

Universidad ORT Uruguay
Facultad de Ingeniería
Ing. Bernard Wand-Polak

Diseño de aplicaciones 2

Obligatorio 2

Pablo Pereira - 210162
Joaquin Touris - 178611

Entregado como requisito de la materia Diseño de
Aplicaciones 2

<https://github.com/ORT-DA2/obligatorio-touris-pereira>

25 de junio de 2020

Resumen

Esta nueva versión del sistema contiene todas las funcionalidades requeridas en la antigua versión y a su vez se le agrega las nuevas funcionalidades especificadas en la letra del segundo obligatorio para esta versión.

Para esta versión estaremos enfocándonos más en los nuevos cambios y su impacto. Observaremos lo siguiente:

- Descripción y justificación sobre las principales decisiones de diseño tomadas para esta nueva versión.
- Modelo de persistencia de datos.
- Justificaciones sobre Clean Code y buenas prácticas.
- Decisiones sobre testing y tecnologías utilizadas en el mismo.

Se ha logrado cumplir con todos los requerimientos especificidades para esta versión y no se han reportado bugs hasta el momento.

Creemos que el resultado de esta versión es bueno aunque sabemos que se puede continuar mejorando, debido a restricciones de tiempo no pudimos añadir un “tracking” a la interfaz de usuario la cual se había planeado con tiempo y la misma agregaría una nueva capa de usabilidad para el cliente.

Índice general

1 - Diseño y Arquitectura	4
1.1 - Sistema	4
1.1.1 - Domain	7
1.1.2 - DataAccess y Repository	8
1.1.3 - WebApi	9
1.2 - Diseño de Parser	9
1.3 - Diseño de Reports	12
1.4 - Diseño de la API REST	13
2 - Interfaz de usuario: Diseño y Arquitectura	13
2.1 - Angular	14
2.2 - Relación entre el sistema y el mundo real	15
3 - Clean Code	16
4 - Persistencia de datos	17
4.1 - Borrado Lógico	17
4.2 - Tecnologías	18
4.3 - Data Seeding	18
5 - Modelo de tablas	19
5.1 - Diagrama de tablas y sus relaciones	19
6 - Testing	20
6.1 - Pruebas unitarias y TDD	20
6.2 - Pruebas unitarias	21
7 - Excepciones	21
8 - Instalación	22
8.1 - Configuración	23
8.2 - Datos de prueba	23
9 - Anexos	23
9.1 - API	23
Administrator	24
Request	25
Area	26
Topic	28
Type	30
Additional Field	31
Login	33
Report	34

Parser	34
9.2 - Front-end	35
Reporte A	35
Reporte B	36
Importación del XML	36
Ingreso de nueva solicitud	37
Modificación de un administrador	37
Alta de tipo de solicitud con campos adicionales	38
Listado de solicitudes	38
Cambio de estado de una solicitud	39
10 - Bibliografía	39

1 - Diseño y Arquitectura

El diseño del sistema fue realizado siguiendo los principios de diseño SOLID. En todo momento se pensó en desarrollar el sistema lo más extensible y flexible dentro de lo que consideramos para que a futuro si se desean agregar funcionalidades nuevas, nuestro código no se vea afectado. Sabemos que el cambio es inevitable y por lo tanto muchas de nuestras discusiones iniciales sobre el diseño se basaron en esto.

Para apoyar esta extensibilidad, se siguió el principio de inversión de dependencias el cual se menciona en otra sección de la documentación. Para esto se utilizaron interfaces, logrando que que no existieran dependencias sobre implementaciones particulares y también que el costo de añadir/cambiar código sea bajo.

1.1 - Sistema

El sistema contiene 15 proyectos, de los cuales tres son únicamente de pruebas, y solamente uno es de Angular. Este es el resultado de la búsqueda de seguir y cumplir con los principios de diseño SOLID, también de cumplir con Clean Code, y el modelo de una Single Page Application.

Durante esta sección podremos observar distintas vistas del modelo 4 + 1.

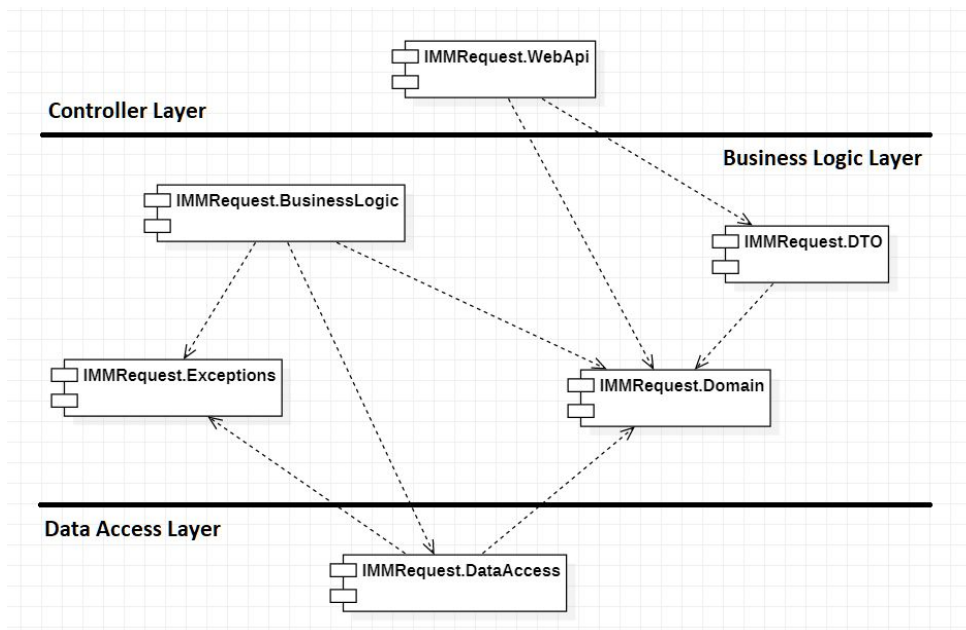


Figura 1.1: Diagrama de componentes, vista de desarrollo

De la misma forma que se mencionó en la entrega anterior, se tuvieron en cuenta los principios SOLID en conjunto con algunos GRASP relacionados a la separación de módulos y también a la asignación de responsabilidades.

También se consideraron principios a nivel de paquetes propuestos por Robert C. Martin.

A continuación observaremos un diagrama a nivel de paquetes del sistema seguido por el análisis de métricas generado por nDepend.

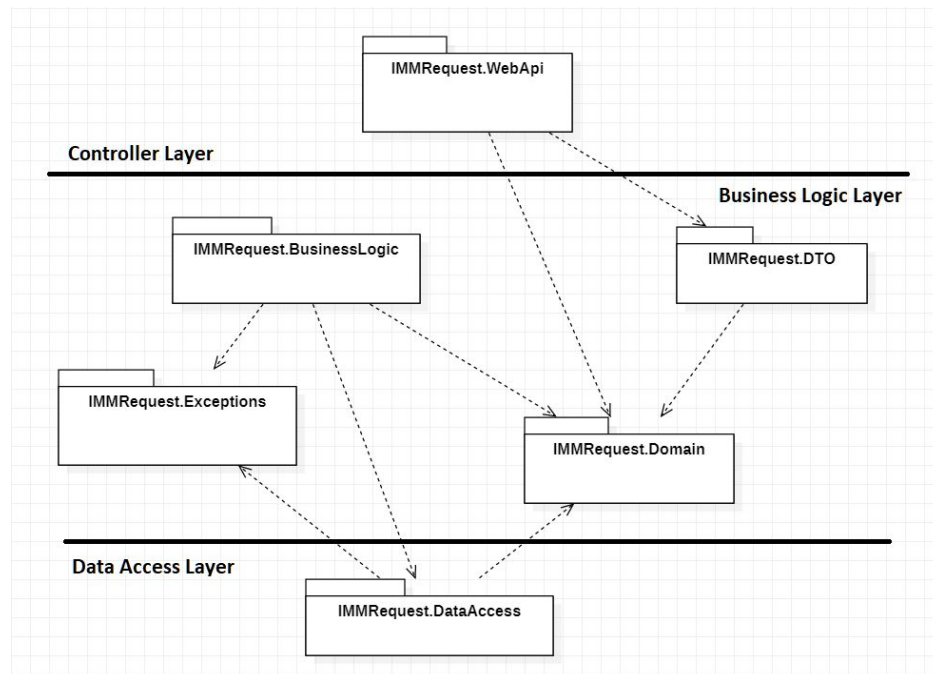


Figura 1.2: Diagrama de paquetes, vista de desarrollo

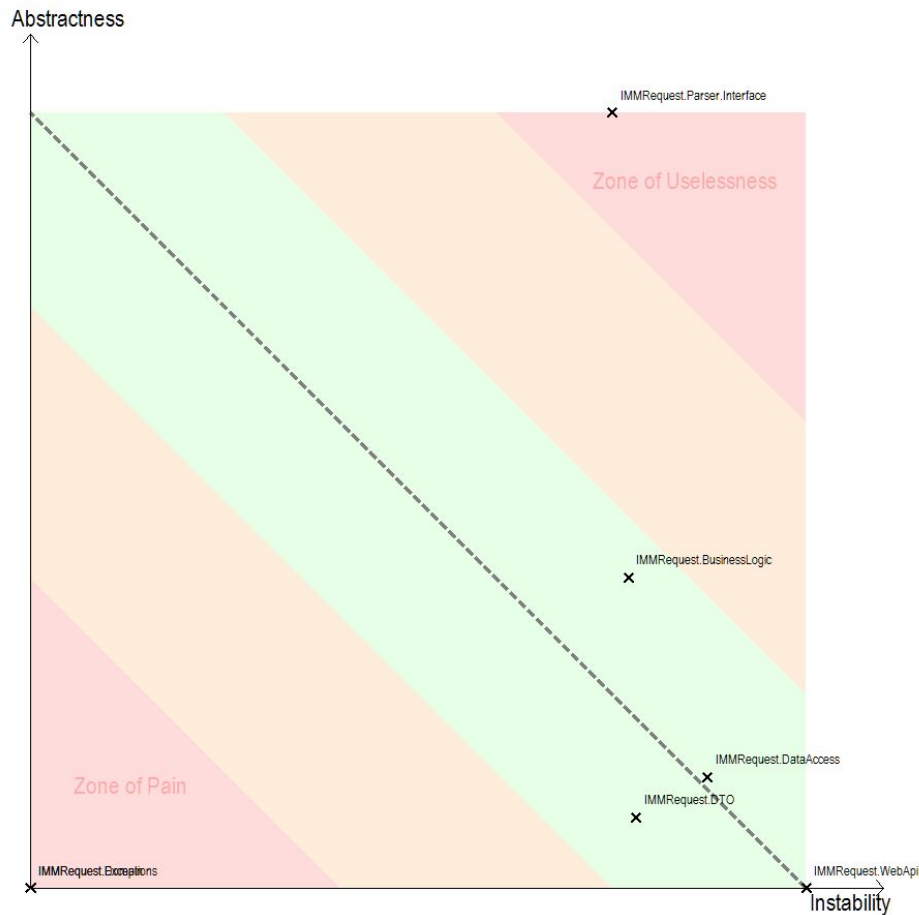


Figura 1.3: Gráfica comparativa abstracción y seguridad nDepend

Si siguiendo los principios a nivel de paquetes, las dependencias deben ir en dirección hacia la estabilidad, por lo tanto si una clase A que depende de otra clase B, esta última debe ser más estable que la primera.

