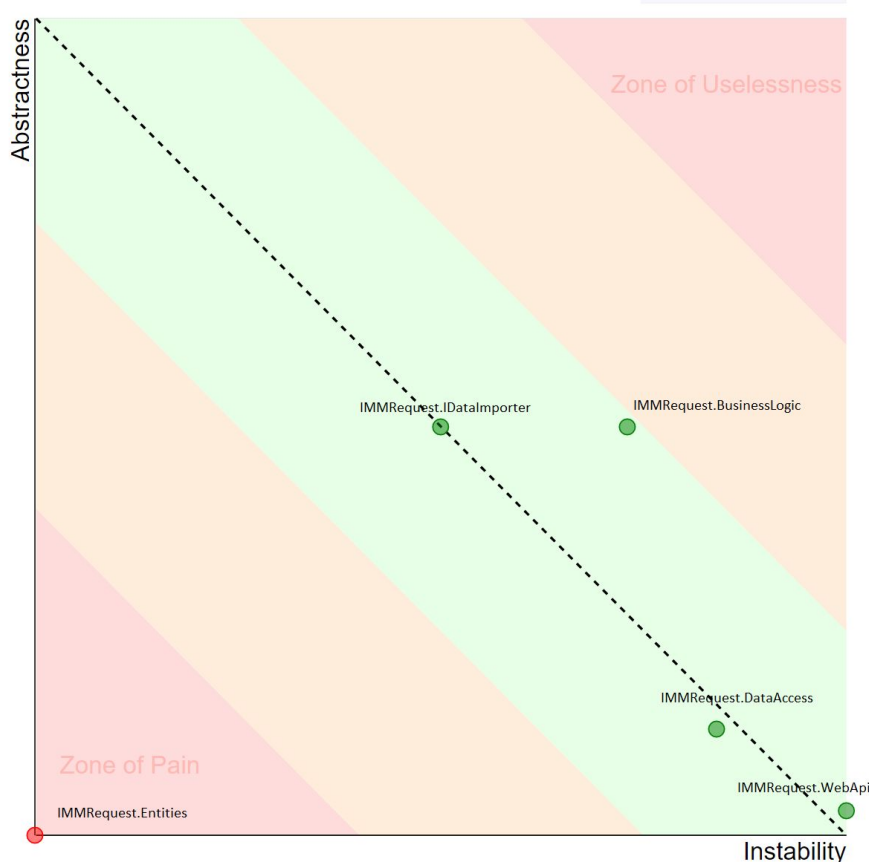


Figura 15: Gráfica de abstracción vs. inestabilidad

Como lo predijimos, el único paquete que tenemos en la zone of pain es el IMMRequest.Entities, que resulta ser un paquete del cual dependen muchos otros paquetes y también un paquete sumamente concreto, por lo que cambiarlo supondría un gran impacto en el resto de la solución, y esto obviamente no es bueno para el diseño de la misma. Sin embargo, hay que tener en cuenta de que estamos hablando del paquete que modela el dominio del problema y carga con reflejar las entidades de un problema de la realidad, por lo que tiene sentido que si estas entidades cambian, tengan un impacto fuerte en el resto del sistema.

Por otra parte, lo positivo es que el resto de nuestros paquetes se encuentran cerca de la secuencia principal y dentro de la green zone, lo que significa que tienen un grado de abstracción e inestabilidad conveniente.

## Anexo I: Modelo de tablas de la estructura de la base de datos

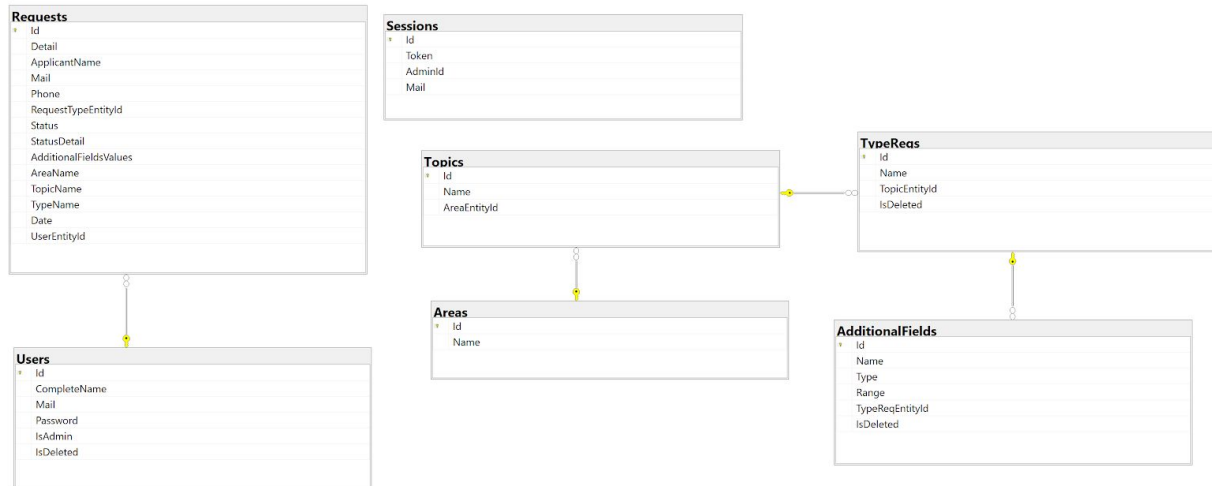


Figura 16: Modelo de tablas de la estructura de la base de datos

En este diagrama de la figura podemos visualizar todas las tablas del sistema incluyendo las relaciones entre ellas.