Otro principio menciona que los paquetes que son más estables tienden a ser más abstractos, ya que se debe tener en mente que otros paquetes dependen de él.

Lo dicho anteriormente se deriva de los principios OCP y DIP.

Por lo tanto, el sistema siempre debe buscar balancear el hecho de tener paquetes que dependan de otros y de que estos sean abstractos.

No tiene sentido tener un paquete que sea demasiado abstracto si no existe otro paquete que dependa de él. En otras palabras, no tiene sentido tener un paquete que sea muy abstracto al que pueda realizar modificaciones sin preocuparme si no existe otro paquete que dependa del mismo.

Por otro lado, debemos tener cuidado con los paquetes poco abstractos que poseen muchos otros dependientes, ya que realizar cambios sobre el mismo posee cierta

complejidad, lo mismo sucede con la extensibilidad del mismo, se vuelve complejo añadir nuevo comportamiento sin alterar el código relacionado al mismo. Esto es malo porque cualquier modificación genera un impacto a todos los paquetes que dependen de él.

1.1.1 - Domain

En este paquete se encuentra la mayor parte del código relacionado a la lógica del negocio, por lo que se considera ser un paquete de más alto nivel.

Aquí se encuentran las clases que poseen nombres que son sustantivos relacionados al lenguaje utilizado en el dominio del problema.

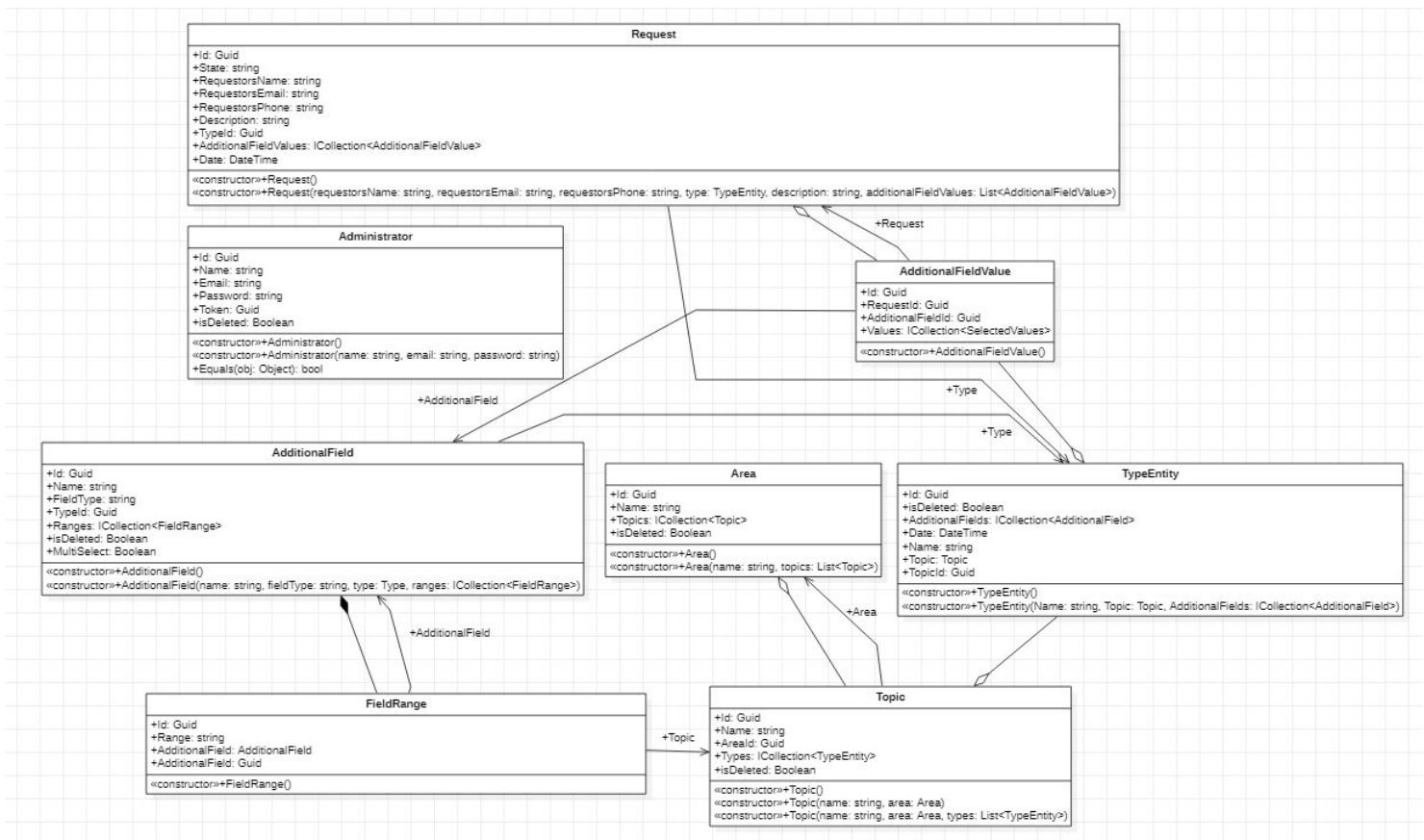


Figura 1.4: Diagrama de clases IMMRequest.Domain, vista lógica

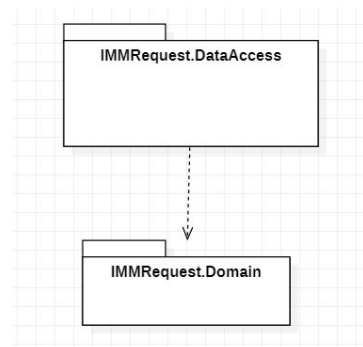
Como podemos observar, este paquete recibió algunos cambios pero no muy significativos, se realizaron refactor sobre las clases TypeEntity y FieldRange, también se quitaron los métodos equals de la mayoría de las clases.

1.1.2 - DataAccess y Repository

Estos paquetes representan el acceso a datos y la persistencia del sistema. Por un lado tenemos a IMMRequest.DataAccess.Contexts que es donde encontramos el contexto de la base de datos y toda la configuración se da utilizando Fluent API.

Este contexto es utilizado por todo el paquete IMMRequest.DataAccess.Repositories.

Tanto DataAccess.Contexts como DataAccess.Repositories utilizan a IMMRequest.Domain. Se decidió utilizar los mismos elementos del dominio para realizar la persistencia ya que no queríamos tener código duplicado para lograr cierta independencia del dominio.



Lo mencionado anteriormente también corresponde a la entrega anterior pero consideramos importante mencionarlo para tener presente las decisiones tomadas.

Se utilizaron dos patrones de persistencia, por un lado el *Repository Pattern* y por otro el *Unit of Work*. Ambos fueron utilizados con el objetivo de crear una capa de abstracción entre la persistencia en sí y lógica relacionada al negocio.

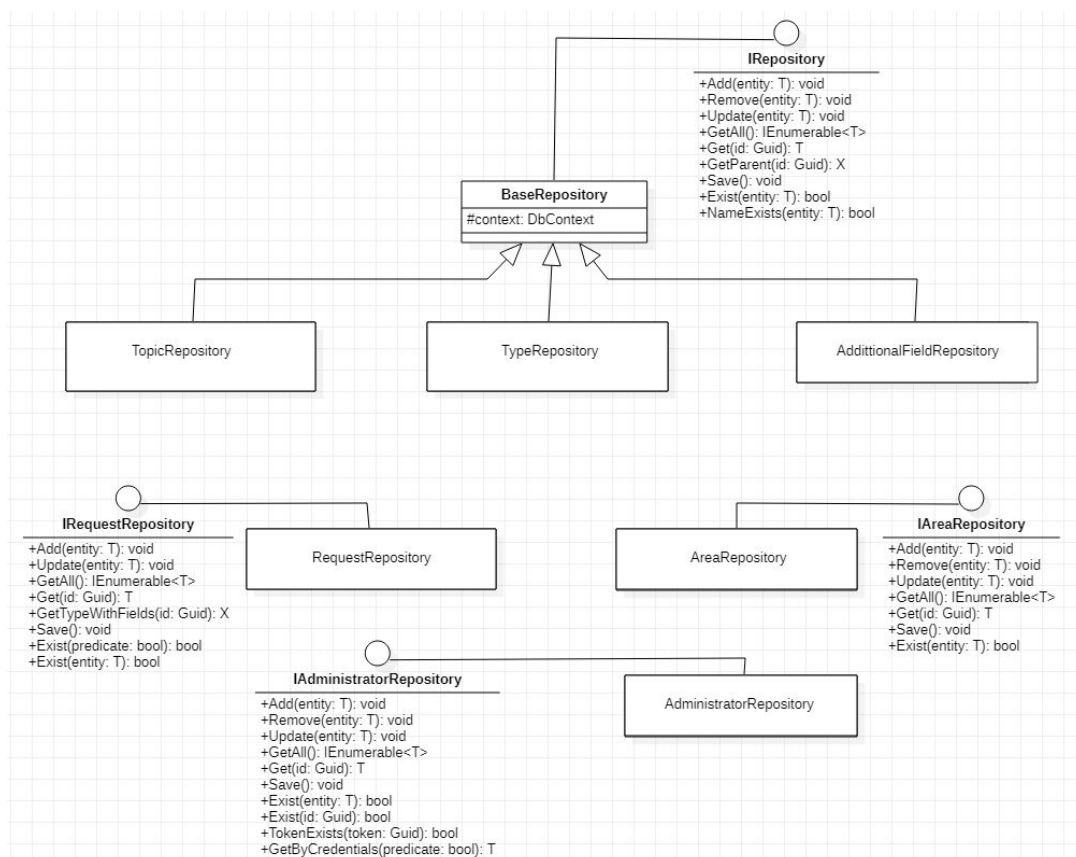


Figura 1.1.2: Repository Pattern y Unit of Work, vista lógica

1.1.3 - WebApi

Este paquete está encargado de exponer la API al desarrollador de forma en que puede lograr una comunicación con el sistema a través de la web, cumpliendo con las características REST que se mencionaron en la entrega anterior.

A este paquete se le agregaron dos nuevos controladores, *ParsersController* y *ReportsController*, a continuación podemos observar a ambos utilizando un diagrama.

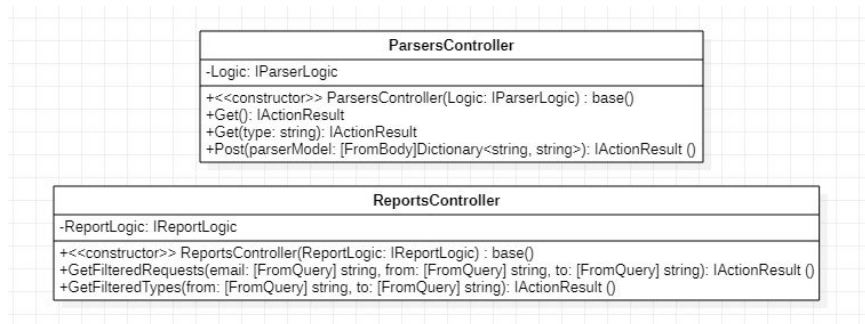


Figura 1.1.3: Diagrama de clases *ParsersController* y *ReportsController*, vista lógica

1.2 - Diseño de Parser

El Parser es el módulo encargado de cumplir con el requerimiento de importación de Áreas, Tipos y Temas. Esta importación actualmente es posible mediante los formatos requeridos: Json y XML, aunque el backend está diseñado para poder soportar otros formatos a futuro.

Para esto se ha utilizado el patrón de diseño Reflection cómo se explica a continuación:

Todos los archivos que se deseen importar deben estar dentro del path “*..WebApi/ImportFiles*” de manera de mantener una estructura de carpetas ordenada. (de todas maneras, el archivo se puede encontrar en una ruta diferente, siempre y cuando la ruta sea correcta)

A su vez, dentro de la carpeta *WebApi* hay una carpeta llamada *Parsers* que contiene las .dll compiladas de las clases importadoras. Esto quiere decir que no es necesario volver a recompilar la aplicación para poder leer nuevos Assemblies, ya que alcanzaría con agregar la nueva dll de una de las clases importadoras dentro de esta misma carpeta, incluso en tiempo de ejecución.

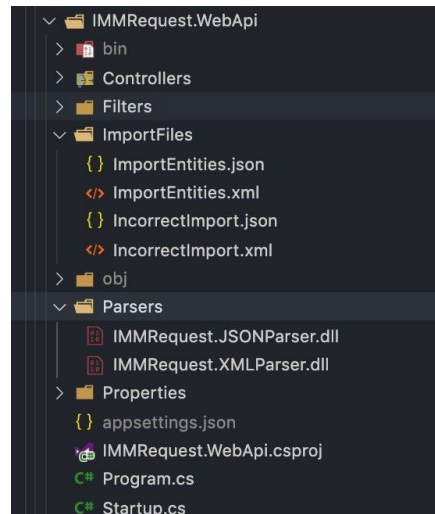


Figura 1.2.1: Organización de carpetas IMMRequest.WebApi

Esto es así dado que *ParserLogic* luego se encarga de recorrer las librerías dentro de la carpeta, obteniendo todos los tipos que haya dentro del Assembly y de esa manera se van creando las instancias *IParseables* necesarias.- Ver figura Figura 1.2.2.

```
2 references
private static IEnumerable<IParseable> GetParsers()
{
    List<IParseable> parsers = new List<IParseable>();
    string[] folder = Directory.GetFiles(@"../IMMRequest.WebApi/Parsers", "*.dll");
    foreach (string library in folder)
    {
        var libFile = new FileInfo(library);
        Assembly libAssembly = Assembly.LoadFile(libFile.FullName);
        IEnumerable<Type> implementations = GetTypesInAssembly<IParseable>(libAssembly);
        IParseable parser = (IParseable)Activator.CreateInstance(implementations.First());
        parsers.Add(parser);
    }

    return parsers;
}
```

Figura 1.2.2: Método GetParsers()

Como se observa en el código arriba, se leen las librerías, se cargan los Assembly y sus tipos, se toma la primera implementación que haya del tipo *IParseable* en ese assembly y se agrega a una lista de parsers, que es finalmente lo que devuelve el método *GetParsers*. De esta manera, ya queda creada la instancia de la clase para poder ser utilizada, dado que cumpliendo con este contrato nos aseguramos de que el importador tenga lo necesario para ejecutarse correctamente.

Siguiendo en esta línea y como se puede observar en la Figura 1.2.3 la interfaz “*IParseable*” es la encargada de definir el contrato que todo parser va a tener que implementar si desea poder cumplir con la funcionalidad de importación.

Los “Parsers” (*JSONParser*, *XMLParser*) implementan la interfaz *IParseable* y de esta misma manera, cada nuevo convertidor deberá implementar la interfaz.

Esto permite realizar importaciones por ejemplo desde conexiones a bases de datos externas, ya que con la implementación de esta clase importadora podrá parsear la información utilizando sus propios parámetros definidos en los constructores de las clases, como por ejemplo su propio connection string, solicitando estos datos dinámicamente al usuario final, dependiendo del tipo de importador seleccionado.

Dicho Parser a su vez se compone de una interfaz llamada “*IParserLogic*” que se utiliza de manera interna para implementar la segregación de la lógica de negocio, por lo que no queda en ningún momento expuesta a los importadores.

En la implementación de la interfaz como se observa en Figura 1.2.4 , las 3 lógicas, AreaLogic, TopicLogic y TypeLogic, se implementan con el objetivo de agregar sus correspondientes listas (de áreas, de temas y de tipos) a cada una de sus clases, de manera de reutilizar la implementación de la creación de las mismas.

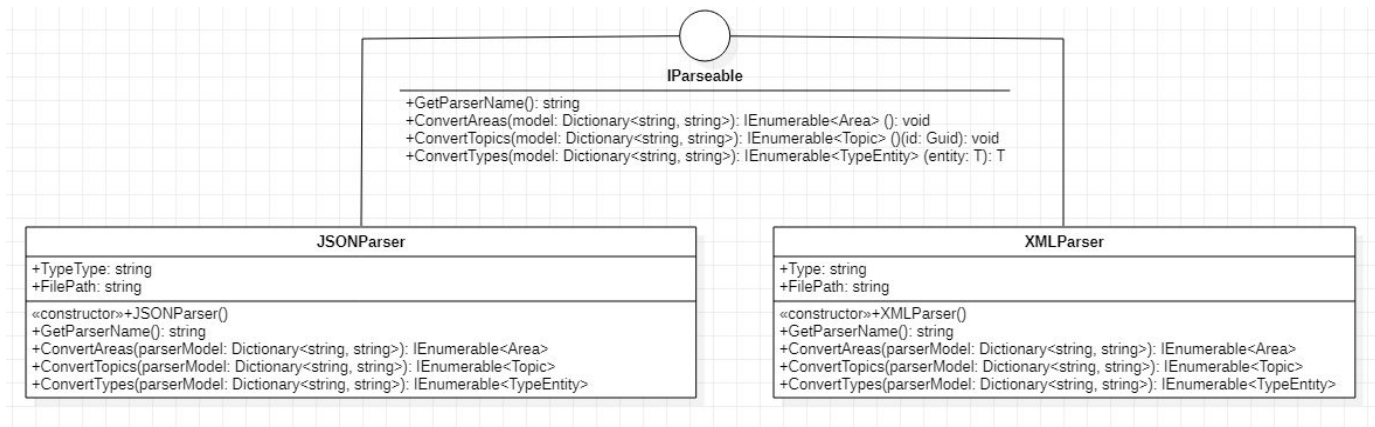


Figura 1.2.3: Diagrama de clases sobre IParseable, vista lógica

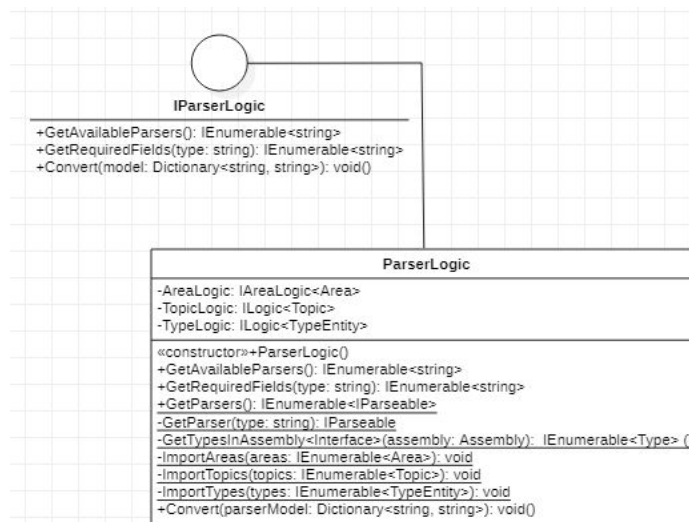


Figura 1.2.4: Diagrama de clases sobre Parser, vista lógica

1.3 - Diseño de Reports

Otra de las funcionalidades solicitadas, es la de crear reportes de dos tipos. Para esto se ha creado una interfaz “IReportLogic” con su correspondiente implementación en la clase ReportLogic.