## Anexo II: Análisis de cobertura de pruebas

Esta segunda entrega fue realizada enteramente mediante TDD, siguiendo la estrategia que ya se explicó anteriormente. Esta técnica se respetó durante todo el desarrollo del backend de la solución.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
▲ Nahuel_DESKTOP-B5NVOVE 2020-06-25 16_4...	1053	11,25 %	8304	88,75 %
▶ immrequest.businesslogic.dll	612	25,35 %	1802	74,65 %
▶ immrequest.businesslogic.test.dll	119	4,27 %	2666	95,73 %
▶ immrequest.dataaccess.dll	22	5,08 %	411	94,92 %
▶ immrequest.dataaccess.test.dll	0	0,00 %	406	100,00 %
▶ immrequest.entities.dll	12	6,74 %	166	93,26 %
▶ immrequest.idataimporter.dll	1	25,00 %	3	75,00 %
▶ immrequest.jsondataimporter.dll	6	6,67 %	84	93,33 %
▶ immrequest.jsondataimporter.test.dll	0	0,00 %	93	100,00 %
▶ immrequest.webapi.dll	275	27,86 %	712	72,14 %
▶ immrequest.webapi.test.dll	0	0,00 %	1720	100,00 %
▶ immrequest.xmldataimporter.dll	6	4,72 %	121	95,28 %
▶ immrequest.xmldataimporter.test.dll	0	0,00 %	120	100,00 %

Figura 17: Cobertura de pruebas unitarias de la solución

Observando de forma general el porcentaje de cobertura de pruebas unitarias, podemos observar que decayó un poco comparando con la primera entrega, ya que ahora tenemos un 88,75% de cobertura, aunque debemos tener en cuenta que en este porcentaje hay muchos paquetes que no deben ser tomados en cuenta.

### IMMRequest.BusinessLogic

▶ immrequest.businesslogic.dll	612	25,35 %	1802	74,65 %
▶ IMMRequest.BusinessLogic	612	25,35 %	1802	74,65 %
▶ AdditionalFieldLogic	67	19,09 %	284	80,91 %
▶ AdditionalFieldLogic.<>c	0	0,00 %	2	100,00 %
▶ AreaLogic	0	0,00 %	84	100,00 %
▶ ImportLogic	74	62,18 %	45	37,82 %
▶ RequestLogic	420	43,75 %	540	56,25 %
▶ RequestLogic.<>c	0	0,00 %	2	100,00 %
▶ RequestLogic.<>c_DisplayClass31_0	4	100,00 %	0	0,00 %
▶ RequestLogic.<>c_DisplayClass4_0	0	0,00 %	3	100,00 %
▶ SessionLogic	9	6,04 %	140	93,96 %
▶ TopicLogic	9	6,87 %	122	93,13 %
▶ TypeReqLogic	23	7,10 %	301	92,90 %
▶ TypeReqLogic.<>c	0	0,00 %	4	100,00 %
▶ UserLogic	6	2,15 %	273	97,85 %
▶ UserLogic.<>c	0	0,00 %	2	100,00 %

Figura 18: Cobertura de pruebas de BusinessLogic

Haciendo foco en el paquete que encapsula la lógica del negocio de la solución, tenemos una cobertura promedio de 74,65%. Esto puede deberse a que la lógica de

importación maneja variables IConfiguration, y estas hacen muy complejas las pruebas, por lo que las pruebas de la clase ImportLogicTest no logramos hacer que pasaran.

## IMMRequest.DataAccess

immrequest.dataaccess.dll	22	5,08 %	411	94,92 %
IMMRequest.DataAccess	22	5,08 %	411	94,92 %
Context	7	2,06 %	333	97,94 %
Repository<T>	7	17,07 %	34	82,93 %
UnitOfWork	8	15,38 %	44	84,62 %

Figura 19: Cobertura de pruebas de DataAccess

En cuanto a la cobertura de pruebas del paquete que encapsula el acceso a datos, obtenemos un muy buen porcentaje de 94,92% de cobertura.

## IMMRequest.Entities

immrequest.entities.dll	12	6,74 %	166	93,26 %
IMMRequest.Entities	12	6,74 %	166	93,26 %
AdditionalFieldEntity	2	7,41 %	25	92,59 %
AreaEntity	1	4,76 %	20	95,24 %
RequestEntity	2	4,88 %	39	95,12 %
SessionEntity	3	25,00 %	9	75,00 %
TopicEntity	2	8,70 %	21	91,30 %
TypeReqEntity	1	4,00 %	24	96,00 %
UserEntity	1	3,45 %	28	96,55 %

Figura 20: Cobertura de pruebas de Entities

En el paquete donde se encuentran las entidades que modelan el problema de la realidad, nos encontramos nuevamente con un alto porcentaje de cobertura de 93,26%. Siendo además la única entidad por debajo del 90% de cobertura SessionEntity, que se encarga de guardar la información de las sesiones de usuarios para luego manejar la autenticación.