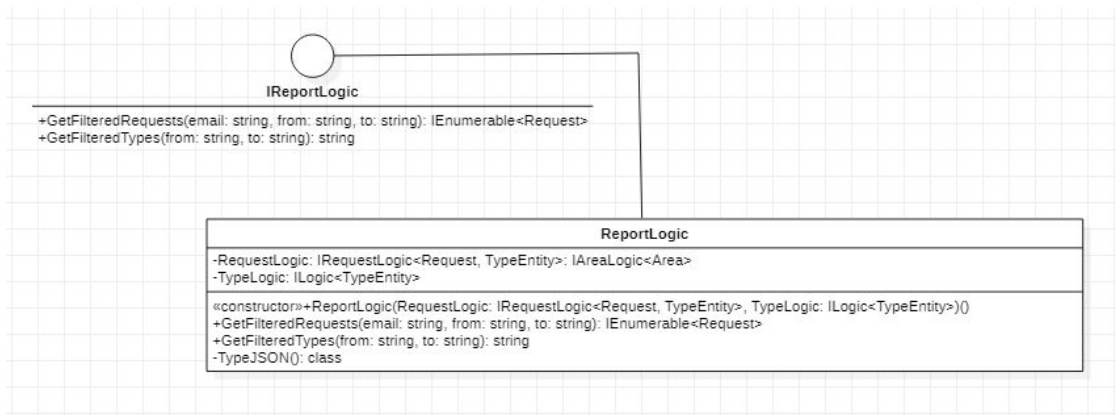


Figura 1.3: Diagrama de clases sobre Report, vista lógica

Esta funcionalidad no tiene mayor complejidad ya que son simplemente dos llamados, uno para cada reporte y cada uno con sus correspondientes parámetros. Estos son:

- **GetFilterRequests:** Recibiendo un email y dos fechas(From y To), filtra todas las requests creadas por ese email, dentro de las fechas especificadas.
- **GetFilterTypes:** Recibiendo dos fechas (From y To) filtra las requests creadas en ese rango de fechas.

Este segundo método, para cada una de las requests encontradas, toma los tipos dentro de cada una de ellas, las agrega a una lista de tipos, las agrupa por id, itera sobre cada una de las agrupaciones y crea una instancia de una clase interna de clave-valor con la cantidad y el tipo asociado, con todas sus propiedades.

Al momento de devolver esta información al controlador, se serializa a *JSON* y se le devuelve al controlador, quien devuelve como una lista de objetos al frontend, donde cada uno de los objetos tiene una cantidad y un tipo asociado.

1.4 - Diseño de la API REST

La API se diseñó teniendo en cuenta los tanto los requerimientos dados para la versión anterior como los nuevos requerimientos que surgieron para esta versión en conjunto a lo que creímos que como desarrolladores la API debería tener para soportar las condiciones actuales y que cumpla con REST, y a la vez tener cierto nivel de mantenibilidad y extensibilidad.

A seguir será posible observar el diseño actual de la API con un enfoque en la nueva versión en donde se cubrirán los recursos relacionados a los mismos. Decidimos tratarlo de esta forma ya que se agregaría un nivel de complejidad ya que no existe mucha información sobre los controladores.

1.4.1 - Controladores y Recursos

Según lo mencionado anteriormente, las peticiones de la API se diseñaron bajo la noción de los controladores ya que son los encargados de recibir las peticiones y devolver las respuestas al cliente cuando corresponden. Estos controladores se crearon a partir de la necesidad lógica del sistema que surgió con los nuevos requerimientos para la versión.

- **Parser:** En este caso el controlador del parser actúa simplemente de ruteo. Cuando hace un `GetParserTypes`, hace todos los tipos de parses que hay y cuando hace un `/ID` devuelve los parámetros necesarios dependiendo del parser en cuestión.
- **Report:** El controlador asociado a los reportes tiene dos métodos (`GetFilterRequests`, `GetFilterTypes`), donde lo importante a destacar es el parámetro *root*, ya que es quien le indica hacia qué reporte tiene buscar y por lo tanto qué función invocar. Cada uno va a obtener esos parámetros y posteriormente lo envía a la capa lógica para obtener la información asociada al reporte como fue explicado en la sección 1.3.

2 - Interfaz de usuario: Diseño y Arquitectura

Aquí detallaremos la implementación, diseño y arquitectura de la interfaz de usuario el cual fue realizado utilizando el framework Angular. Comenzaremos con una descripción del diseño de la aplicación desde el nivel de usuario y luego veremos detalles relacionados al código en sí el cual se apoyó en Angular.

Para esta versión se desarrolló una *Single Page Application (SPA)* utilizando *Angular* lo cual nos permite renderizar los componentes de la página sin la necesidad de recargar la página por completo. Esto permite al usuario navegar/interactuar con la página de una forma más fluida y rápida al no tener que volver a cargar componentes estáticos o con pequeños cambios por completo. Los componentes dinámicos se renderizan únicamente cuando es necesario.

Lo mencionado anteriormente favorece a la usabilidad ya que algunas de las funcionalidades de la aplicación no son muy amigables para el usuario, al seguir los lineamientos y guías de *Angular* creemos que la parte gráfica es algo fundamental para la aplicación y por lo tanto tratamos de realizarla de forma más amigable y natural posible.

2.1 - Angular

En esta sección destacaremos algunas de las implementaciones de diseño realizadas a través en *Angular*.

La organización de la aplicación se divide entre módulos, componentes y servicios. A continuación se explicará en mayor detalle cada uno de ellos.

- **Módulos:** En el módulo encontramos todos los componentes que se utilizaron en el proyecto, tanto propios como de terceros. Para la aplicación nos apoyamos en algunas librerías externas tales como *material-ui*, *ngx-bootstrap* y *font-awesome*, lo cual nos permitió centrarnos más en la lógica del sistema que en aspectos gráficos y mismo así se logró implementar la interfaz de manera muy rápida y atractiva. Durante el desarrollo decidimos apoyarnos tanto en material como en bootstrap y aunque son muy similares, en algunos casos optamos por una mezcla entre componentes de estas librerías con el fin de lograr interfaces más atractivas.
 - **Rutas, restricciones y permisos:** El sistema cuenta con un módulo de ruteo en el cual según el url actual podemos indicar qué componentes se deben mostrar. Este módulo también nos permite aplicar restricciones a través mediante el uso de *Guards*, las cuales interceptan el ruteo antes de ser realizado, aplican las políticas indicadas y luego autorizan o no la realización del ruteo.

Para la aplicación se implementó la siguiente guarda:

 - **Auth Guard:** Esta guarda protege al sistema en caso de que se trate de acceder funcionalidades que requieren estar ingresado al sistema. Si se trata de acceder sin autorización, esta guarda realiza una redirección hacia la interfaz de login.

Algo destacable es el hecho de que existen rutas por defecto en las que se renderizan componentes que no requieren estar ingresado en el sistema.

- **Modelos:** Son clases que nos permiten tener lógica interna utilizados tanto para la transferencia como para el recibimiento de datos desde la Api, las clases mapean las respuestas que vienen del backend y las convierte en entidades conocidas a través del uso de *Typescript*. En este sistema los modelos son instanciados tanto en los servicios y luego son utilizados en los componentes o viceversa.
- **Componentes:** Los componentes se fueron creando e implementando tanto en base a los requerimientos como en la necesidad para ir creando el flujo del sistema.

Algunos componentes se agruparon según la funcionalidad a la que están relacionada, por eso es posible observar a los componentes relacionados al manejo de un administrador dentro de la carpeta `admin` y a su vez a la entidad a la que están relacionados. Esto genera una mayor reusabilidad a futuro aparte de aumentar su mantenibilidad y simplicidad en lectura.

Por otro lado también tenemos a componentes como `navbar` y `footer` los cuales se utilizan a lo largo de toda la aplicación en conjunto con otros componentes y de esta forma se logra encapsularlos.

- **Servicios:** Los servicios se encargan de realizar las peticiones al *backend*, mapear la respuesta en objetos y enviar estos objetos a donde sea necesario para que continúe el procesamiento de esta información. Para esto se utilizan los observadores. En otras palabras, las llamadas que utilicen la información de manera asincrónica deberán estar suscritos al evento, el cual notificará a los suscriptores cuando deberán actualizar la información deseada.
- **Inyección de dependencias:** Dada la facilidad que Angular nos provee para el uso de inyección de dependencias en nuestros componentes y servicios, decidimos aprovecharnos de esto y realizamos interacciones con estos objetos sin preocuparnos por su instanciación.
- **Pipes:** Utilizamos pipes para realizar filtros a nivel de usuario. Estas directivas personalizadas no permiten realizar un filtrado directamente en el html de manera muy sencilla. Se utilizaron en todos los componentes relacionados al manejo de información por parte del administrador.
- **Almacenamiento local:** Utilizamos el *Local Storage* para almacenar el token generado una vez que un administrador ingresa en el sistema. Esta propiedad del browser nos permite almacenar y consultar de manera muy sencilla la información que deseamos guardar. Esto en conjunto con la guarda relacionada a la autorización nos permitió que el administrador pudiera acceder a todas las funcionalidades mientras que el usuario no ingresado tuviera restricciones de acceso sobre algunas de estas entidades.

2.2 - Relación entre el sistema y el mundo real

Se intentó desarrollar una aplicación que considerara la mayor naturalidad para el usuario posible, para esto nos apoyamos en las tendencias de la comunidad del framework sobre algunos componentes y los utilizamos.

Un claro ejemplo se puede visualizar en la selección de dos fechas en ambos reportes, siguiendo el estilo utilizado por Airbnb, la selección consiste en elegir una fecha en cada calendario (izquierda y derecha) y de esta forma se marcarían todas las fechas que se encuentran entre las dos que seleccionamos. Esto también trae como ventaja cierta prevención de error ya que deja claro al usuario cual es el rango de fechas contenido para el reporte.

3 - Clean Code

Para el desarrollo de la solución, se aplicaron las siguientes prácticas del libro *Clean Code* y se siguieron los principios *SOLID*.