A continuación vemos en detalle dicha entidad:

SessionEntity	3	25,00 %	9	75,00 %
SessionEntity()	0	0,00 %	4	100,00 %
get_AdminId()	1	100,00 %	0	0,00 %
get_Id()	1	100,00 %	0	0,00 %
get_Mail()	1	100,00 %	0	0,00 %
get_Token()	0	0,00 %	1	100,00 %
set_AdminId(int)	0	0,00 %	1	100,00 %
set_Id(System.Guid)	0	0,00 %	1	100,00 %
set_Mail(string)	0	0,00 %	1	100,00 %
set_Token(System.Guid)	0	0,00 %	1	100,00 %

Figura 21: SessionEntity

Como podemos ver, la causa de la baja cobertura son los get de algunas properties de esta clase, ya que al utilizar properties autogeneradas {get; set;} se necesita tener un get y este no es utilizado en todo el código para esas tres propiedades.

## IMMRequest.WebApi

immrequest.webapi.dll	275	27,86 %	712	72,14 %
IMMRequest.WebApi	72	100,00 %	0	0,00 %
Program	10	100,00 %	0	0,00 %
Program.<>c	2	100,00 %	0	0,00 %
Startup	42	100,00 %	0	0,00 %
Startup.<>c	18	100,00 %	0	0,00 %
IMMRequest.WebApi.Controllers	131	24,95 %	394	75,05 %
IMMRequest.WebApi.Filters	40	100,00 %	0	0,00 %
AuthFilter	40	100,00 %	0	0,00 %
IMMRequest.WebApi.Mapper	0	0,00 %	212	100,00 %
IMMRequest.WebApi.Models	32	23,19 %	106	76,81 %

Figura 22: Cobertura de pruebas de WebApi

Este paquete tiene una cobertura de pruebas satisfactoria de 72,14%, aunque no es tan buena como la del resto de los paquetes. Esto se debe a que en este paquete se encuentran las clases Program y Startup que no son probadas, y también se encuentra el auth filter que tampoco está probado. Por otro lado, también en los Models se encuentran muchas clases que tienen properties autogeneradas con métodos get que no se utilizan, como sucede con la clase SessionEntity explicado antes.



## **Anexo III: Documento actualizado de la API**

### **Criterios utilizados para cumplimiento de REST**

REST (Representational State Transfer) define un estado arquitectónico para construir aplicaciones web o distribuidas en general. Es decir, un conjunto de restricciones arquitectónicas. Funciona en base a un protocolo cliente servidor, de tipo stateless y en la mayoría de los casos, este protocolo es HTTP. Las arquitecturas de este tipo imponen una capa de abstracción, al cliente no le importa el lenguaje o la tecnología que hay del lado del servidor.

El primero de los criterios a seguir, dice que una API REST tiene una arquitectura cliente servidor, y nosotros cumplimos esto en nuestra solución.

El segundo de los criterios que cumplimos de REST, es el de stateless. Esto significa que cada request que un cliente haga será independiente, y deberá contener el estado necesario para que el servidor resuelva dicha solicitud. Esto hace que el comportamiento de la API sea más predecible y entendible y favorece la escalabilidad. Cada request debe ser autocontenida, y no debemos tener estado en el lado del servidor.

El tercero de los criterios de REST es el de cacheable. Esta restricción implica que el servidor puede llegar a indicar a su cliente, en su respuesta, si esta es cacheable o no, para ver si servidores intermediarios como un proxy pueden cachear la respuesta. Este criterio no podemos asegurar que haya sido cumplido con total seguridad, dado que no realizamos pruebas específicas para explorar su funcionamiento.

Finalmente, el cuarto criterio es el de interfaz uniforme, y habla de nuestra API debe ofrecer una interfaz que desacople la forma de consumir la API de su implementación concreta. Para definir esta interfaz, utilizamos los estándares que REST propone:

- Identificación de recursos (URI): REST es orientado a recursos en lugar de a operaciones, los cuales serán identificados por URIs.
- Manipulamos estos recursos a partir de su representación, es decir, utilizamos un estándar para describir la comunicación e indicar qué tipo de representación se desea obtener. Esto quiere decir que en nuestro caso, dado que utilizamos un protocolo HTTP, usamos verbos HTTP (GET, POST, PUT, DELETE, etc) combinados con la URI, que debe seguir también ciertos estándares que explicamos más adelante.

Para el manejo de las URLs, que son el principal elemento de la usabilidad de la API, se definen los siguientes estándares:

- Utilizamos un máximo de 3 URIs por resource
- Utilizamos sustantivos en lugar de verbos

## Descripción del mecanismo de autenticación de requests

Conceptualmente, una vez que al servidor le llega una request y se elige una acción, se ejecutan una serie de pasos definidos aparte de las acciones que definimos en los controllers de nuestra solución. El conjunto completo de estos pasos se conoce como pipeline de la request, y los filtros son parte de esta.

Para la autenticación de nuestro sistema, y por ende, de las requests, utilizamos un Authorization Filter. Como ya se explicó anteriormente, esta fue una de nuestras mejoras con respecto a la primera entrega.

Para implementar este filtro, primero tuvimos que modificar nuestra solución. Agregamos una clase SessionEntity, que representa a la entidad de una sesión de un usuario, dentro de la cual hay información sobre la sesión, como el token y el usuario que inició la sesión. Luego, en el contexto de la base de datos creamos un DbSet de SessionEntity, con el fin de que se cree una tabla que almacene la información sobre las sesiones en la base de datos.

Pasando a la implementación de la lógica, creamos la clase SessionLogic que implementa la interfaz ISessionLogic, la cual es muy sencilla, implementa métodos de inicio y cierre de sesión, y un método booleano de chequeo de validez de token. Esta lógica es utilizada desde un SessionController que creamos en nuestra API, a partir de la cual el usuario puede realizar las funciones de inicio y cierre de sesión.

Finalmente, nuestro Authorization Filter es llamado AuthFilter e implementa la interfaz IAuthorizationFilter, y luego se utilizó a lo largo de todos los métodos que nuestro sistema requería que tuvieran una autenticación o filtro para ser ejecutados. Más concretamente, para las funciones de administrador utilizamos el filtro, mientras que las funcionalidades del usuario ciudadano, utilizamos un [AllowAnonymous] que permite que cualquier usuario sin necesidad de estar autenticado pueda realizar esa request.

## Descripción general de los códigos de error

A continuación se especifica una lista descriptiva de los códigos de respuesta que existen en HTTP con una breve descripción.

1XX	Respuestas informativas
2XX	Peticiones correctas
3XX	Redirecciones
4XX	Error del cliente
5XX	Error del servidor

Tabla 1: Códigos de respuesta HTTP

Como se puede observar, los códigos de respuesta 1xx y 3xx correspondientes a respuestas informativas y redirecciones respectivamente, no se acoplan demasiado a nuestra API, por lo que no se encontrarán estos códigos de respuesta a la hora de su testeo o utilización.

Ahora veremos cuáles de estos códigos de respuesta fueron utilizados en nuestra solución más concretamente, dependiendo de la operación realizada (verbo HTTP) y de si la información recibida es válida o no.

Código de respuesta	Utilización
200 Ok	Utilizado para todas aquellas request con verbo HTTP de tipo GET y DELETE que reciben correctamente la información en la URI correspondiente
201: Created	Utilizado para todas aquellas request con verbo HTTP de tipo PUT y POST que reciben correctamente la información en la URI y el body correspondientes
400: Bad Request	Utilizado para todas las request de todos los verbos HTTP cuando hay un error del cliente
401: Unauthorized	Este código se utiliza cuando un usuario intenta realizar una request para la cual no tiene el rol necesario
404: Not Found	Utilizado cuando no se encuentra la URI escrita por el usuario
405: Method Not Allowed	Utilizado cuando una request fue hecha a una URI utilizando un verbo HTTP que esa URI no acepta
500: Internal Server Error	Utilizado cuando hay un error de parte del servidor

Tabla 2: Códigos utilizados en nuestra solución

A continuación se detalla la especificación actualizada de la API, siguiendo los consejos de la corrección obtenida por la primera entrega.

## Especificación de Resource Session de la API

URL base: <http://{{ip}}/{{port}}/api/session>

Description	Responses	Endpoint + Body + Headers
<u>Description:</u> Ingresa a una sesión de un usuario administrador en el sistema. Corresponde al requerimiento: “Un administrador debe poder iniciar sesión usando su email y contraseña”.	<u>Responses:</u>  <b>201</b> - { token, mail, userId }  <b>401</b> - “You have to be an admin to login”  <b>401</b> - “Wrong mail/password”	<u>Verbo:</u> POST  <u>URI:</u> /login  <u>Body:</u> { mail: string, password: string }
<u>Description:</u> Cierra la sesión de un usuario, eliminandola de la base de datos. Se debe pasar por la URI el id del usuario del que se quiere cerrar la sesión.	<u>Responses:</u>  <b>200</b> - {}  <b>400</b> - “You have to be logged in to logout”	<u>Verbo:</u> DELETE  <u>URI:</u> /logout  <u>Headers:</u> Authorization {{token}}

Tabla 3: Datos sobre el resource Session

## Especificación de Resource User de la API

URL base: <http://{{ip}}/{{port}}/api/user>

Description	Responses	Endpoint + Body + Headers
<p><u>Description:</u> Devuelve una lista de todos los usuarios que se encuentran almacenados en base de datos.</p>	<p><u>Responses:</u></p> <p><b>200 -</b></p> <pre>[   userModelOut =   {     id,     completeName,     mail,     password,     isAdmin,     role   } ]</pre>	<p><u>Verbo:</u> GET</p>
<p><u>Description:</u> Permite registrar un usuario en el sistema. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento del sistema: “Un administrador puede realizar mantenimiento de administradores del sistema”.</p>	<p><u>Responses:</u></p> <p><b>201 -</b></p> <pre>{   id,   completeName,   mail,   mail,   password,   isAdmin,   role }</pre> <p><b>401 -</b> “You have to be an admin to do this”</p> <p><b>400 -</b> Bad request</p>	<p><u>Verbo:</u> POST</p> <p><u>Body:</u></p> <pre>{   completeName: string,   mail: string,   password: string,   isAdmin: bool }</pre> <p><u>Headers:</u> Authorization {{token}}</p>
<p><u>Description:</u> Permite hacer</p>	<p><u>Responses:</u></p>	<p><u>Verbo:</u> PUT</p>