- **Nombres mnemotécnicos:** Durante el desarrollo de seleccionaron los nombres que más representaban al propósito para todas las propiedades, métodos, clases, paquetes, parámetros y variables.
- **Pasaje de argumentos en funciones:** En todo momento se buscó utilizar la menor cantidad de parámetros posibles en los métodos, el libro *Clean Code* recomienda un máximo de 3 parámetros y en nuestro sistema lo respetamos.
- **Uso de comentarios:** Se buscó utilizar la mínima cantidad posible de comentarios, debido a los demás puntos mencionados de *Clean Code*, se logró que el código fuera entendible por sí mismo, por lo que no debería existir la necesidad de utilizar los comentarios, indiferente de su propósito.
- **Formateo:** Para que el código sea más legible, se buscó que se cumplan cierto estándares de programación, por eso las clases tienen declarado al principio sí necesarios los enums, luego se definieron las propiedades, seguido por el constructor y por último los métodos, estos últimos se declaran primero el método público seguido por los métodos privados que utiliza.
- **Manejo y uso de excepciones:** Para el proyecto decidimos utilizar excepciones creadas y customizadas por nosotros para manejar los estados de error, estas se muestran al usuario con un lenguaje lo más natural posible cuando se realiza una acción incorrecta. Al ser creadas y customizadas por nosotros, nos permite tener el control sobre las mismas.
- **Single responsibility principle:** Cada clase de nuestra solución posee una única responsabilidad, por lo tanto, cada clase posee un único motivo para ser modificada. Lo mismo aplica a todos los métodos del proyecto, esto hace que sea fácil el encontrar y resolver bugs.
- **Open/closed principle:** Las entidades del sistema, tanto módulos como clases están abiertas a extensiones pero cerradas a modificaciones, debido a que durante el desarrollo se fueron realizando cambios y surgiendo modificaciones necesarias, se trató de respetar desde el inicio este principio para que al aplicar los cambios no surgieran efectos indeseados en otras partes del sistema.
- **Interface segregation principle:** Los clientes del sistema no están obligados a depender de interfaces que no necesitan usar. Las interfaces desarrolladas son pequeñas y más específicas por lo que los clientes solo necesitan saber lo que les interesa.
- **Dependency inversion principle:** Al utilizar este principio, los módulos del sistema se desacoplan entre sí, los módulos de alto nivel no dependen de módulos de niveles inferiores, sino que dependen de abstracciones, en nuestro caso utilizamos interfaces para representar esa dependencia.

Las interfaces no dependen de las implementaciones concretas (detalles) pero si lo contrario.

Como ventaja nuestro código se hace más testeable y mantenible.

- Ejemplo: La capa de Business Logic no depende de la capa de Data Access, sino que depende de una Interfaz IRepository.
- **Dependency injection pattern:** En conjunto con el principio de inversión de dependencia, utilizamos este patrón para lograr un mayor desacoplamiento. Las clases de nuestro sistema no crean los objetos de otras clases que necesitan, sino que se utiliza la clase Startup como “contenedor” y esta se encarga de inyectar de forma dinámica las implementaciones deseadas.
- **Mocking:** Se utilizó *Mock* en los tests para simular el comportamiento de objetos reales externos de forma controlada evitando que se prueben sus dependencias, para nuestro caso se utilizó la herramienta llamada *Moq*.
- **Authorization Filters:** En el sistema se utilizaron estos filtros con el fin de autenticar y establecer una política de seguridad para nuestra aplicación web. Utilizamos un sistema de roles para que se logre acceder a endpoints más restringidos, para este sistema se definió únicamente el rol Administrator.

4 - Persistencia de datos

Se utilizó *Entity Framework* con la modalidad *Code First* para la persistencia de las clases del sistema. Esto nos permitió modelar el dominio y desarrollar el código sin preocuparnos por la creación de las tablas en la base de datos, a través de un model builder que inspecciona las clases que se manejan en el contexto y utiliza un conjunto de reglas para determinar cómo se representan en la base de datos.

También nos apoyamos en *LinQ* para manejar de manera sencilla las colecciones de datos. Esto se logra utilizando los métodos de extensión que nos ofrece *LinQ* en conjunto con expresiones lambda.

4.1 - Borrado Lógico

Para esta versión del sistema vimos la necesidad de pasar desde un borrado físico hacia un borrado lógico.

Este tipo de borrado en la base de datos nos permite mantener un historial/tracking sobre los datos una vez que se marcan como borrados.

Esto también nos trae otra ventaja, no existe la preocupación por realizar la eliminación en cascada a través de las distintas tablas en la base de datos que hacen referencia a la fila que se desea eliminar.

Por otro lado, una desventaja que consideramos significativa es el tamaño de la base de datos al utilizar este tipo de borrado, se requiere un mayor tamaño ya que no se eliminarían los datos de la base de datos sino que se marcaría en una columna de la tupla si la misma fue eliminada o no.

4.2 - Tecnologías

- **Entity Framework Core:**
Esta tecnología de acceso a datos para .NET Core y .NET Framework nos sirve como mapeador de objetos relacionales (ORM) ayudándonos con el acceso a los datos haciéndolo muy sencillo.
- **Sql Server:**
Es un sistema de gestión de base de datos relacional de Microsoft que se utilizó para la persistencia de las entidades y sus datos que fueron requeridos para esta versión del sistema.
- **Microsoft Sql Server Management Studio:**
Se utilizó esta herramienta de Microsoft para la manipulación y gestión de la base de datos del sistema. Permite visualizar las distintas tablas y ver rápidamente las tuplas de información que estas contienen. La utilizamos en el equipo de desarrollo que posee como sistema operativo una de las versiones de Windows.
- **DBeaver:**
Para el equipo que posee iOS decidimos utilizar la herramienta DBeaver, la misma es un editor de SQL multiplataforma que también permite gestionar y visualizar la base de datos de forma intuitiva.
- **Docker:**
Es un proyecto Open Source que permite la automatización del despliegue de aplicaciones a través de la utilización de contenedores de software. Esta aplicación permitió que fuera posible el uso de DBeaver en el equipo con iOS.

4.3 - Data Seeding

Utilizamos la feature de *EF Core* llamada *Model Seed Data* para realizar el proceso de poblar la base de datos del sistema con un conjunto de datos iniciales.

El proceso de seeding data puede ser asociado con un tipo de entidad como parte del modelo de configuración, esto hace que el *EF Core Migrations* sea capaz de reconocer qué conjunto de datos debe añadir, modificar o borrar al momento de pasar a una nueva versión del modelo.

A continuación podemos observar como se refleja dicha configuración de data seeding en el código del sistema.

```

#region Super Admin
    Guid baseToken = new Guid();
    Administrator admin = new Administrator() { Name = "Administrator", Email = "main@admin.com", Password = "root", Token = baseToken };
#endregion

#region Default Areas
    Area espaciosPublicosCalles = new Area() { Name = "Espacios Públicos y Calles" };
    Area limpieza = new Area() { Name = "Limpieza" };
    Area saneamiento = new Area() { Name = "Saneamiento" };
    Area transporte = new Area() { Name = "Transporte" };
#endregion

#region Default Topics
    /* Espacios Públicos y Calles */
    Topic alumbrado = new Topic() { AreaId = espaciosPublicosCalles.Id, Name = "Alumbrado" };
    Topic arboladoPlantacion = new Topic() { AreaId = espaciosPublicosCalles.Id, Name = "Arbolado y plantación" };
    Topic equipamientoUrbano = new Topic() { AreaId = espaciosPublicosCalles.Id, Name = "Equipamiento urbano" };
    Topic fuentesGraffitisInstalaciones = new Topic() { AreaId = espaciosPublicosCalles.Id, Name = "Fuentes, graffitis e instalaciones" };
    Topic otrosPublicoCalles = new Topic() { AreaId = espaciosPublicosCalles.Id, Name = "Otros" };

    /* Limpieza */
    Topic estadoContenedores = new Topic() { AreaId = limpieza.Id, Name = "Estado de los contenedores" };
    Topic problemaslimpieza = new Topic() { AreaId = limpieza.Id, Name = "Problemas de limpieza" };
    Topic solicitudRetiro = new Topic() { AreaId = limpieza.Id, Name = "Solicitud de retiro de poda, escombros o residuos de gran tamaño" };
    Topic otrosLimpieza = new Topic() { AreaId = limpieza.Id, Name = "Otros" };

```

Figura 4.3.1: Data Seeding

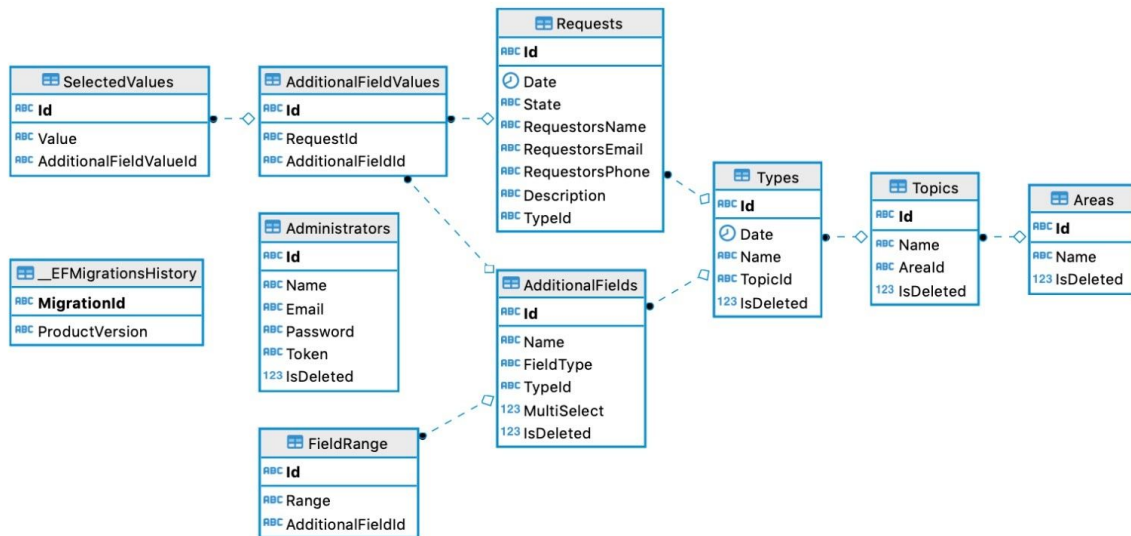
5 - Modelo de tablas

5.1 - Diagrama de tablas y sus relaciones

Con respecto a la primera versión del sistema, no se han agregado nuevas tablas al modelo. Sin embargo sí se añadió la columna `isDeleted` relacionada al Soft Delete a las tablas que se deseó tener un historial/tracking sobre el borrado.