<p>modificaciones a un usuario existente del sistema. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento del sistema: “Un administrador puede realizar mantenimiento de administradores del sistema”.</p>	<p><b>201 -</b>  <pre>{   id,   completeName,   mail,   mail,   password,   isAdmin,   role }</pre></p> <p><b>401 -</b>          “You have to be an admin to do this”</p> <p><b>400 -</b>          “The user with id {{userId}} doesn’t exists”</p>	<p><u>Body:</u>  <pre>{   completeName: string,   mail: string,   password: string,   isAdmin: bool }</pre></p> <p><u>Headers:</u>          Authorization {{token}}</p>
<p><u>Description:</u>          Devuelve al usuario con el id pasado en la URI si existe.</p>	<p><u>Responses:</u></p> <p><b>200 -</b>          userModelOut =  <pre>{   id,   completeName,   mail,   password,   isAdmin,   role }</pre></p> <p><b>400 -</b>          “The user with id {{userId}} doesn’t exists”</p>	<p><u>Verbo:</u>          GET</p> <p><u>URI:</u>          /{userId}</p> <p>El path param refiere al id del usuario que se quiere modificar</p> <p><u>Headers:</u>          Authorization {{token}}</p>
<p><u>Description:</u>          Elimina el usuario con el id pasado en la URI de la base de datos. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento del sistema: “Un administrador puede</p>	<p><u>Responses:</u></p> <p><b>200 -</b>  <pre>{}</pre></p> <p><b>400 -</b>          “The user with id {{userId}} doesn’t exists”</p>	<p><u>Verbo:</u>          DELETE</p> <p><u>URI:</u>          /{userId}</p> <p>El path param refiere al id del usuario que se quiere eliminar</p>

realizar mantenimiento de administradores del sistema”.		<u>Headers:</u> Authorization {{token}}
<p>Get Requests</p> <p><u>Description:</u> Devuelve la lista de requests que haya realizado el usuario con el id pasado por la URI.</p>	<p><u>Responses:</u></p> <p><b>200 -</b> [ requestModelOut =   {     id,     detail,     applicantName,     mail,     status,     statusDetail,     phone,     requestTypeId,     [additionalFieldVls       {         date,         string,         int       }]     areaName,     topicName,     typeName   } ]  <b>400 -</b> “The user with id {{userId}} doesn’t have any requests”</p>	<p><u>Verbo:</u> GET</p> <p><u>URI:</u> /{userId}</p> <p>El path param refiere al id del usuario del que se quiere obtener las solicitudes</p>

Tabla 4: Datos sobre resource User

## Especificación de Resource Request de la API

URL base: <http://{{ip}}/{{port}}/api/request>

Description	Responses	Endpoint + Body + Headers
<p><u>Description:</u> Devuelve una lista de todas las requests que se encuentran almacenadas en base de datos. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento: “Listar las solicitudes existentes”.</p>	<p><u>Responses:</u></p> <p><b>200 -</b> [ requestModelOut = {     id,     detail,     applicantName,     mail,     status,     statusDetail,     phone,     requestTypeId,     [additionalFieldVls     {         date,         string,         int     }],     areaName,     topicName,     typeName     } }] ]</p> <p><b>401 -</b> “You have to be an admin to do this”</p>	<p><u>Verbo:</u> GET</p> <p><u>Headers:</u> Authorization {{token}}</p>
<p><u>Description:</u> Permite ingresar una request al sistema. Corresponde al requerimiento: “Realizar el ingreso de una nueva solicitud (sin necesidad de estar registrado ni autenticado)”.</p>	<p><u>Responses:</u></p> <p><b>201 -</b> [ requestModelOut = {     id,     detail,     applicantName,     mail,     status,     statusDetail,     phone,</p>	<p><u>Verbo:</u> POST</p> <p><u>Body:</u> {     detail: string,     applicantName: string,     mail: string,     phone: string,     requestTypeId: int,     additionalFieldValues:[] }</p>



	<pre> requestTypeId, [additionalFieldVls {   date,   string,   int }], areaName, topicName, typeName } ] </pre> <p><b>400 -</b> Bad request</p>	
<p><u>Description:</u> Devuelve la request a la que corresponde el id pasado en la URI si existe. Esta funcionalidad es sólo permitida para los administradores.</p>	<p><u>Responses:</u></p> <p><b>200 -</b> [ requestModelOut = {   id,   detail,   applicantName,   mail,   status,   statusDetail,   phone,   requestTypeId,   [additionalFieldVls   {     date,     string,     int   }],   areaName,   topicName,   typeName } ]</p> <p><b>400 -</b> “The request with id {{requestId}} doesn’t exists”</p>	<p><u>Verbo:</u> GET</p> <p><u>URI:</u> /{requestId}</p> <p>El path param refiere al id de la solicitud que se quiere obtener</p> <p><u>Body:</u> {   detail: string,   applicantName: string,   mail: string,   phone: string,   requestTypeId: int,   additionalFieldValues:[] }</p> <p><u>Headers:</u> Authorization {{token}}</p>

	<b>401 -</b> “You have to be an admin to do this”	
<u>Description:</u> Permite hacer modificaciones a una solicitud existente del sistema. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento del sistema: “Un administrador debe poder cambiar el estado de una solicitud”.	<u>Responses:</u>  <b>201 -</b> <pre>[   requestModelOut =   {     id,     detail,     applicantName,     mail,     status,     statusDetail,     phone,     requestTypeId,     [additionalFieldVls     {       date,       string,       int     }],     areaName,     topicName,     typeName   } ]</pre> <b>401 -</b> “You have to be an admin to do this”	<u>Verbo:</u> PUT  <u>URI:</u> /{requestId}  El path param refiere al id de la solicitud que se quiere modificar  <u>Body:</u> <pre>{   detail: string,   applicantName: string,   mail: string,   phone: string,   requestTypeId: int,   additionalFieldValues:[] }</pre> <u>Headers:</u> Authorization {{token}}
<u>Description:</u> Permite obtener el reporte A.	<u>Responses:</u>  <b>201 -</b> <pre>[   requestModelOut =   {</pre>	<u>Verbo:</u> GET  <u>URI:</u> /{dateFrom}/{dateTo}/{mail}

	id, detail, applicantName, mail, status, statusDetail, phone, requestTypeId, [additionalFieldVls { date, string, int }], areaName, topicName, typeName } ]  <b>401 -</b> “You have to be an admin to do this”	El path param refiere a la fecha inicio desde la que se quiere obtener el reporte (dateFrom), la fecha fin (dateTo), y el mail del usuario (mail) de quien se desea obtener las solicitudes.  <u>Headers:</u> Authorization {{token}}
<u>Description:</u> Permite obtener el reporte B.	<u>Responses:</u>  <b>201 -</b> [ requestModelOut = { id, detail, applicantName, mail, status, statusDetail, phone, requestTypeId, [additionalFieldVls { date, string, int }], areaName, topicName, typeName } ]  <b>401 -</b> “You have to be an admin to do this”	<u>Verbo:</u> GET  <u>URI:</u> /{dateFrom}/{dateTo}  El path param refiere a la fecha inicio desde la que se quiere obtener el reporte (dateFrom) y la fecha fin (dateTo)  <u>Headers:</u> Authorization {{token}}

	<pre>       typeName     }   ] </pre> <p><b>401 -</b> “You have to be an admin to do this”</p>	
<p><u>Description:</u> Devuelve el estado de la request que corresponde al id que se pasa por la URI si esta existe. Corresponde al requerimiento: “Consultar el estado de una solicitud usando el número de solicitud (sin necesidad de estar registrado ni autenticado)”.</p>	<p><u>Responses:</u></p> <p><b>200 -</b> “La request de id {{requestId}} tiene estado: {{status}}”</p> <p><b>400 -</b> “The request with id {{requestId}} doesn’t exists”</p>	<p><u>Verbo:</u> GET</p> <p><u>URI:</u> /{requestId}</p> <p>El path param refiere al id de la solicitud que se quiere obtener</p>

Tabla 5: Datos del resource Request

## Especificación de Resource TypeReq de la API

URL base: <http://{{ip}}/{{port}}/api/typereq>

Description	Responses	Endpoint + Body + Headers
<p><u>Description:</u> Devuelve una lista de todos los tipos de solicitudes que se encuentran almacenados en base de datos.</p>	<p><u>Responses:</u></p> <p><b>200 -</b> [   typeReqModelOut =   {     id,     name,     topicId   } ]</p>	<p><u>Verbo:</u> GET</p>
<p><u>Description:</u> Permite registrar un tipo</p>	<p><u>Responses:</u></p>	<p><u>Verbo:</u> POST</p>

<p>de solicitud en el sistema. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento del sistema: “Un administrador puede dar de alta a un nuevo tipo”.</p>	<p><b>201 -</b> typeReqModelOut = {   id,   name,   topicId }</p> <p><b>401 -</b> “You have to be an admin to do this”</p> <p><b>400 -</b> “The topic with id {{ topicId }} doesn’t exists”</p>	<p><u>Body:</u> {   name: string,   topicId: ind }</p> <p><u>Headers:</u> Authorization {{token}}</p>
<p><u>Description:</u> Permite hacer modificaciones a un tipo de solicitud existente del sistema. Esta funcionalidad es sólo permitida para los administradores.</p>	<p><u>Responses:</u></p> <p><b>201 -</b> typeReqModelOut = {   id,   name,   topicId }</p> <p><b>401 -</b> “You have to be an admin to do this”</p> <p><b>400 -</b> “The topic with id {{ topicId }} doesn’t exists”</p>	<p><u>Verbo:</u> PUT</p> <p><u>URI:</u> /{typeReqId}}</p> <p>El path param refiere al id del tipo de solicitud que se quiere modificar</p> <p><u>Body:</u> {   name: string,   topicId: ind }</p> <p><u>Headers:</u> Authorization {{token}}</p>
<p><u>Description:</u> Devuelve al tipo de solicitud con el id pasado en la URI si existe.</p>	<p><u>Responses:</u></p> <p><b>200 -</b> typeReqModelOut = {   id,   name,   topicId }</p> <p><b>400 -</b></p>	<p><u>Verbo:</u> PUT</p> <p><u>URI:</u> /{typeReqId}}</p> <p>El path param refiere al id del tipo de solicitud que se quiere modificar</p>