También se añadió la columna `Date` tanto para la tabla relacionada a la Solicitud como para la tabla relacionada al Tipo para conocer su fecha de creación.

Por último se añadió la columna `MultiSelect` en la tabla relacionada al Campo Adicional para saber si el mismo posee o no más rangos.



6 - Testing

6.1 - Pruebas unitarias y TDD

Se utilizó *Test Driven Development* para el desarrollo del sistema. El mismo consiste en el ciclo *Red-Green-Refactor*.

- Red: Escribir la prueba antes de desarrollar la implementación, esto hace que el test falle y de ahí deriva el nombre "Red".
- Green: Implementar únicamente el código necesario para que la prueba pase.
- Refactor: Luego de que nuestro código pase el test, se examina el mismo buscando mejoras a realizar.

Para el sistema se utilizó la herramienta de Microsoft para *UnitTesting* y también la herramienta *Moq* para realizar mocking.

Durante el desarrollo de las mismas se siguió el principio *F.I.R.S.T.* (Fast, Independent, repeatable, self-validating, timely), esto ayuda a mantener un estándar y trae como ventaja un nivel alto de calidad en las pruebas.

Al utilizar *Test Driven Development* y desarrollar las pruebas antes que la implementación del método, fue posible pensar en todos los posibles caminos que tendría el método.

6.1.1 - TDD en SourceTree

Se utilizó el cliente GUI llamado SourceTree manejar el repositorio git. Desde este herramienta es posible observar el seguimiento de los pasos de la realización de TDD en el proyecto.

6.2 - Pruebas unitarias

Como mencionado anteriormente, durante el desarrollo del sistema se utilizaron pruebas unitarias y se espera que exista al menos un test que pruebe un camino para un método dado.

Estas pruebas deben probar una única funcionalidad, para esto nos apoyamos en mock cuando necesario. También para las pruebas decidimos seguir los principios *F.I.R.S.T.* para hacerlo lo más simple posible.

Al utilizarlo pudimos observar varias ventajas, las cuales listamos a continuación.

- Trabajo ágil: Nos permitió detectar errores a tiempo, de forma en que reescribir el código o corregir sus errores nos fue muy fácil, reduciendo su costo casi a cero.
- Calidad del código: Al seguir *F.I.R.S.T.* y realizar las pruebas antes que la implementación del método, nuestro código se mantuvo limpio y de alta calidad.
- Detectar errores: Nos permitió detectar errores e incluso nos mostró nuevas salidas que podrían fácilmente conducirnos a errores.
- El diseño: Al crear las pruebas primero, nos fue más fácil saber cómo enfocar el diseño y entender qué deberíamos validaciones realizar para lograr lo esperado.
- Reducción de costos: Fue necesario desarrollar menos código debido a que los errores se detectaron rápidamente.

Utilizar las pruebas unitarias nos permitió probar pequeñas partes del sistema, pero no la integración total del mismo, por lo que también realizamos pruebas de integración, las cuales describiremos a continuación.

7 - Excepciones

Para esta instancia del sistema no se han realizado cambios sobre el manejo de excepciones, por lo que se mantiene igual a la instancia anterior.

Cada capa del sistema es capaz de capturar las excepciones definidas en las capas descendientes de tal forma en que atrapan únicamente excepciones del padre.

De dicha forma se evita realizar RTTI sobre las excepciones, generando código más extensible y mantenible.

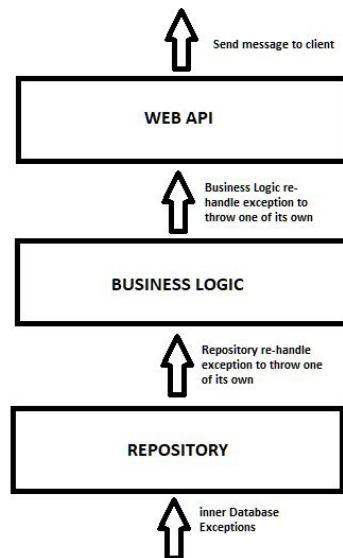


Figura 7.1: Jerarquía de excepciones

8 - Instalación

Por último se mostrará como la instalación del sistema. Para esto se mostrará a continuación el diagrama de entrega de la aplicación la cual se separa en tres partes:

1. **Cliente Web:** Se encarga de realizar las peticiones hacia la API.
2. **Servidor API:** Es el encargado de exponer la API hacia los clientes y procesar tanto las peticiones recibidas como las enviadas.
3. **Servidor BD:** Encargado del almacenamiento de toda la información requerida por el sistema.

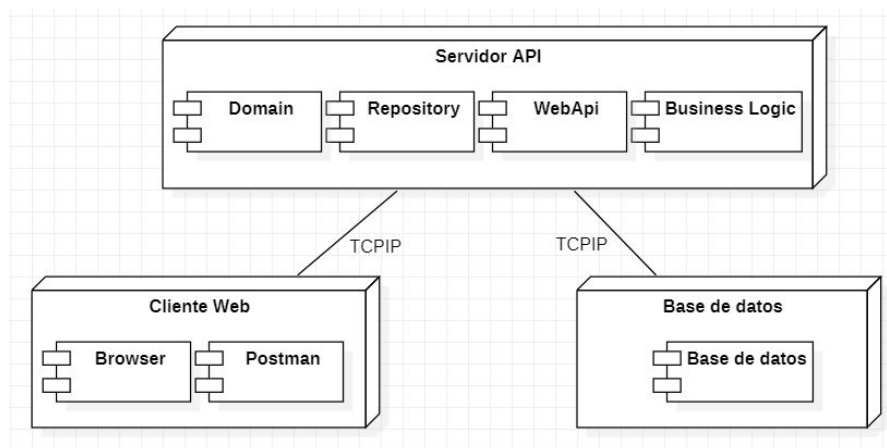


Figura 8: Diagrama de despliegue

Como podemos observar en el diagrama, el Servidor API no contiene todos los componentes que realmente posee el sistema debido a la cantidad de los mismos, para este diagrama decidimos mostrar la información de manera resumida para simplificar la idea sobre los distintos nodos.

8.1 - Configuración

Para realizar la configuración del sistema es necesario realizar los siguientes pasos:

1. Dirigirse al archivo *appsettings.json* dentro de la carpeta *publish* y cambiar el *connection string* por el string de conexión del motor de base de datos que se utilizará para almacenar la información.
2. Importar el archivo .bak de la base de datos que se generó previamente utilizando el SQL Server Management Studio u otro software con la misma utilidad.
3. Utilizar el instructivo de IIS que se encuentra en aulas para realizar el deploy.

8.2 - Datos de prueba

Como en la versión anterior, junto al código del sistema también se entregan dos archivos .bak que poseen datos para ser importados en la base de datos.

- **sinDatos.bak**: Este archivo .bak posee únicamente un administrador, el cual se puede utilizar para ingresar e insertar nuevos datos a través de las demás funcionalidades que el sistema posee.

El administrador mencionado posee la siguiente información:

GET /api/login

```
{
  "email": "main@admin.com"
  "password": "root"
}
```

La petición nos retornará un token que se utilizará internamente para manejar el acceso a las distintas funcionalidades según el rol.

- **conDatos.bak**: Esta es una base con datos de prueba ingresados con el fin de facilitar el flujo por el sistema y poder utilizar todas las funcionalidades sin mucha dificultad.
También contiene al mismo usuario administrador que vimos anteriormente por lo que ingresar en el sistema se da de la misma manera.

9 - Anexos

9.1 - API

A continuación tendremos listadas todas las rutas de la API de forma ordenada:

Administrator

Descripción:

Recurso utilizado para representar a la entidad Administrador.

Endpoints:

- POST api/administrators
 - **Descripción:** Ingresa un nuevo administrador al sistema.
 - **Headers:** Authorization
 - Ejemplo de Request Body:

```
1 {  
2   "Id" : "803e6e74-a5ae-4995-8cbb-72646f520621",  
3   "Email" : "admsin@admin.com",  
4   "Name" : "Just Name",  
5   "Password" : "admin"  
6 }
```

- Ejemplo salida:

```
1 {  
2   "id": "803e6e74-a5ae-4995-8cbb-72646f520621",  
3   "name": "Just Name",  
4   "email": "admsin@admin.com"  
5 }
```

- GET api/administrators
 - **Descripción:** Obtiene la lista de administradores del sistema.
 - Ejemplo salida:

```
{  
  "id": "97452d5f-4e20-4c74-9fcd-0474ebeb031c",  
  "name": "Other Name",  
  "email": "otheradmin@admin.com"  
},  
{  
  "id": "803e6e74-a5ae-4995-8cbb-72646f520621",  
  "name": "Just Name",  
  "email": "admsin@admin.com"  
}
```

- GET api/administrators/{idAdministrator}
 - **Descripción:** Obtiene la información de un administrador según su identificador.
 - Ejemplo salida:

```

1 {
2   "id": "97452d5f-4e20-4c74-9fcd-0474ebeb031c",
3   "name": "Other Name",
4   "email": "otheradmin@admin.com"
5 }

```

- PUT api/administrators/{idAdministrator}
 - **Descripción:** Modifica un administrador según el identificador dado.
 - **Headers:** Authorization
 - Ejemplo de Request Body:

```

1 {
2   "Id" : "b27745bc-27fc-4a11-9e9a-f0b1bb676f09",
3   "Email" : "admin@admin.com",
4   "Name" : "Name Changed Name",
5   "Password" : "admin"
6 }

```

- Ejemplo de salida:

```

1 {
2   "id": "b27745bc-27fc-4a11-9e9a-f0b1bb676f09",
3   "name": "Other Common Name",
4   "email": "admintwo@admin.com"
5 }

```

- DELETE api/administrators/{idAdministrator}
 - **Descripción:** Borra el administrador del sistema según el identificador dado.
 - **Headers:** Authorization
 - Ejemplo de salida:

```

1 The Administrator was Deleted: b27745bc-27fc-4a11-9e9a-f0b1bb676f09

```

Request

Descripción:

Recurso utilizado para representar a la entidad Solicitud.

Endpoints:

- POST api/requests
 - **Descripción:** Inserta una nueva solicitud en el sistema.
 - Ejemplo de Request Body:

```

{
  "RequestorsName": "Camila Fernandez",
  "RequestorsEmail": "cfernandez@gmail.com",
  "RequestorsPhone": "123456789",
  "TypeId": "e87309e4-84f4-407c-95e3-08d7f8411bbb",
  "AdditionalFieldValues":
  [
    {
      "AdditionalFieldId": "adf9517c-46ba-4d4e-3871-08d7f8413d87",
      "Value": "03/15/2020"
    },
    {
      "AdditionalFieldId": "6061097d-1cc5-49f2-3872-08d7f8413d87",
      "Value": "SCA1234"
    },
    {
      "AdditionalFieldId": "bb4b631f-07a1-4519-3873-08d7f8413d87",
      "Value": "095937800"
    },
    {
      "AdditionalFieldId": "e7bd40a5-1f82-4340-3874-08d7f8413d87",
      "Value": "Radio Taxi"
    }
  ]
}

```

- GET api/requests
 - **Descripción:** Obtiene todas las solicitudes registradas en el sistema.
- GET api/requests/{idRequest}
 - **Descripción:** Obtiene la información de una solicitud según el identificador dado.
- PUT api/requests/{idRequest}
 - **Descripción:** Modifica una solicitud según el identificador dado.
 - Ejemplo de Request Body:


```

{
  "State": "Aceptada",
  "Description": "New description"
}

```

Area

Descripción:

Recurso utilizado para representar a la entidad Area.

Endpoints:

- POST api/areas

- **Descripción:** Inserta una nueva área en el sistema.

- Ejemplo de Request Body:

```
1 {  
2   "Id": "dcb030a6-00d8-4ea6-a874-e1777af01b57",  
3   "Name": "Area Name",  
4   "Topics": []  
5 }
```

- Ejemplo de salida:

```
1 {  
2   "id": "dcb030a6-00d8-4ea6-a874-e1777af01b57",  
3   "name": "Area Name",  
4   "topics": []  
5 }
```

- GET api/areas

- **Descripción:** Obtiene la lista de áreas registradas en el sistema.

- Ejemplo de salida:

- GET api/areas/{idArea}

- **Descripción:** Obtiene la información de una área según el identificador dado.

- Ejemplo de salida:

```
1 {  
2   "id": "63ce17b5-7ced-49a9-9f11-c573232e7247",  
3   "name": "Other Area Name",  
4   "topics": []  
5 }
```

- PUT api/areas/{idArea}

- **Descripción:** Modifica un área según el identificador dado.

- Ejemplo de Request Body:

```

1 {
2   "id": "63ce17b5-7ced-49a9-9f11-c573232e7247",
3   "name": "Changed Name",
4   "topics": []
5 }

```

- Ejemplo de salida:

```

1 {
2   "id": "63ce17b5-7ced-49a9-9f11-c573232e7247",
3   "name": "Changed Name",
4   "topics": []
5 }

```

- DELETE api/areas/{idArea}

- **Descripción:** Borra un área del sistema según el identificador dado.

- Ejemplo de salida:

```

1 The Area was Deleted: 63ce17b5-7ced-49a9-9f11-c573232e7247

```

Topic

Descripción:

Recurso utilizado para representar a la entidad Tema.

Endpoints:

- POST api/topics

- **Descripción:** Inserta un nuevo tema en el sistema.

- Ejemplo de Request Body:

```

1 {
2   "id": "25064be6-1c27-4dac-b28c-34fc46365333",
3   "areaId": "dcb030a6-00d8-4ea6-a874-e1777af01b57",
4   "name": "Topic Names",
5   "types": []
6 }

```

- Ejemplo de salida:

```

1 {
2   "id": "25064be6-1c27-4dac-b28c-34fc46365333",
3   "areaId": "dcb030a6-00d8-4ea6-a874-e1777af01b57",
4   "name": "Topic Names",
5   "types": []
6 }

```

- GET api/topics

- **Descripción:** Obtiene la lista de temas registrados en el sistema.

- Ejemplo de salida:

```

1  [
2    {
3      "id": "225b3825-3843-4258-973f-0346e36e9c15",
4      "areaId": "d710d7a3-577d-44e0-9d3c-969f78ea4c12",
5      "name": "Taxis, remises, escolares",
6      "types": []
7    },
8    {
9      "id": "2caa669c-8976-4089-98e3-0f7ecec1169e",
10     "areaId": "d710d7a3-577d-44e0-9d3c-969f78ea4c12",
11     "name": "Otros",
12     "types": []
13   },
14 ]

```

- GET api/topics/{idTopic}
 - **Descripción:** Obtiene la información de un tema según el identificador dado.
 - Ejemplo de salida:

```

1  {
2    "id": "225b3825-3843-4258-973f-0346e36e9c15",
3    "areaId": "d710d7a3-577d-44e0-9d3c-969f78ea4c12",
4    "name": "Taxis, remises, escolares",
5    "types": []
6  }

```

- PUT api/topics/{idTopic}
 - **Descripción:** Modifica un tema según el identificador dado.
 - Ejemplo de Request Body:

```

1  {
2    "id": "d7a3c323-8811-4d54-8644-bb1b88cda41c",
3    "areaId": "dcb030a6-00d8-4ea6-a874-e1777af01b57",
4    "name": "Changed Name",
5    "types": []
6  }

```

- Ejemplo de salida:

```

1  {
2    "id": "d7a3c323-8811-4d54-8644-bb1b88cda41c",
3    "areaId": "dcb030a6-00d8-4ea6-a874-e1777af01b57",
4    "name": "Changed Name",
5    "types": []
6  }

```

- DELETE api/topics/{idTopic}
 - **Descripción:** Borra un tema del sistema según el identificador dado.
 - Ejemplo de salida:

```

1  The Topic was Deleted 225b3825-3843-4258-973f-0346e36e9c15

```

Type

Descripción:

Recurso utilizado para representar a la entidad Tipo.

Endpoints:

- POST api/types
 - **Descripción:** Inserta un nuevo tipo en el sistema.
 - Ejemplo de Request Body:

```
1 {  
2   "id": "8fd9650e-b400-4fef-8c43-d44b2f5b255a",  
3   "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",  
4   "name": "Another Type Name",  
5   "additionalFields": []  
6 }
```

- Ejemplo de salida:

```
1 {  
2   "id": "8fd9650e-b400-4fef-8c43-d44b2f5b255a",  
3   "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",  
4   "name": "Another Type Name",  
5   "additionalFields": []  
6 }
```

- GET api/types
 - **Descripción:** Obtiene la lista de tipos registrados en el sistema.
 - Ejemplo de salida:

```
1 [   
2   {  
3     "id": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",  
4     "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",  
5     "name": "Type Name",  
6     "additionalFields": []  
7   },  
8   {  
9     "id": "72e72bf8-7718-419f-ba96-562550e6ae63",  
10    "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",  
11    "name": "Other Type Name",  
12    "additionalFields": []  
13  }  
14 ]
```

- GET api/types/{idType}
 - **Descripción:** Obtiene la información de un tipo según el identificador dado.
 - Ejemplo de salida:

```
1 {  
2   "id": "72e72bf8-7718-419f-ba96-562550e6ae63",  
3   "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",  
4   "name": "Other Type Name",  
5   "additionalFields": []  
6 }
```


- PUT api/topics/{idType}
 - **Descripción:** Modifica un tipo según el identificador dado.
 - Ejemplo de Request Body:

```

1 {
2     "id": "8fd9650e-b400-4fef-8c43-d44b2f5b255a",
3     "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",
4     "name": "Another Changed Name",
5     "additionalFields": []
6 }

```

- Ejemplo de salida:

```

1 {
2     "id": "8fd9650e-b400-4fef-8c43-d44b2f5b255a",
3     "topicId": "25064be6-1c27-4dac-b28c-34fc46365333",
4     "name": "Another Changed Name",
5     "additionalFields": []
6 }

```

- DELETE api/topics/{idType}
 - **Descripción:** Borra un tipo del sistema según el identificador dado.
 - Ejemplo de salida:

```

1 The Type was Deleted: 8fd9650e-b400-4fef-8c43-d44b2f5b255a

```

Additional Field

Descripción:

Recurso utilizado para representar a la entidad Campo Adicional.

Endpoints:

- POST api/additionalfields
 - **Descripción:** Inserta un nuevo campo adicional en el sistema.
 - Ejemplo de Request Body:

```

1 {
2     "TypeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
3     "Name": "Fecha Y Hora",
4     "FieldType": "Fecha",
5     "Ranges": [
6         { "range": "03/01/2020" },
7         { "range": "03/30/2020" }
8     ]
9 }

```

- Ejemplo de salida:


```

1  {
2    "id": "a83e9335-7dda-471f-37e3-08d7f840c7eb",
3    "typeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
4    "name": "Fecha Y Hora",
5    "fieldType": "Fecha",
6    "ranges": [
7      {
8        "id": "764cba7c-9d1b-4c24-ce2d-08d7f840c7ec",
9        "additionalFieldId": "a83e9335-7dda-471f-37e3-08d7f840c7eb",
10       "range": "03/01/2020"
11     },
12     {
13       "id": "7d9cc29f-ea34-4cb3-ce2e-08d7f840c7ec",
14       "additionalFieldId": "a83e9335-7dda-471f-37e3-08d7f840c7eb",
15       "range": "03/30/2020"
16     }
17   ]
18 }

```

- GET api/additionalfields
 - **Descripción:** Obtiene la lista de campos adicionales registrados en el sistema.
 - Ejemplo de salida:

```

1  {
2    {
3      "id": "ba110bd3-4234-4d66-99b4-245177fc9f28",
4      "typeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
5      "name": "Other Additional Name",
6      "fieldType": "Texto",
7      "ranges": []
8    },
9    {
10     "id": "4fff8ce2-dcbf-4874-8900-474271bbdd34",
11     "typeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
12     "name": "Additional Name",
13     "fieldType": "Texto",
14     "ranges": []
15   }
16 }