	“The type request with id {{typeReqId}} doesn’t exists”	
<u>Description:</u> Elimina el tipo de solicitud con el id pasado en la URI de la base de datos. Esta funcionalidad es sólo permitida para los administradores. Corresponde al requerimiento del sistema: “Un administrador puede realizar borrar un tipo existente”.	<u>Responses:</u>  <b>200 -</b> {}  <b>400 -</b> “The type request with id {{typeReqId}} doesn’t exists”	<u>Verbo:</u> DELETE  <u>URI:</u> /{typeReqId}  El path param refiere al id del tipo de solicitud que se quiere modificar  <u>Headers:</u> Authorization {{token}}

Tabla 6: Datos del resource TypeReq

## Especificación de Resource AdditionalField de la API

URL base: <http://{{ip}}/{{port}}/api/additionalfield>

Description	Responses	Endpoint + Body + Headers
<u>Description:</u> Devuelve una lista de todos los campos adicionales almacenados en la base de datos. Esta funcionalidad es sólo permitida para los administradores	<u>Responses:</u>  <b>200 -</b> additionalFieldModelOut = { id, name, type, range: [], typeReqId }  <b>400 -</b> “The type request with id {{typeReqId}} doesn’t exists”	<u>Verbo:</u> GET  <u>Headers:</u> Authorization {{token}}
<u>Description:</u>	<u>Responses:</u>	<u>Verbo:</u>

<p>Permite registrar un campo adicional de un tipo de solicitud en el sistema. Esta funcionalidad es sólo permitida para los administradores.</p>	<p><b>201 -</b> additionalFieldModelOut = {     id,     name,     type,     range: [],     typeReqId }</p> <p><b>401 -</b> “You have to be an admin to do this”</p> <p><b>400 -</b> “The type request with id {{ typeReqId }} doesn’t exists”</p>	<p>POST</p> <p><u>Body:</u> {     name: string,     type: string,     range: [],     typeReqId: int }</p> <p><u>Headers:</u> Authorization {{token}}</p>
<p><u>Description:</u> Devuelve al campo adicional del tipo de solicitud con el id pasado en la URI si existe. Esta funcionalidad es sólo permitida a los administradores</p>	<p><u>Responses:</u></p> <p><b>200 -</b> additionalFieldModelOut = {     id,     name,     type,     range: [],     typeReqId }</p> <p><b>400 -</b> “The additional field with id {{additionalFieldId}} doesn’t exists”</p>	<p><u>Verbo:</u> GET</p> <p><u>URI:</u> /{additionalFieldId}</p> <p>El path param refiere al id del campo adicional que se quiere obtener</p> <p><u>Headers:</u> Authorization {{token}}</p>

Tabla 7: Datos del resource AdditionalField

## Especificación de Resource Import de la API

URL base: <http://{{ip}}/{{port}}/api/import>

Description	Responses	Endpoint + Body + Headers
<u>Description:</u> Devuelve una lista de todos los importadores disponibles en la carpeta /Assemblies	<u>Responses:</u>  <b>200 -</b> importers = { "JsonDataImporter.dll", "XmlDataImporter.dll" }  <b>400 -</b> BadRequest("There are no importers available")	<u>Verbo:</u> GET
<u>Description:</u> Permite realizar la importación con uno de los importers disponibles en la carpeta /Assemblies	<u>Responses:</u>  <b>201 -</b> Tuple<int,int>[3]= { { item1: int, item2: int, }, { item1: int, item2: int, }, { item1: int, item2: int, } }  <b>400 -</b> BadRequest	<u>Verbo:</u> POST  <u>Body:</u> { name: string, parameters: [{ name: string, value: string }] }

Tabla 9: Datos del resource Import




## Anexo IV: Capturas de pantalla del Frontend


### Reportes A y B

Id	Detail	ApplicantName	Mail	Phone	RequestTypeE...	Status	StatusDetail	AdditionalFiel...	AreaName	TopicName	TypeName	Date	UserEntityId
1	El contenedor d...	Juan Perez	jperez@hotmail...	097463723	9	ENREVISION	asdasdas		Area 1	Topic 1	Contenedor roto	2020-06-24 00:1...	1
2	otrasdas	Jasd	jasd@l.com	097463723	9	CREADA	La solicitud fue ...		Area 1	Topic 1	Contenedor roto	2020-06-24 00:1...	NULL
3	otranffcvsvdas...	Jasd	jasd@l.com	097463723	9	CREADA	La solicitud fue ...		Area 1	Topic 1	Contenedor roto	2020-06-24 00:1...	NULL
4	nueva request	Pedro	pedrito@hotmail...	NULL	9	CREADA	La solicitud fue ...	Empresa de taxi...	Area 1	Topic 1	Contenedor roto	2020-06-24 16:3...	NULL
5	nueva requezc...	Pedro	pedrito@hotmail...	NULL	9	CREADA	La solicitud fue ...	Empresa de taxi...	Area 1	Topic 1	Contenedor roto	2020-06-24 16:4...	NULL
6	nueva requezc...	Pedro	pedrito@hotmail...	NULL	9	CREADA	La solicitud fue ...	Empresa de taxi...	Area 1	Topic 1	Contenedor roto	2020-06-24 17:4...	NULL
7	nueva requeasd...	Pedro	pedrito@hotmail...	NULL	9	CREADA	La solicitud fue ...	Empresa de taxi...	Area 1	Topic 1	Contenedor roto	2020-06-24 21:2...	NULL
8	nueva requeasd...	Pedro	pedrito@hotmail...	NULL	9	CREADA	La solicitud fue ...	Empresa de taxi...	Area 1	Topic 1	Contenedor roto	2020-06-24 21:2...	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL


Primera captura: Datos de la base


#### Reporte A

 Hora:

 Hora:

#### Reporte B

 Hora:

 Hora:

Segunda captura: Pantalla del frontend donde se ingresan los datos para generar los reportes

### Reporte A

- En revisión (3) = [2, 4, 5]
- Aceptada (2) = [3, 7]
- Denegada (1) = [6]
- Finalizada (2) = [1, 8]

### Reporte B

- Contenedor roto (8)

Tercera captura: Resultado de reportes

### Ingreso de nueva solicitud

Mail
Detalle
Nombre del aplicante
Telefono
Detalle del estado
Id del tipo

Traer campos adicionales

Ok

Captura: Ingreso de nueva solicitud

# Listado de solicitudes

## Id de la solicitud: 1

Detalle: El contenedor de la esquina Av Gral Rivera y Eduardo Mac Eachen fue incendiado en la madrugada del sábado 21 de marzo

Nombre del aplicante: Juan Perez

Estado: FINALIZADA

Detalle del estado: asdasd

Area: Area 1

Tema: Topic 1

Tipo: Contenedor roto

Fecha:

Cambiar estado

## Id de la solicitud: 2

Detalle: otrarasdas

Nombre del aplicante: Jasd

Estado: ENREVISION

Detalle del estado: asdas

Area: Area 1

Tema: Topic 1

Tipo: Contenedor roto

Fecha:

Cambiar estado

## Id de la solicitud: 3

Detalle: otrarxffcvsdvasdas

Nombre del aplicante: Jasd

Estado: ACEPTADA

Primera captura: Listado de solicitudes sin campos adicionales

## Id de la solicitud: 7

Detalle: nueva requeasdazxczxt

Nombre del aplicante: Pedro

Estado: ACEPTADA

Detalle del estado: asdadfa

Area: Area 1

Tema: Topic 1

Tipo: Contenedor roto

Fecha:

Campos adicionales:

- Empresa de taxi: System.Collections.Generic.List`1[System.String]
- Empresa de taxdfsdfi: System.Collections.Generic.List`1[System.String]
- Empresa de taxizxczx: System.Collections.Generic.List`1[System.String]
- Empresa de tsdfsdbsdaxi: Radio Taxi\*Taxi aeropuerto
- erroneo: 3
- dsdfwds: true
- errzdcsdoneo: 17/03/2021

Cambiar estado

Segunda captura: Aquí hay una nueva captura con una solicitud que tiene campos adicionales

## Modificación de usuario

The screenshot shows a user management interface with a list of users on the left and a modal for editing a user on the right.

**User List:**

- Prueba**  
pruezxzba@hotmail.com  
El usuario es administrador  
[Editar] [Eliminar]
- Prueba**  
pruezasxzba@hotmail.com  
El usuario es administrador  
[Editar] [Eliminar]
- federicoasda**  
fede@as.com  
El usuario es administrador

**Modificar usuario modal:**

Nombre completo  
Contraseña

[Cancelar] [Ok]

## Cambio de estado de solicitud

The screenshot shows a modal for changing the state of a request.

**Modificar estado**

Estado actual: ACEPTADA  
Seleccione el nuevo estado  
ENREVISION ▼

Descripción actual: asdadfa  
pasa a en revisión

[Ok]

# Alta de tipo de solicitud con campos adicionales

Administrar usuarios   Crear usuario   Administrar solicitudes   Crear solicitud   Consul

a@a
detalle
nombre
tel
detalle
9

Traer campos adicionales

Si se desea ingresar multiples valores hacerlo de la siguiente manera: valor1\*valor2\*valor3

Campo adicional Empresa de taxi de tipo TEXTO

Campo adicional Empresa de taxdfsdfi de tipo TEXTO

Campo adicional Empresa de taxizxczx de tipo TEXTO

Campo adicional Empresa de tsdfsdbsdaxi de tipo TEXTO

Campo adicional erroneo de tipo ENTERO

El valor debe estar comprendido entre 1 y 6

Campo adicional dsdfwds de tipo BOOL

Campo adicional errzdcsdoneo de tipo FECHA (DD/MM/YYYY)

Campo adicional Empresa de taxizxczx de tipo TEXTO



Campo adicional Empresa de tsdfsdaxi de tipo TEXTO

Taxi aeropuerto



Campo adicional erroneo de tipo ENTERO

El valor debe estar comprendido entre 1 y 6

2

Campo adicional dsdfwds de tipo BOOL  
false



Campo adicional errzdcdoneo de tipo FECHA (DD/MM/YYYY)

El valor debe estar comprendido entre 17/03/2021 y 12/03/22

17/04/2021

Campo adicional qwqs de tipo TEXTO

fghfg



Campo adicional sdfsd de tipo TEXTO

dfgdf



Campo adicional asdas de tipo ENTERO

Ingrese un valor entero 2

Campo adicional sdfsd de tipo ENTERO