```

- GET api/additionalfields/{idAdditionalField}
 - **Descripción:** Obtiene un campo adicional registrado en el sistema según el identificador dado.
 - Ejemplo de salida:

```

1  {
2    "id": "a83e9335-7dda-471f-37e3-08d7f840c7eb",
3    "typeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
4    "name": "Fecha Y Hora",
5    "fieldType": "Fecha",
6    "ranges": [
7      {
8        "id": "764cba7c-9d1b-4c24-ce2d-08d7f840c7ec",
9        "additionalFieldId": "a83e9335-7dda-471f-37e3-08d7f840c7eb",
10       "range": "03/01/2020"
11     },
12     {
13       "id": "7d9cc29f-ea34-4cb3-ce2e-08d7f840c7ec",
14       "additionalFieldId": "a83e9335-7dda-471f-37e3-08d7f840c7eb",
15       "range": "03/30/2020"
16     }
17   ]
18 }

```

- PUT api/additionalfields/{idAdditionalField}
 - **Descripción:** Modifica un campo adicional según el identificador dado.
 - Ejemplo de Request Body:

```

1  {
2    "id": "ba110bd3-4234-4d66-99b4-245177fc9f28",
3    "typeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
4    "name": "Changed Other Name",
5    "fieldType": "Texto",
6    "ranges": []
7  }

```

- Ejemplo de salida:

```

1  {
2    "id": "ba110bd3-4234-4d66-99b4-245177fc9f28",
3    "typeId": "7c97bbf7-b731-42b7-97fd-54c5e1962a9c",
4    "name": "Changed Other Name",
5    "fieldType": "Texto",
6    "ranges": []
7  }

```

- DELETE api/additionalfields/{idAdditionalField}
 - **Descripción:** Borra un campo adicional del sistema según el identificador dado.
 - Ejemplo de salida:

```

1  The Additional Field was Deleted: ba110bd3-4234-4d66-99b4-245177fc9f28

```

Login

Descripción:

Recurso relacionado a la sesión y autenticación del rol administrador.

Endpoints:

- POST api/login

- **Descripción:** Genera un nuevo token para el administrador si el mismo no existe, de lo contrario solamente lo retorna en caso de que el administrador sea válido.

- Ejemplo de Request Body:

```
1 {  
2   "Email" : "otheradmin@admin.com",  
3   "Password" : "second"  
4 }
```

- Ejemplo de salida:

```
1 "166f2e74-ccbe-4db0-8fea-ba9f2631952a"
```

Report

Descripción:

Recurso relacionado a los diferentes tipos de reportes.

Endpoints:

- GET api/reports
 - **GetFilteredRequests**
- GET api/reports
 - **GetFilteredTypes**

Parser

Descripción:


Recurso relacionado al parser.

Endpoints:

- GET api/parsers
 - POST api/parsers
-

9.2 - Front-end

Reporte A



ManageReportParserRequestMy RequestLogout

Generate Report A

<

Jun

2020

>

June 2020

July 2020

Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7		1	2	3	4	5	
8	9	10	11	12	13	14	6	7	8	9	10	11	12
15	16	17	18	19	20	21	13	14	15	16	17	18	19
22	23	24	25	26	27	28	20	21	22	23	24	25	26
29	30						27	28	29	30	31		

Email

jhondoe@example.com

Generate

© 2020 Montevideo Request - Github Repository

Figura 9.2.1: Reporte A

Reporte B

Generate Report B

Jun 2020 July 2020

Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7		1	2	3	4	5	
8	9	10	11	12	13	14	6	7	8	9	10	11	12
15	16	17	18	19	20	21	13	14	15	16	17	18	19
22	23	24	25	26	27	28	20	21	22	23	24	25	26
29	30						27	28	29	30	31		

Generate

© 2020 Montevideo Request - Github Repository

Figura 9.2.2: Reporte B

Importación del XML

Parser

Format
XML

FilePath
FilePath

Convert

© 2020 Montevideo Request - Github Repository

Figura 9.2.3: Importación del XML

Ingreso de nueva solicitud

The screenshot shows the 'Request' form in the Montevideo Request system. The form is titled 'Request' and has a progress bar with four steps: 'SELECT AREA', 'SELECT TOPIC', 'SELECT TYPE', and 'ADDITIONAL FIELDS'. The first step, 'SELECT AREA', is currently active. Below the progress bar, there is a dropdown menu labeled 'Area' and a 'Next Step' button. The top navigation bar includes links for 'Manage', 'Report', 'Parser', 'Request' (highlighted), 'My Request', and 'Logout'. The footer contains the copyright notice: '© 2020 Montevideo Request - Github Repository'.

Figura 9.2.4: Ingreso de nueva solicitud

Modificación de un administrador

The screenshot shows the 'Administrator edition' modal form in the Montevideo Request system. The modal is titled 'Administrator edition' and has a close button (X). It contains two input fields: 'Name' with the value 'Administrator' and 'Email' with the value 'main@admin.com'. At the bottom of the modal, there are 'Close' and 'Save' buttons. The background shows the 'Administrators' section of the system, with a search bar and a list of administrators. The top navigation bar includes links for 'Parser', 'Request' (highlighted), 'My Request', and 'Logout'. The footer contains the copyright notice: '© 2020 Montevideo Request - Github Repository'.

Figura 9.2.5: Modificación de un administrador

Alta de tipo de solicitud con campos adicionales

SELECT AREA SELECT TOPIC SELECT TYPE ADDITIONAL FIELDS

Almost There

Additional Fields

Description:

Contact Information

Name:

Jhon Doe

Email:

enter@email.com

Phone:

Go Back Submit

Figura 9.2.6: Alta de tipo de solicitud con campos adicionales

Listado de solicitudes

Montevideo

Manage Report Parser Request My Request Logout

Requests

Search By Email

# ID	State	Requestors Email	Requestors Name	Type	Description	Action
afee66ca-a168-49d2-00a9-08d81953d214	Creada	test@test.com	test	ENTREGA FINAL		
5177b3f3-fdcd-476b-00aa-08d81953d214	Creada	test@test.com	test	ENTREGA FINAL		
08efcb73-513a-4fde-00ab-08d81953d214	Creada	test@test.com	test	ENTREGA FINAL		
d88ce71e-46eb-47d8-00ac-08d81953d214	Creada	cfernandez@gmail.com	Camila Fernandez	ENTREGA FINAL		
adb3a802-e800-481a-00ad-08d81953d214	Creada	cfernandez@gmail.com	Camila Fernandez	ENTREGA FINAL		
276fc157-4068-460b-00ae-08d81953d214	Creada	cfernandez@gmail.com	Camila Fernandez	ENTREGA FINAL		
0083c9b6-a289-448b-00af-08d81953d214	Creada	cfernandez@gmail.com	Camila Fernandez	ENTREGA FINAL		
...

Figura 9.2.7: Listado de solicitudes

Cambio de estado de una solicitud

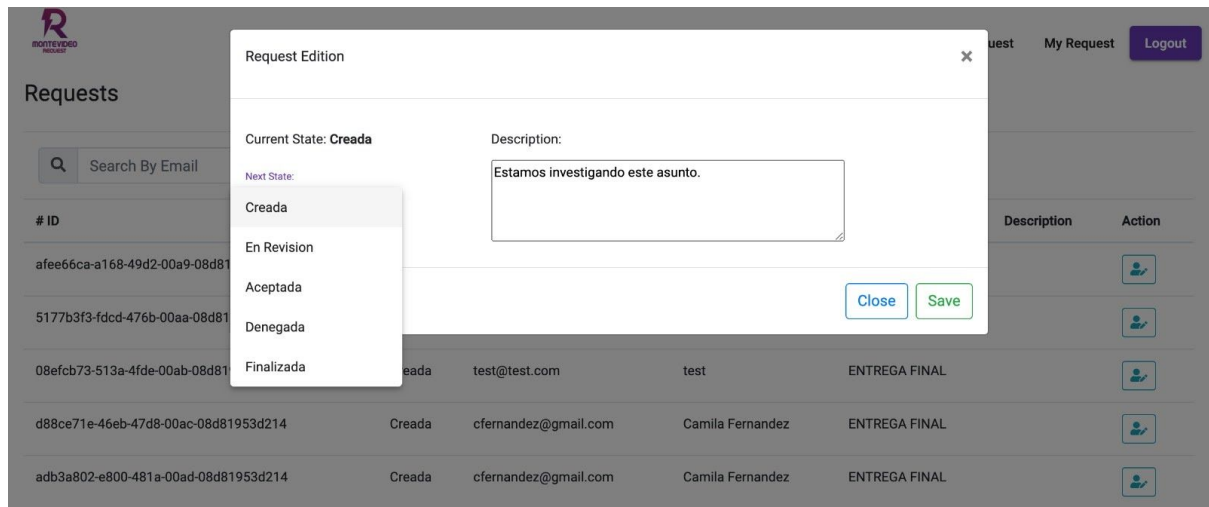


Figura 9.2.8: Cambio de estado de una solicitud

10 - Bibliografía

1. R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2008.
2. R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2017.
3. R. C. Martin. Clean coder blog - Principios a nivel de paquetes.
<http://blog.cleancoder.com/uncle-bob/2016/01/04/ALittleArchitecture.html>
4. Open Closed Principle. https://en.wikipedia.org/wiki/Open-closed_principle
5. Liskov Substitution Principle. https://en.wikipedia.org/wiki/Liskov_substitution_principle
6. Interface Segregation Principle. https://en.wikipedia.org/wiki/Interface_segregation_principle

7. Wikipedia. Single Responsibility Principle. https://en.wikipedia.org/wiki/Single_responsibility_principle
8. Wikipedia. Dependency Inversion Principle. https://en.wikipedia.org/wiki/Dependency_inversion_principle
9. Wikipedia. Single Page Application. [https://en.wikipedia.org/wiki/Single-page_application#Challenges with the SPA model](https://en.wikipedia.org/wiki/Single-page_application#Challenges_with_the_SPA_model)
10. 4+1 architectural view model. [https://en.wikipedia.org/wiki/4%2B1_architectural view model](https://en.wikipedia.org/wiki/4%2B1_architectural_view_model)
11. Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. <http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf>