

Universidad ORT Uruguay

Facultad de Ingeniería

Diseño de aplicaciones II

Obligatorio II

Descripción del diseño

Juan Manuel Gallicchio 233335

Federico Carbonell 224359

<https://github.com/ORT-DA2/CarbonellGallicchio>

Índice

Descripción General	3
Bugs o Defectos conocidos:	4
Descripción y Justificación de Diseño - Principios y patrones	4
Vista Lógica	4
Vista de Proceso	9
Vista de Desarrollo	12
Vista de Despliegue	13
Modelo de Base de Datos	13
Mecanismos de extensibilidad	14
Calidad de diseño	15
Cohesión relacional	15
Estabilidad/Inestabilidad	16
Abstracción	17
Mejoras al diseño	23
Descripción de jerarquías de herencia utilizadas	25
Anexo	26
Decisiones Generales	26
Diagrama de clases vista lógica	27
Descripciones de la API	31
Análisis de cobertura de pruebas	42
Importar archivos desde la web	44
Cohesión relacional	45
Estabilidad/Inestabilidad	46
Abstracción	47
Link a repositorio	48
Super-Admin	48
Urls de video contents	48

Descripción General

Se entrega el sistema BetterCalm compuesto por el BackEnd y un FrontEnd, la finalidad de esta sería ayudar a las personas a sobrellevar el estrés de una manera más sana.

Está compuesto por dos grandes funcionalidades, un reproductor de contenido reproducible y un sistema de agendamiento de consultas con psicólogos. El desarrollo se llevó a cabo utilizando framework .Net Core para la Api y Angular 11 para el front end.

La API se realizó cumpliendo el estándar REST. Este estándar aprovecha los headers HTTP para darle un nombre a las consultas. Esta API a su vez no posee lógica y su función principal es atender peticiones y devolver una respuesta. Los verbos principales y utilizados de REST son GET, POST, PUT y DELETE. A pesar de ser solo 4, mediante la especificación de distintas URIs la API logra tener más de 4 métodos por controller.

Los retornos de la API pueden contener un mensaje de error u éxito pero su parte principal que indica que sucedió con la consulta es el Status Code. Estos errores se clasifican en cinco “familias”:

- 100 – Informational
- 200 – Success
- 300 – Redirections
- 400 – Client Errors
- 500 – Server Errors

Para el envío de entidades en el body de una consulta se utiliza JSON.

Para el acceso a datos utilizamos el ORM Entity Framework Core realizando la técnica de Code First.

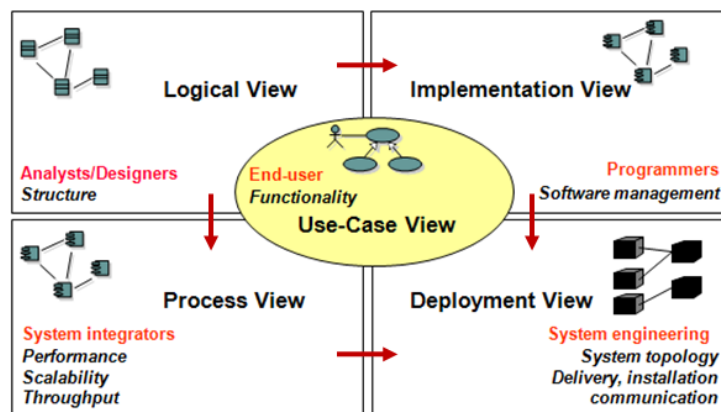
La aplicación fue realizada utilizando la metodología TDD, implica primero codificar las pruebas, luego implementar las funciones y por último realizar un refactor de las mismas. Por lo que la aplicación presenta sus proyectos de pruebas. Estos son MSTest.

Bugs o Defectos conocidos:

A partir de que el nivel de cobertura de pruebas del backend es altamente satisfactorio y luego de realizar pruebas exhaustivas podemos decir que no hemos localizado bugs o defectos, esto no quiere decir que no existan sino que no hemos encontrado.

Descripción y Justificación de Diseño - Principios y patrones

Seguiremos las vistas del Modelo 4+1 para describir y justificar nuestro diseño. Este modelo sirve para describir la arquitectura de un sistema de software basado en varias vistas concurrentes. El siguiente diagrama describe al modelo 4+1. A medida que vayamos explicando las diferentes vistas iremos comentando los principios utilizados y el por qué.

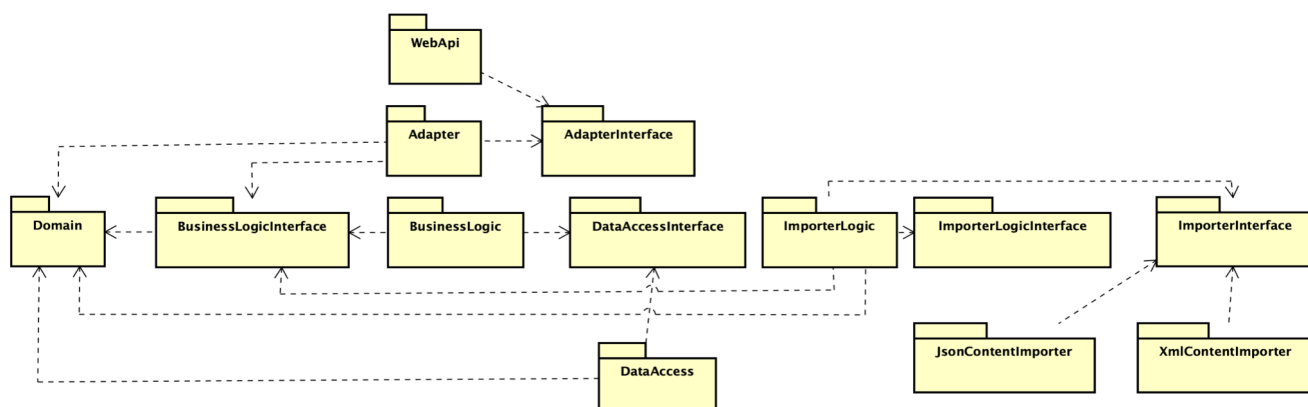


Vista Lógica

Esta vista describe el modelo desde los elementos de diseño que componen el sistema y también cómo interactúan entre ellos.

El siguiente diagrama ilustra la división de capas y las dependencias de nuestro proyecto, decidimos omitir paquetes que no son relevantes para esta instancia como el paquete de test , validaciones, sesión y excepciones.

Se puede apreciar que el proyecto está dividido en cuatro capas distintas, no se realizan atajos entre capas. Los paquetes JsonContentImporter y XmlContentImporter están a la misma altura que ImporterInterface pero para visualizarlo correctamente lo posicionamos debajo.



Es decir, en ningún momento la capa de Web Api interactúa directamente con la Business Logic ni viceversa.

Paquete	Responsabilidad
WebApi	Posee nuestra Web Api. Su responsabilidad es recibir las consultas del cliente, esto se realiza a través de controllers y luego debe derivarlas a la lógica de negocio correspondiente. La entrada y salida de datos se realizan a través de modelos.
AdapterInterface	Poseen las interfaces de nuestros adapters, estos se encargan de mapear los modelos a objetos del dominio entre los controllers y la business logic.
Adapter	Posee las implementaciones de las clases del paquete AdapterInterface.
BusinessLogicInterface	Poseen las interfaces de nuestra lógica de negocio. Estas interfaces son utilizadas por las clases del paquete Adapter.
BusinessLogic	Poseen las implementaciones de la clase del paquete BusinessLogicInterface. Utilizan la interfaz IRepository para interactuar con la base de datos.
Domain	Posee nuestras entidades. Estas mismas persisten en la base de datos.
DataAccessInterface	Posee la interfaz de los métodos relacionados al acceso a datos.
DataAccess	Posee la implementación de la clase del paquete DataAccessInterface. También contiene el contexto de nuestra base de datos.

ImporterLogicInterface	Posee la interfaz de los métodos relacionados a la importación de contenido reproducible.
ImporterLogic	Posee las implementaciones de las clases del paquete ImporterLogicInterface.
ImporterInterface	Posee la interfaz de los metodos relacionados a la importacion de contenido reproducible que es provista a un tercero si quisiera extender los mecanismos
JsonContentImporter	Posee la implementación de un importador de tipo Json, este implementa la interface ImporterInterface
XmlContentImporter	Posee la implementación de un importador de tipo Xml, este implementa la interface ImporterInterface

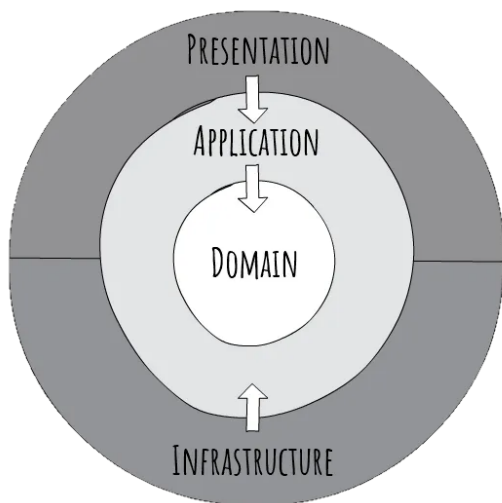
Como se puede apreciar en el diagrama ambas capas de bajo nivel (Web Api y Data Access) dependen de capas de alto nivel (Data Access Interface y Adapter Interface). Esto es sumamente importante y reutilizable ya que las componentes de bajo nivel están más dispuestos a los cambios, por lo que sí dependen de capas de alto nivel (menor probabilidad de cambio) nos genera estabilidad evitando tener que modificar gran parte de la aplicación. También nos genera bajo acoplamiento ya que no las clases no presentan grandes dependencias, si el día de mañana se desea cambiar la implementación de la Web Api o de Data Access solamente deberíamos integrarlo a la lógica ya hecha. Para decidir el diseño base de la aplicación nos basamos en el principio SOLID de inversión de dependencias y en bajo acoplamiento de GRASP.

Se desprende también del diagrama que se cumple el principio equivalencia de reuso/liberación (REP) este indica que todo lo que se usa junto, se libera junto. Lo que buscamos con esto es que si el día de mañana se modifica una clase de un paquete no es necesario recompilar todo el sistema sino que solamente el paquete contenedor de esta.

A partir del siguiente análisis se puede observar cómo se cumple con Clean Architecture (Robert C. Martin). Este concepto plantea ciertas reglas:

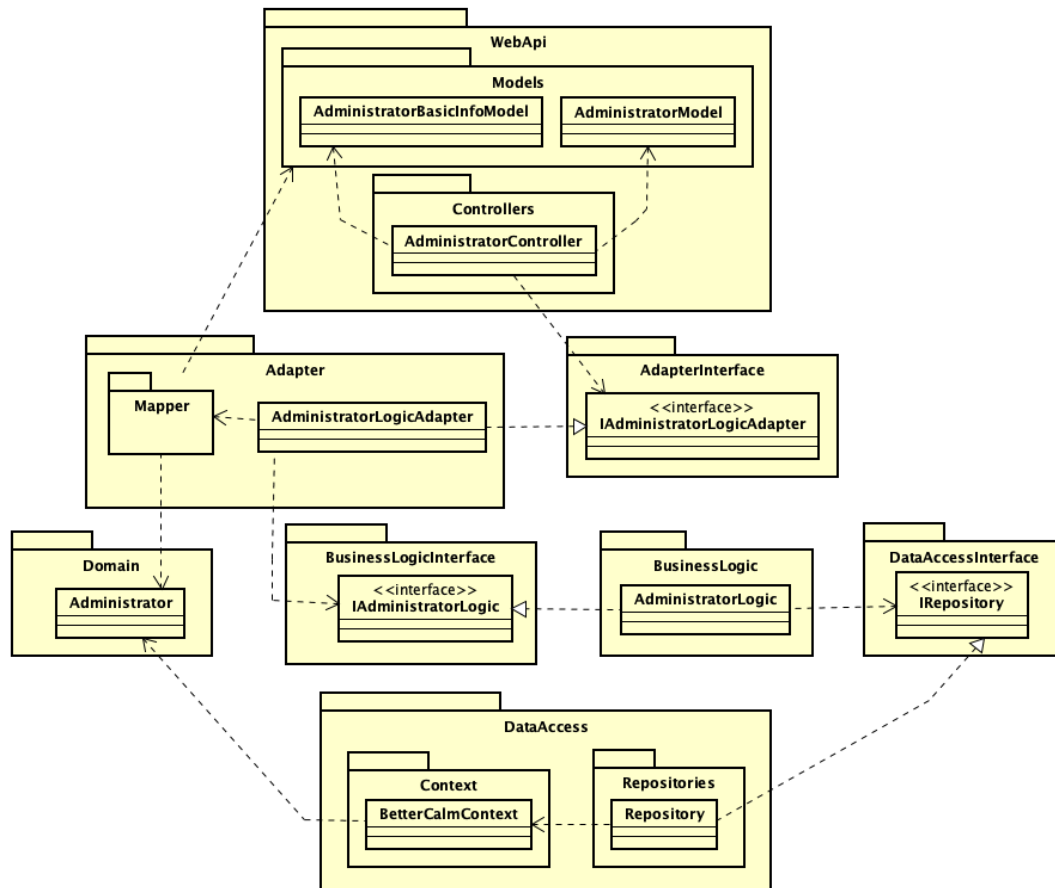
- La aplicación no debe depender de la interfaz de usuario
- La aplicación no debe depender a la base de datos
- Todas las capas deben poder probarse de forma independiente
- No se debe depender de frameworks específicos

En conclusión lo que nos indica las reglas es que las dependencias deben ser hacia adentro, es decir a capas que sean menos propensas al cambio. No se puede depender de capas externas ya que si se desea modificar el motor de base de datos esto no nos genera un gran impacto en el sistema.



Si observamos nuestro diagrama de capas se puede notar que todas nuestras dependencias van hacia el centro. Es cierto que dependemos del dominio, pero este posee una baja probabilidad de cambio en comparación con el resto por lo que es una buena práctica.

Para mejorar el entendimiento y la visibilidad decidimos realizar el siguiente diagrama para ilustrar las dependencias de la clase Administrator. Decidimos representarlo de esta forma ya que las dependencias entre los controllers, business logic, models y domain son iguales para todas las entidades.



Siguiendo el principio Single Responsibility de SOLID el equipo decidió separar las responsabilidades en distintos paquetes, a su vez, dentro de estos se decidió separarlos a partir de cada entidad de dominio, por ejemplo podemos observar que con este fin se creó una clase de BusinessLogic por entidad, lo mismo aplica en los controllers de WebApi y en el paquete Adapter. Esto nos garantiza que solamente se realicen acciones sobre cada entidad mediante su propia lógica, ésta será modificada solamente si la entidad es afectada por cambios. Además de esto, cada lógica implementa una interfaz que expone las funciones requeridas. Con esto estamos invirtiendo las dependencias y además segregando las interfaces ya que se implementa lo mínimo indispensable para su uso.

También hemos separado la lógica de negocio de la capa de presentación, esto aumenta la reutilización de código y nos genera mayor control. Para diseñar esta parte nos basamos en el principio Controlador de GRASP, que plantea esto mismo, con el fin de lograr los objetivos mencionados.

También podemos observar cómo se cumple el principio de clausura común, ya que todas las clases que cambian juntas, permanecen juntas. Por esta razón se encuentra separado por entidades, para evitar cambios en múltiples paquetes. Por este motivo, por ejemplo, el día de mañana el equipo puede separar únicamente la capa lógica del resto de la solución, a su vez también podemos realizar cambios sólo en una capa sin afectar las demás, favoreciendo un menor grado de cambio a la hora de re compilar los proyectos necesarios.

Esto igualmente es complementado con la inyección de dependencias. Este concepto se basa en un contenedor en el cual se asocian interfaces con su implementación correspondiente. Una vez declarada esta asociación en el contenedor, se puede realizar la inyección a través del constructor de cada clase. De esta forma, como la creación concreta de la implementación de la interfaz no se realiza dentro de la clase que lo necesita, no se obtiene la dependencia entre la clase. La inyección de dependencias se utilizó para todas las capas de la solución a nivel de backend.

Además, con el fin de facilitar el reuso y la mantenibilidad, el equipo decidió implementar el Repositorio de acceso a datos y la validación de los distintos objetos de la solución de forma genérica y siguiendo con el principio de inversión de dependencias anteriormente mencionado. Esta decisión fue tomada luego de empezar a codificar y visualizar la repetición lógica de la gran mayoría de los métodos de acceso a datos y de validación, gracias a esto logramos disminuir la cantidad de líneas de código y pruebas. Por lo que también obtuvimos una mejora en performance y mantenibilidad.

Los restantes diagramas de clase de la vista lógica se encuentran en el anexo (VER ANEXO, sección “Diagrama de clases vista lógica”).

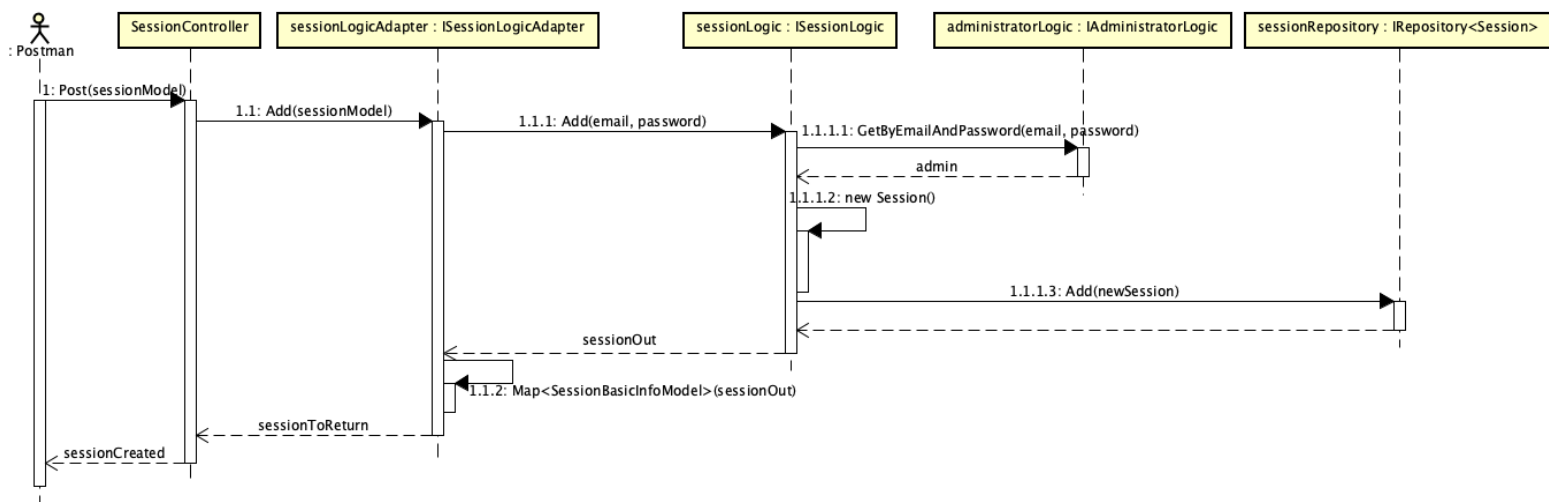
Vista de Proceso

Aquí mostraremos el comportamiento de algunas funciones de nuestro sistema en tiempo de ejecución. Decidimos realizar de la obtención de un token de administración y la lógica de la agenda de una consulta con un psicólogo ya que están compuestas por la lógica más abstracta, las demás funciones leyendo el código se logran entender sencillamente. El

propósito de este documento es darle al lector una ayuda para entender el funcionamiento general del sistema, por esto decidimos no abrumarlo con diagramas.

Obtención de token

Para la creación de un token lo primero que verificamos es que los datos enviados por el cliente son válidos, luego de verificarlos el adapter se encarga de realizar el mapeo al tipo de datos utilizado por la lógica de negocio. Ya en la lógica de negocio se obtiene el administrador con los datos proporcionados por el adapter. Con este y un nuevo token se crea una instancia de sesión que es enviada al repositorio. De ser satisfactoria la creación se le devuelve al cliente el correo proporcionado y el token de privilegio administrador.



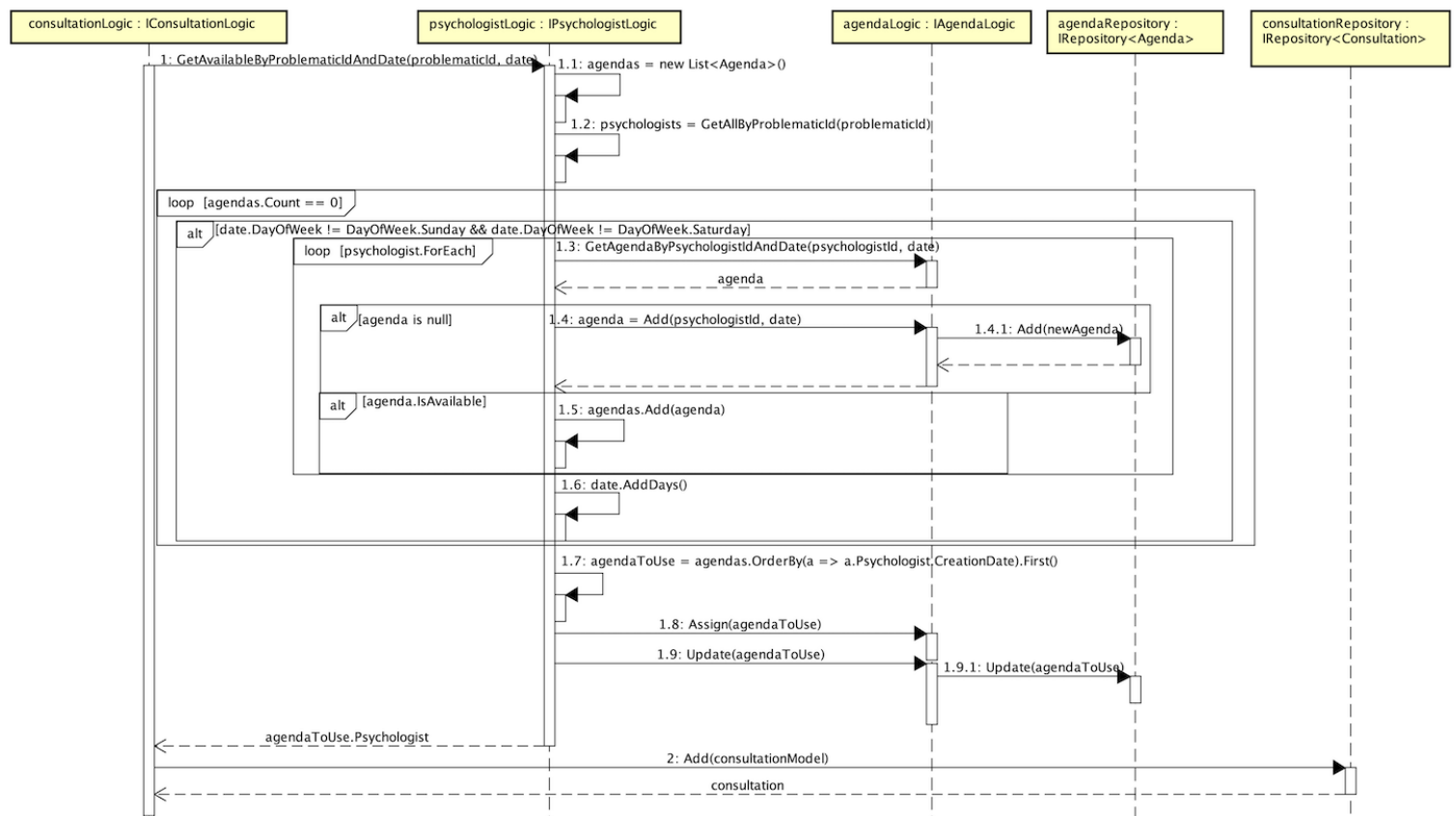
Lógica de agendamiento de consulta con psicólogo

No incluimos la ejecución desde el controller a la lógica ya que su finalidad es realizar los mapeos correspondientes y no aportaba información relevante. La finalidad del siguiente diagrama es aclarar la lógica que se realiza a la hora de agendar una consulta.

Inicialmente se ejecuta el método de la lógica de negocios de psicólogo, este creará una lista de tipo agenda que se compone por un Psicólogo, una fecha, la cantidad de consultas y una variable que indica si el psicólogo se encuentra disponible. Luego obtendrá todos los psicólogos que son expertos en la problemática indicada en la consulta.

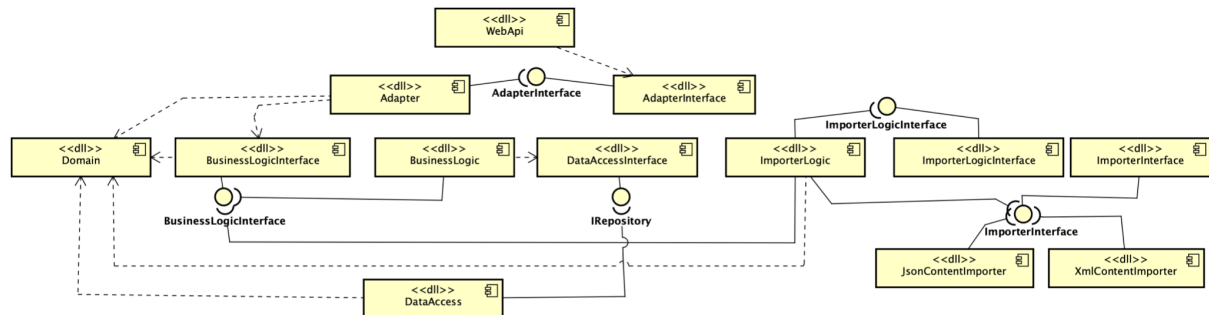
Al tener las dos listas creadas se ejecutará un while recorriendo la agenda a partir de la fecha hasta encontrar un psicólogo que sea experto en la problemática indicada y tenga menos de cinco consultas al día.

Al finalizar la iteración se ordenará la lista obtenida para obtener el psicólogo más antiguo en el sistema, se asignará en la agenda y se modificará en el repositorio. Si el proceso fue satisfactorio se le retornará al usuario el psicólogo que lo atenderá en la consulta.



Vista de Desarrollo

Aquí se mostrará el diagrama de los componentes de nuestro proyecto. Estos son los elementos físicos que se manifiestan en tiempo de ejecución.



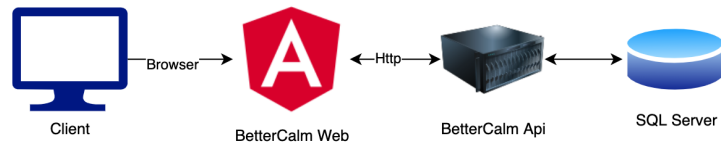
Aquí se puede observar como el componente AdapterInterface provee interfaces que son requeridas por Adapter, asimismo BusinessLogicInterface provee interfaces que son requeridas por BusinessLogic y DataAccessInterface provee la interfaz del repositorio a DataAccess.

Para esta segunda entrega agregamos cinco componentes relacionados a la importación de contenido reproducible. ImporterLogicInterface provee interfaces que son requeridas por ImporterLogic y a su vez ImporterInterface provee una interfaz utilizada por ImporterLogic y los importadores específicos para extender el sistema.

Se puede observar a simple vista como ningún componente depende de una implementación, sino que dependen de interfaces. Aquí se vuelve a notar el principio de inversión de dependencias.

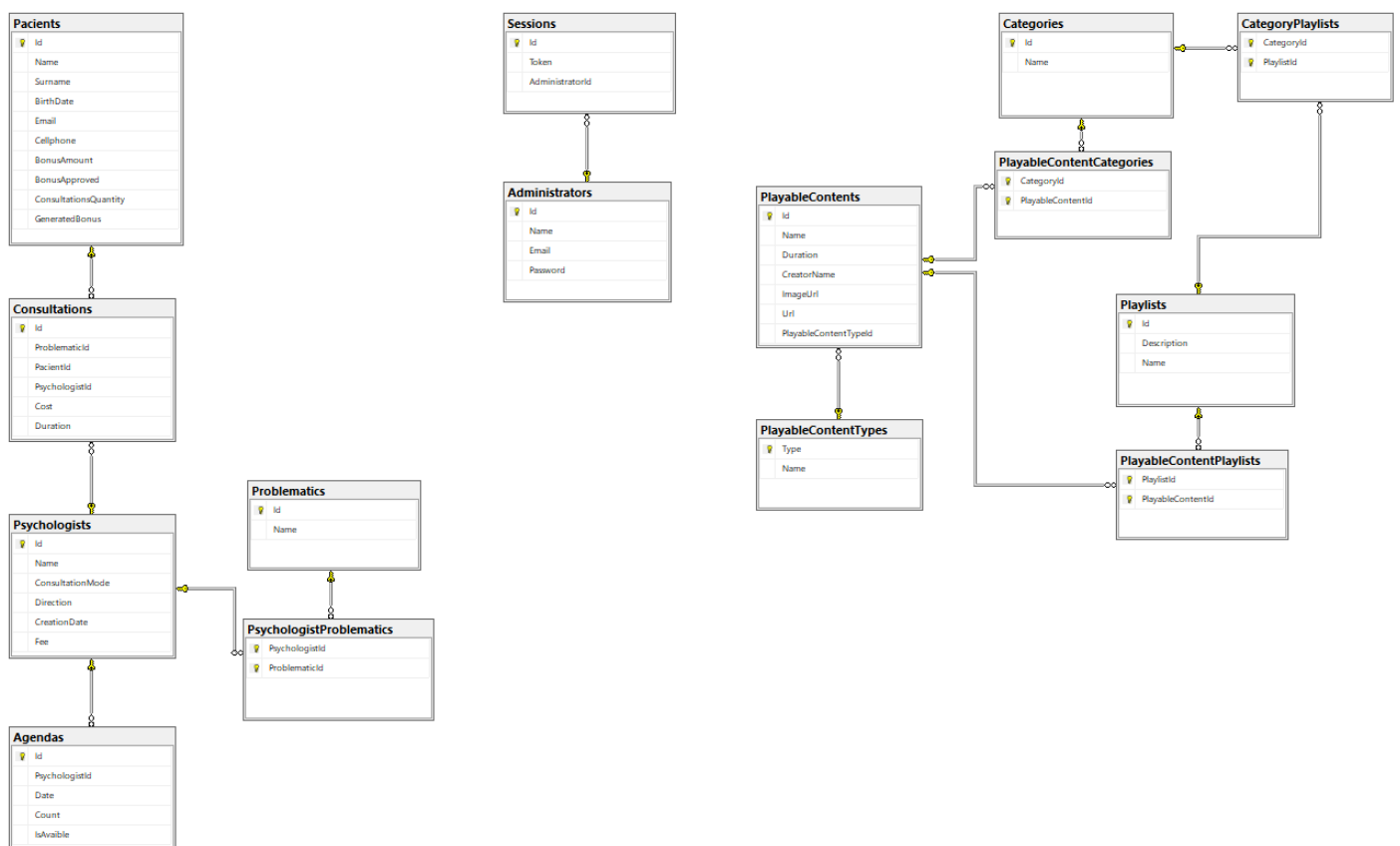
Vista de Despliegue

Aquí mostraremos los artefactos de nuestra aplicación en su ambiente de ejecución.



A simple vista se puede notar que la potencia de nuestro hardware no deberá ser muy grande por el momento.

Modelo de Base de Datos



Mecanismos de extensibilidad

El objetivo de esta sección es describir los mecanismos posibles para que usuarios externos a la aplicación puedan hacer uso de la importación de contenido reproducible mediante archivos desde cualquier tipo de fuente que se desee implementar, y que también sirva como información para la futura implementación.

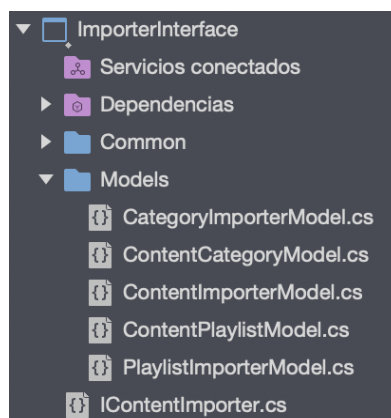
Para la importación de contenido utilizando implementaciones externas nuestro sistema utiliza reflection para autoexaminarse y encontrar ensamblados mediante el método ***Assembly.LoadFile(dll)***. Al usar esto podemos crear la instancia mediante ***Activator.CreateInstance(type)*** y luego crear objetos e invocar sus métodos en tiempo de ejecución sin haber tenido que realizar una referencia al ensamblado que contiene la clase o su namespace por ejemplo ***contentImporter.ImportContent(filePath)***.

Nuestro sistema expone la interfaz ***IContentImporter*** que contiene los siguientes métodos:

```
using ImporterInterface.Models;

namespace ImporterInterface
{
    public interface IContentImporter
    {
        ContentImporterModel ImportContent(string filePath);
        string GetId();
    }
}
```

Si se desea generar un nuevo tipo de formato de importación se deben implementar los métodos detallados previamente cumpliendo con los tipos indicados. Además de exponer dicha interfaz también se proveen los modelos necesarios para lograr implementarlo de una manera exitosa.



A manera de ejemplo mostraremos un importador de contenido mediante un archivo Json implementado por el equipo.

```
public class JsonContentImporter : IContentImporter
{
    public string GetId()
    {
        return "json";
    }

    public ContentImporterModel ImportContent(string filePath)
    {
        string file = File.ReadAllText(filePath);
        var serializerOptions = new JsonSerializerOptions
        {
            Converters = { new TimeSpanConverter() },
            PropertyNameCaseInsensitive = true
        };
        ContentImporterModel exampleJson = JsonSerializer.Deserialize<ContentImporterModel>(file, serializerOptions);
        return exampleJson;
    }
}
```

Como se puede apreciar esta clase implementa IContentImporter y todos sus métodos siguiendo los tipos definidos. Luego de realizar lo anterior es posible generar un request a api/importers y realizar la importación de un contenido reproducible ya sea audio o video mediante un archivo Json. La documentación sobre cómo realizar el request se encuentra en el anexo.

Calidad de diseño

Se analizará la calidad del diseño en base a distintas métricas de diseño y contrastandolas respecto a la aplicación de principios. Para obtener las métricas que se presentarán se utilizó el software Ndepend, el cual retorna un profundo análisis de nuestro código, nosotros decidimos utilizar solo algunas de las métricas que obtuvimos con el objetivo de mostrar la calidad de nuestro diseño. Las imágenes con los resultados detallados obtenidos con dicha herramienta se pueden encontrar en el anexo.

Cohesión relacional

Esta métrica mide la relación entre clases de un paquete. El valor asociado a esta métrica lo vemos en la última columna de esta tabla (VER ANEXO, sección “Cohesión relacional”). El cálculo de la cohesión relacional se hace de la siguiente manera, $H = (R + 1)/N$. R se refiere al número de relaciones internas del paquete y N se refiere al número de clases de paquetes.

Como las clases dentro de un assembly o paquete deben estar fuertemente relacionadas, la cohesión debe ser alta. Sin embargo, los valores demasiado altos pueden indicar un acoplamiento excesivo. Se puede decir que buen rango para Cohesión relacional es de 1,5 a 4,0.

En este caso puede llamar la atención el valor de la business logic, cuyo valor es de 0.2. Esto se da porque fue necesario crear distintas clases encargadas de la lógica y ellas tienen relación con distintos repositorios directamente. Esto sería incorrecto en caso de que el equipo duplique código o mismas lógicas en distintas clases, esto no sucede en nuestro caso. Por lo tanto, no nos preocupa este valor.

Otros paquetes, como los de excepciones, los de interfaces o los validadores, tiene sentido que tengan un valor bajo, ya que no debe haber relaciones entre las clases del mismo paquete. Por lo tanto, en estos paquetes hay varias clases pero con pocas relaciones entre sí.

Estabilidad/Inestabilidad

Esta métrica se calcula para saber el esfuerzo necesario para cambiar un paquete. Su cálculo es de la siguiente manera: $I = \text{Acoplamiento eferente} / (\text{Acoplamiento eferente} + \text{Acoplamiento aferente})$. Acoplamiento eferente = Dependencias que “salen” del paquete. Acoplamiento aferente = Dependencias que “entran” al paquete. Su resultado varía entre 0 y 1. Cuanto más cercano sea a 1 entonces el paquete es más inestable, ya que, existen más dependencias salientes que entrantes. Por el contrario, cuanto más cercano a 0 menos inestable por lo que existen más dependencias entrantes que salientes.

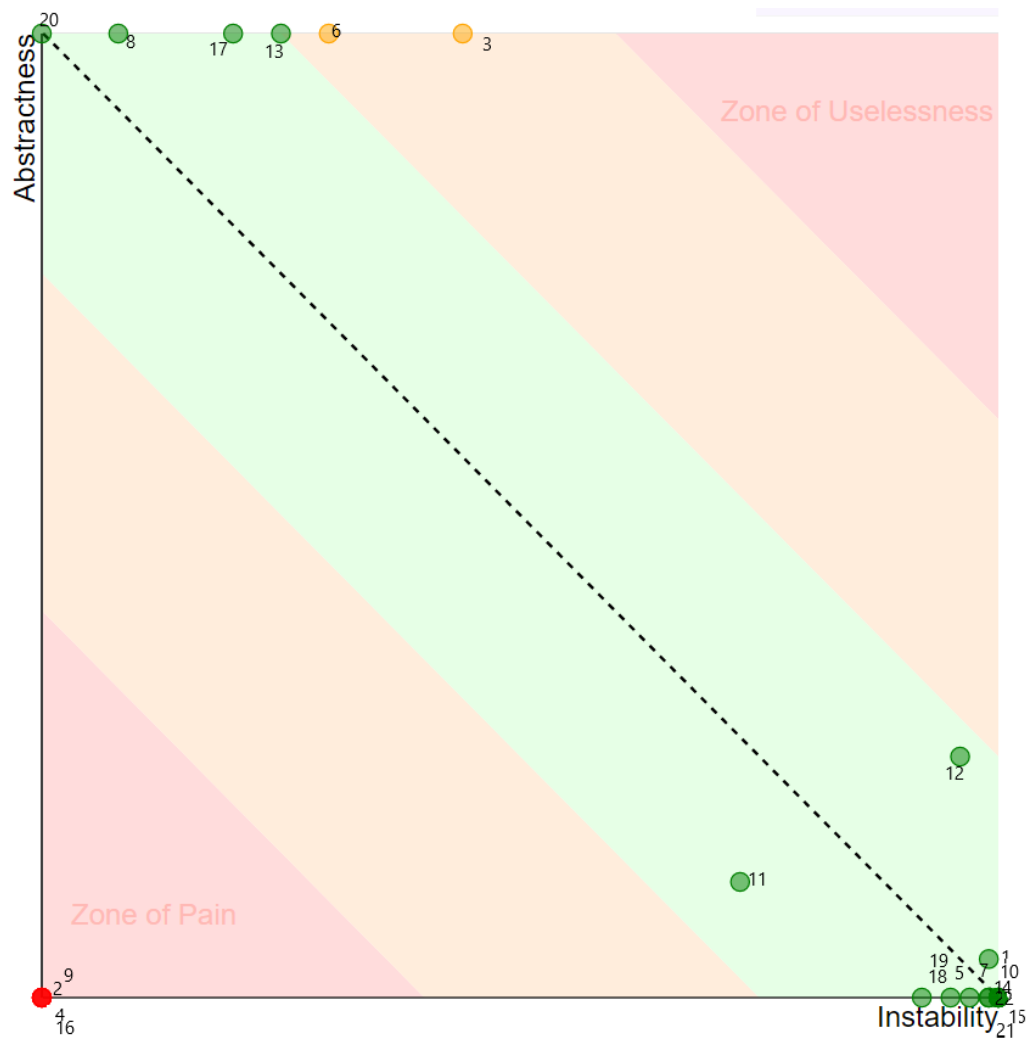
Para ver los valores detallados de esta métrica, se puede consultar el anexo en la sección “Estabilidad/Inestabilidad”. En los valores obtenidos para esta métrica, vemos que el paquete BusinessLogic tiene muy buenos números, al contrario que en la métrica anterior, este es bastante inestable, esto es bueno en este caso, ya que los cambios en la lógica del negocio no afectarán al resto de los paquetes. Lo mismo sucede para WebApi, Adapter, DataAccess y SessionLogic, esto nos parece bastante importante, ya que son los paquetes claves en nuestro proyecto y esta métrica nos da indicios de un diseño de calidad, ya que los componentes son reusables. Esto se dió ya que el equipo a la hora de diseñar la solución

siguió el principio de reuso común, por lo que clases que no se resusan juntas no pertenecen juntas. Esto lo cumplimos y se ve con estas métricas porque podemos reutilizar clases por sí mismas y no afectan a otros paquetes.

Abstracción

Esta métrica es la relación entre el número de clases y clases abstractas de un paquete. Sirve para saber qué tan abstracto es un paquete. Cuanto más estable es un paquete, más abstracto será y por el contrario, cuanto más inestable sea un paquete menos abstracto será. El cálculo de la métrica se forma de la siguiente manera: $A = N_a/N_c$, siendo N_a el número de interfaces y clases abstractas en el paquete y siendo N_c el número de interfaces, clases abstractas y clases concretas. A varía entre 0 y 1, cuanto más cerca esté A de 1 más abstracta es y si se acerca a 0 la clase es más definida. Los valores obtenidos por esta métrica se pueden encontrar en el anexo en la sección “Abstracción”.

Para analizar esta métrica, al equipo le pareció interesante hacer un análisis en formato tabla a partir de la siguiente gráfica que indica la relación entre la métrica de inestabilidad y la métrica de abstracción de los paquetes. La gráfica sugiere que los paquetes se encuentren en lo posible en la zona verde en la línea punteada. Luego está la zona naranja que sugiere que puede llegar a ser problemático tener un paquete ahí pero no es un problema que algún paquete se encuentre allí. Por último, la zona roja que es el conjunto de la “zone of pain” y “zone of uselessness”. No es bueno tener paquetes en esa zona por un lado porque pueden ser un gran dolor de cabeza modificarlos (en la “zone of pain”) y por otro pueden significar que son potencialmente “no utilizables” (en la “zone of uselessness”).



Tabla

Índice	Nombre	Resultado			Análisis
		Inestability (I)	Abstractness (A)	Distance (D)	
1	Adapter	0.99	0.04	0.02	<p>Las implementaciones de las interfaces que se encuentran en este paquete son encargadas de conocer tanto la web api como la lógica y hacer de “pasamanos” entre ellas.</p> <p>Por lo tanto, tiene sentido haber obtenido:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Muy inestable (I). - Nada extensible (D). <p>Todos los controllers de la webApi dependen de esta capa, y esta capa depende de la business logic. Por eso los valores de las métricas tienen sentido. Sin embargo, por cómo son estas dependencias (a través de interfaces) es que no creemos que la inestabilidad que se refleja aquí sea una preocupación.</p> <p>Es cierto que quizás sea poco extensible ya que un cambio aquí podría llevarnos a re codificar y re compilar la api. Por eso se obtiene el valor bajo en extensibilidad, pero nos parece esperable ya que es la implementación concreta de la interfaz correspondiente.</p>
2	AdapterExceptions	0	0	0.71	<p>En este paquete se encuentran las excepciones que conoce el adapter y la web api. Todo es concreto, por eso no se espera que sea abstracto.</p> <p>Tiene sentido haber obtenido:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Muy estable (I). - Bastante extensible (D). <p>Aunque se encuentra en la “zona de dolor” el objetivo de este paquete es ser funcional en esta solución y desacoplar a las demás capas. Por lo tanto, cumple su función de funcionar concretamente en nuestra solución y favorecer a SRP y al principio de reuso común ya que agrupa únicamente a las excepciones con el objetivo antes mencionado.</p>
3	AdapterInterface	0.44	1	0.31	<p>Este paquete tiene como objetivo reunir todas las interfaces necesarias para que el paquete Adapter funcione. Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Totalmente abstracto (A). - Bastante estable (I). - Difícil de extender (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que codificar y volver a compilar otros paquetes donde se realizan dichas interfaces, en este caso en mayormente en WebApi. Esta característica de extensibilidad se ve representado en el valor de D.</p>
4	BusinessExceptions	0	0	0.71	<p>En este paquete se encuentran las excepciones que conoce el adapter y la lógica de negocio. Todo es concreto, por eso no se espera que sea abstracto.</p> <p>Tiene sentido haber obtenido:</p> <ul style="list-style-type: none"> - Nada abstracto (A).

					<ul style="list-style-type: none"> - Muy estable (I). - Bastante extensible (D). <p>Aunque se encuentra en la “zona de dolor” el objetivo de este paquete es ser funcional en esta solución y desacoplar a las demás capas. Por lo tanto, cumple su función de funcionar concretamente en nuestra solución y favorecer a SRP y al principio de reuso común ya que agrupa únicamente a las excepciones con el objetivo antes mencionado.</p>
5	BusinessLogic	0.97	0	0.02	<p>En este paquete se implementan las interfaces de la lógica del negocio, por eso no se espera que sea abstracto.</p> <p>Tiene sentido haber obtenido:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Muy inestable (I). - Poco extensible (D). <p>Depende demasiadas veces del Repositorio genérico, por eso es tan inestable. Se hace poco extensible ya que el adapter depende muchas veces de la interfaz que aquí se implementa, sin embargo, esto lo hace muy bueno para el mantenimiento y la resolución de bugs, porque un refactor no afecta para nada otras capas, además como aquí se encuentra la lógica del negocio este diseño nos permite que casi siempre los bugs se encuentren en esta capa. Es muy bueno poder cambiarla “sin miedo”.</p> <p>Aunque en la gráfica se encuentra en la zona verde, de reuso, no creemos que los valores de la métrica reflejen del todo las ventajas que obtiene el paquete en consecuencia del diseño presentado.</p>
6	BusinessLogicInterface	0.3	1	0.21	<p>Este paquete tiene como objetivo reunir todas las interfaces necesarias para que el paquete BusinessLogic funcione, el mismo contiene toda la lógica del negocio. Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Totalmente abstracto (A). - Muy estable (I). - Difícil de extender (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que codificar y volver a compilar otros paquetes donde se realizan dichas interfaces, en este caso en mayormente en el Adapter. Esta característica de extensibilidad se ve representado en el valor de D.</p>
7	DataAccess	0.99	0	0.01	<p>En este paquete se encuentra la configuración del contexto así como la implementación del repositorio genérico.</p> <p>Este paquete es:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Altamente Inestable (I). - Bastante difícil de cambiar(D) <p>Este proyecto es muy inestable ya que aquí se persiste el dominio (Domain) y aunque estas clases son concretas y estables, cualquier cambio allí impacta aquí directamente porque un cambio en una de ellas impactará la configuración y/o la persistencia. Depender tanto de las clases en Domain, lo hace muy poco extensible, está bastante atado a ellas, lo cual no nos parece demasiado mal, ya que a fin de cuentas son las que queremos persistir en esta capa, siempre irán de la mano.</p>
8	DataAccessInterface	0.08	1	0.05	<p>Este paquete tiene como objetivo reunir todas las interfaces necesarias para que el paquete DataAccess funcione. Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Totalmente abstracto (A). - Totalmente estable (I). - No extensible (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que codificar y volver a compilar otros</p>

					paquetes donde se realizan dichas interfaces, en este caso un cambio en uno de estos métodos afectaría a toda la lógica del negocio ya que aquí reside el Repositorio genérico, por lo que todos usan prácticamente los mismos métodos. No es nada extensible, esta característica de extensibilidad se ve representado en el valor de D.
9	Domain	0	0	0.71	<p>En este paquete se encuentran las entidades que forman el dominio de nuestra solución, por lo tanto tiene sentido que sea:</p> <ul style="list-style-type: none"> - No abstracto (A). - Muy estable (I). - No muy complicado para extender (D). <p>Como las entidades son concretas y no se implementan interfaces, es estable y no abstracto, por lo tanto es fácil de extender esta capa directamente, donde impactará fuertemente será en DataAccess, por lo tanto será menos extensible, como ya mencionamos. Es cierto que este paquete se encuentra en la "zona de dolor", sin embargo no creemos que sea muy necesario un refactor ya que es esperable para este tipo de paquetes.</p>
10	Factory	0.99	0	0.01	<p>Este paquete fue creado para tener una fabrica que permita desacoplarnos de la creación de distintos objetos de la solución, entonces vista las métricas, es:</p> <ul style="list-style-type: none"> - No abstracto (A). - Muy inestable (I). - Nada extensible (D). <p>Es claro que estos resultados tienen sentido ya que aquí estamos dependiendo de muchos paquetes distintos externos, por lo tanto cualquier cambio allí nos podría afectar y nos llevaría a recodificar y recompilar. Además no fue diseñado para ser extensible por lo tanto también tiene sentido el valor obtenido en Distancia. El objetivo de esta fabrica es funcionar correctamente para este sistema en concreto.</p>
11	ImporterInterface	0.6	0.14	0.18	<p>Este paquete tiene como objetivo reunir la interface necesarias para que funcione el importador que será implementado por un tercero. Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Bastante inestable (I). - No extensible (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que cambiar cualquier implementación que no le pertenece al equipo, lo que haría que todas dejen de funcionar si no lo hacen. Por lo tanto, el valor obtenido en D que indica no extensibilidad tiene sentido y es lo que el equipo busca.</p>
12	ImporterLogic	0.96	0.25	0.15	<p>Este paquete se encarga de implementar la interfaz de lógica del importador. Es parecido a la capa BusinessLogic, pero está un poco más desacoplada, por eso tiene sentido obtener un resultado un poco superior en la extensibilidad y en abstracción. Los valores obtenidos reflejan:</p> <ul style="list-style-type: none"> - Poco abstracto (A). - Muy inestable (I). - Muy poco extensible (D). <p>Estos valores nos hacen pensar que quizás haya que mejorar un poco este paquete, ya que es muy probable que se desee extender en un futuro. Estos valores se dan ya que tiene muchas dependencias hacia otros puntos del sistema, como el Dominio y los modelos de entrada, se puede considerar como una posible mejora no depender tanto de ellos.</p>
13	ImporterLogicInterface	0.25	1	0.18	<p>Este paquete tiene como objetivo reunir todas las interfaces necesarias para que funcione el ImporterLogic. Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Totalmente abstracto (A).

					<ul style="list-style-type: none"> - Bastante estable (I). - Difícil de extender (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que codificar y volver a compilar otros paquetes donde se realiza dicha interfaz, en este caso un cambio en uno de estos métodos afectaría a toda la lógica del importador. Es poco extensible, esta característica de extensibilidad se ve representado en el valor de D.</p>
14	JsonContentImporter	1	0	0	<p>Dentro de este paquete está una implementación del importador de contenidos reproducibles.</p> <p>No se debería extender y tampoco se espera que pertenezca a la solución.</p> <p>Entonces se entiende que sea:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Bastante inestable (I). - Difícil de cambiar (D).
15	Migrations	1	0	0	No consideramos necesario el análisis sobre este paquete.
16	Model	0	0	0.71	<p>Este paquete contiene modelos que representan el dominio pero con menos información ya que se expondrán al cliente.</p> <p>Entonces tiene sentido que sea:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Totalmente estable (I). - Bastante extensible (D). <p>No es abstracto porque son todas clases concretas y además es estable porque no implementa ninguna interfaz, entonces es muy fácil de extender, agregar una property no debería afectar casi nada, simplemente habría que configurar el mapeo si se utilizará en otra capa. Aunque este paquete se encuentra en la “zona de dolor”, el equipo no ve necesario un refactor como posible mejora dado el análisis anterior y por el objetivo que tienen los modelos en la api.</p>
17	SessionInterface	0.2	1	0.14	<p>Este paquete tiene como objetivo reunir todas las interfaces necesarias para que funcione el SessionLogic.</p> <p>Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Totalmente abstracto (A). - Bastante estable (I). - Difícil de extender (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que codificar y volver a compilar otros paquetes donde se realiza dicha interfaz, en este caso un cambio en uno de estos métodos afectaría a toda la lógica que maneja la sesión. Es poco extensible y es lo que el equipo buscaba, esta característica de extensibilidad se ve representado en el valor de D.</p>
18	SessionLogic	0.92	0	0.05	<p>En este paquete se implementan las interfaces de la lógica de sesión del sistema, por eso no se espera que sea abstracto.</p> <p>Tiene sentido haber obtenido:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Muy inestable (I). - Nada extensible (D). <p>Estas métricas tienen sentido ya que este paquete depende de distintas cosas, además si llevamos el análisis a nivel de código es cierto que depende de otras capas solo a nivel de interfaz, pero sería de utilidad que esta lógica sea más extensible, por lo tanto se podría haber implementado de alguna manera en que solo dependa de una capa en particular en vez de depender de lógica, dominio y acceso a datos.</p>
19	Validator	0.95	0	0.03	<p>En este paquete se implementa la interfaz genérica de validación de los objetos del sistema, por eso no se espera que sea abstracto.</p> <p>Tiene sentido haber obtenido:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Muy inestable (I).

					<ul style="list-style-type: none"> - Nada extensible (D). <p>Estas métricas tienen sentido ya que este paquete inevitablemente depende de varias cosas porque las debe validar. A pesar de que las métricas hablan de poca extensibilidad, nosotros creemos que por cómo se implementó, respetando SRP y el principio de reuso común para poner juntas únicamente las clases que validan esta clase es más extensible de lo que muestran los valores obtenidos y no debería ser necesario un refactor, ya que puedo modificar cualquier modelo o entidad de dominio sin afectar al validador.</p>
20	ValidatorInterface	0	1	0	<p>Este paquete tiene como objetivo reunir todas las interfaces necesarias para que funcione el Validator.</p> <p>Tiene sentido entonces qué sea:</p> <ul style="list-style-type: none"> - Totalmente abstracto (A). - Bastante estable (I). - Difícil de extender (D). <p>Agregar o cambiar un método en una de esas interfaces implica tener que codificar y volver a compilar otros paquetes donde se realiza dicha interfaz, en este caso un cambio en uno de estos métodos afectaría a toda la lógica que maneja la sesión. Es poco extensible y es lo que el equipo buscaba, esta característica de extensibilidad se ve representado en el valor de D.</p>
21	WebApi	1	0	0	<p>Este paquete contiene toda la lógica de la generación del servicio.</p> <p>Según las métricas es:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Muy inestable (I). - Nada extensible (D). <p>Sin embargo, luego de un análisis a nivel de cobertura se concluye que es bastante más extensible de lo que muestran las métricas, ya que se depende de los modelos, que como ya mencionamos son muy estables y se depende de la factory que no debería cambiar mucho tampoco. Luego tiene tres dependencias salientes a través de interfaces, por lo tanto consideramos que para este caso las métricas obtenidas no reflejan la realidad de este paquete.</p>
22	XmlContentImporter	1	0	0	<p>Dentro de este paquete está una implementación del importador de contenidos reproducibles.</p> <p>No se debería extender y tampoco se espera que pertenezca a la solución.</p> <p>Entonces se entiende que sea:</p> <ul style="list-style-type: none"> - Nada abstracto (A). - Bastante inestable (I). - Difícil de cambiar (D).

En resumen, el diseño es bueno, porque estos paquetes centrales evitan las zonas no deseadas. Se encuentran todas cercanas al segmento, donde se debe apuntar a estar. Esto se dio gracias a que el equipo en todo momento siguió los principios y patrones mencionados anteriormente en este documento.

Mejoras al diseño

- A partir del cambio solicitado para esta segunda entrega, el cual solicita que ahora se puedan importar distintos tipos de contenido reproducible, en particular audio y video, el equipo decidió refactorizar la solución para que la lógica que antes era solo

de audio se pueda reutilizar tanto para audio como para video y en el futuro para otros tipos. Para lograr esto el equipo renombró lo que correspondía a nivel BusinessLogic y DataAccess. Además se agregó una clase que representa a nivel de base de datos todos los tipos de audio disponibles y se agregó en PlayableContent (la nueva clase genérica de contenido reproducible) una FK hacia la tabla con los PlayableContentTypes, que es la nueva clase mencionada. De esta manera el equipo solo tuvo que asignar dependiendo de qué tipo se importa el tipo en esa propiedad y la lógica se encarga de validarlo y persistir en base de datos. Esto claramente da muchas ventajas en cuanto a mantenibilidad porque centraliza la lógica en un solo lugar.

Fue un gran desafío en este punto evitar RTTI a la hora de saber que tipo se está importando, esto se dio ya que decidimos mantener por separado audio y video a nivel de controllers de web api y a nivel de Adapter. Esto es porque nuestro diseño hace independiente a cada capa, entonces vemos conveniente para futuros usos que esas capas se mantengan separadas, ya que entendemos que esto hace más extensible el sistema por si por ejemplo, en un futuro se requieren distintas validaciones o endpoints para cada tipo. Gracias al diseño que logramos es que este cambio no nos influyó demasiado en el impacto del cambio.

- Otro cambio solicitado para esta entrega es el costo de consultas con psicólogos y bonificación. Para cumplir con este requerimiento se agregaron distintas propiedades en las entidades: Patient y Consult, de esta forma se puede gestionar la cantidad de consultas para cada paciente, el costo, la duración y si califica para bonificación o no y si la misma se aprueba o se deniega. Para los diferentes cálculos que se debieron agregar se realizó programación en bloque en la lógica de consulta, con el objetivo de cumplir SRP y ayudar a la mantenibilidad del sistema.

Por otro lado, con el fin de cumplir REST, el equipo creó un nuevo endpoint llamado Bonus, mediante el cual se pueden obtener todos los pacientes que tienen bonificación generada, así como también aprobar o denegar la misma. Esto se decidió para poder cumplir con SRP ya que no había otro controller el cual se correspondiera con esta responsabilidad. Todas las capas necesarias para este nuevo controller se implementaron utilizando TDD y siguiendo los patrones del resto de la solución.

- Otro cambio importante que se realizó es la importación de nuevo contenido a cargo de terceros, como ya se explicó este requerimiento se implementó utilizando reflection para lograr que las nuevas implementaciones puedan ser agregadas en tiempo de ejecución sin necesidad de ningún cambio en la aplicación. Los cambios generales para este requerimiento fueron un nuevo endpoint llamado "Importer", que permite importar un archivo particular, especificando la ruta del mismo y seleccionando una implementación determinada a través de un Id. A su vez, se

agregó un endpoint para obtener todos los tipos disponibles de importadores para una ruta predeterminada. Esta ruta se encuentra en el appSettings, lo cual permite que se pueda cambiar sin necesidad de recompilar y que la ruta sea distinta independientemente del ambiente en el que nos encontremos, esto es de gran utilidad en un ambiente de producción por ejemplo.

- Por último, el equipo dedicó bastante tiempo a refactorear a partir de las correcciones obtenidas. Por ejemplo, un cambio importante es que se agregó el “id” en todos los métodos put, para cumplir con todos los principios REST, ya que de la forma en la que estaba implementado no se especificaba cual entidad se quería actualizar. Otra de las correcciones corregidas es que siempre se devuelven variables en lugar de valores directos de las funciones.

Descripción de jerarquías de herencia utilizadas

No hemos utilizado herencias sino que decidimos utilizar composición.

Decidimos esto principalmente por que no ibamos a obtener un beneficio ya que ninguna entidad de nuestra solución estaba correctamente representada utilizando herencia. Sin embargo, creemos que es un recurso muy útil siempre y cuando el sistema se adecue.

Como hemos dicho anteriormente utilizamos composición, recibiendo objetos (interfaces) en los constructores de los métodos, delegando responsabilidades de que cada clase sepa hacer lo que debe hacer. La combinación de composición con patrones como inyección e inversión de dependencias nos permitieron tener clases que utilicen interfaces bien definidas, delegando las implementaciones a clases concretas.

Anexo

Decisiones Generales

Luego de analizar el código hecho inicialmente donde el mapeo de los modelos a las clases del dominio se realizaba en los controller decidimos quitarle la responsabilidad y crear una nueva capa denominada Adapter. Esta se encargará de dicho proceso y de negociador entre los controllers y la lógica de negocio.

Para el mapeo del adapter utilizamos AutoMapper, consideramos que realizar el mapeo de las clases era algo sencillo y no aportaba un gran conocimiento sino más una pérdida de tiempo. Para esto se crearon Profiles por cada tipo de clase a mapear. Esto nos genera una mayor mantenibilidad.

Se utilizaron “filters” para la autenticación de los administradores y para el manejo de excepciones a nivel de controllers, esto nos evita la duplicación de código. Para el caso particular del de autenticación, este se encarga de tomar el token desde el Header y validarlo, se encapsula la lógica en un solo lugar y este se ejecuta antes de llegar al controller. Por lo que si el token no es válido o inexistente no se accede al controller. Por otro lado, en el caso de las excepciones, todos los bloques try/catch se encuentran en el filter y desde allí se manejan todos los comportamientos dependiendo la excepción capturada.

Separamos los modelos en dos paquetes, los de entrada y los de salida. El propósito fue limitar la información de estos mismos ya que la creación de un objeto puede no ser igual al retorno del mismo.

Diagrama de clases vista lógica

Diagrama de clases del paquete Adapter

Poseen las implementaciones de las interfaces de nuestros adapters, estos se encargan de mapear los modelos a objetos del dominio entre los controllers y la business logic.

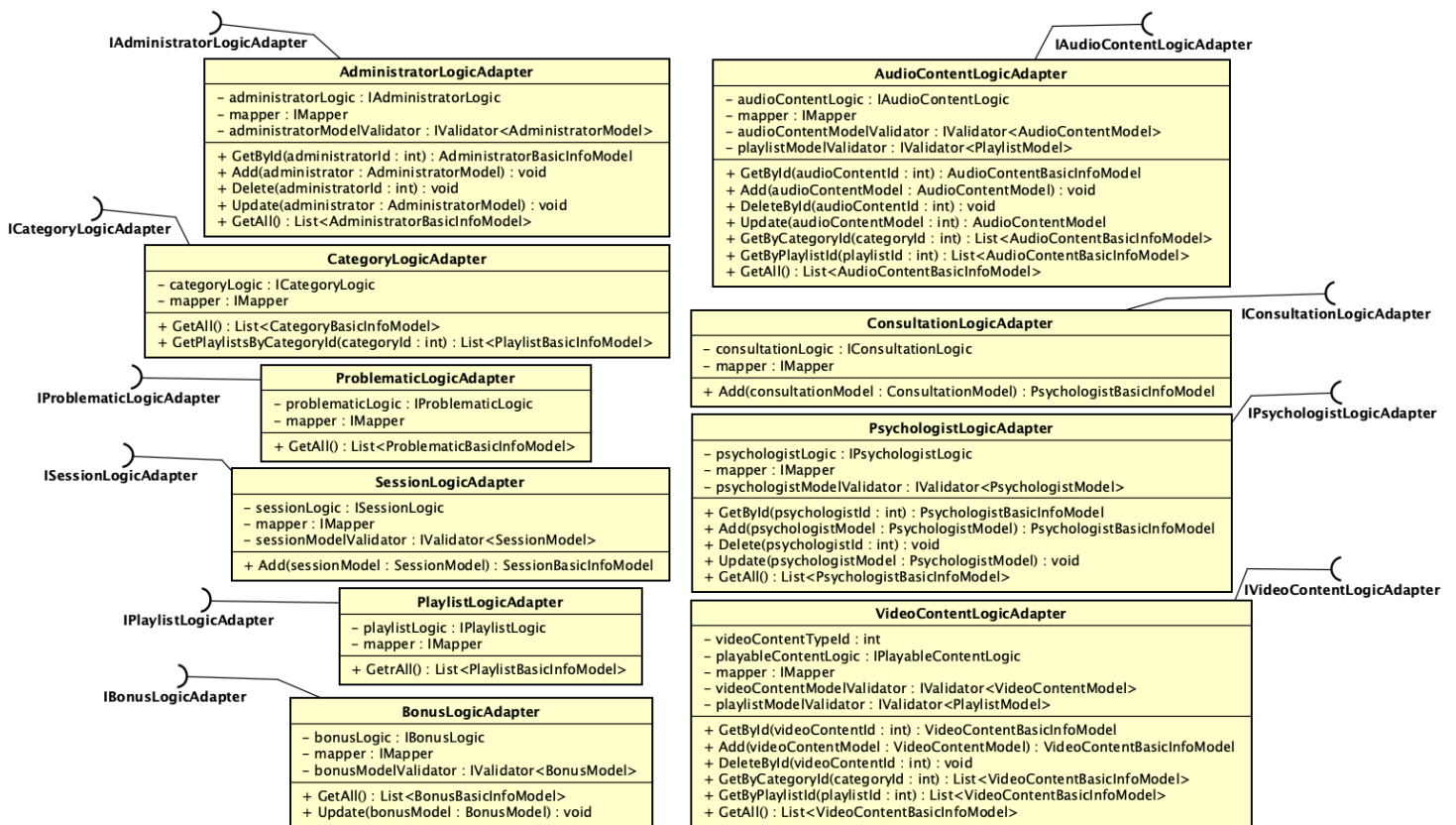


Diagrama de clases del paquete Business Logic

Poseen las implementaciones de la clase del paquete BusinessLogicInterface. Utilizan la interfaz IRepository para interactuar con la base de datos.

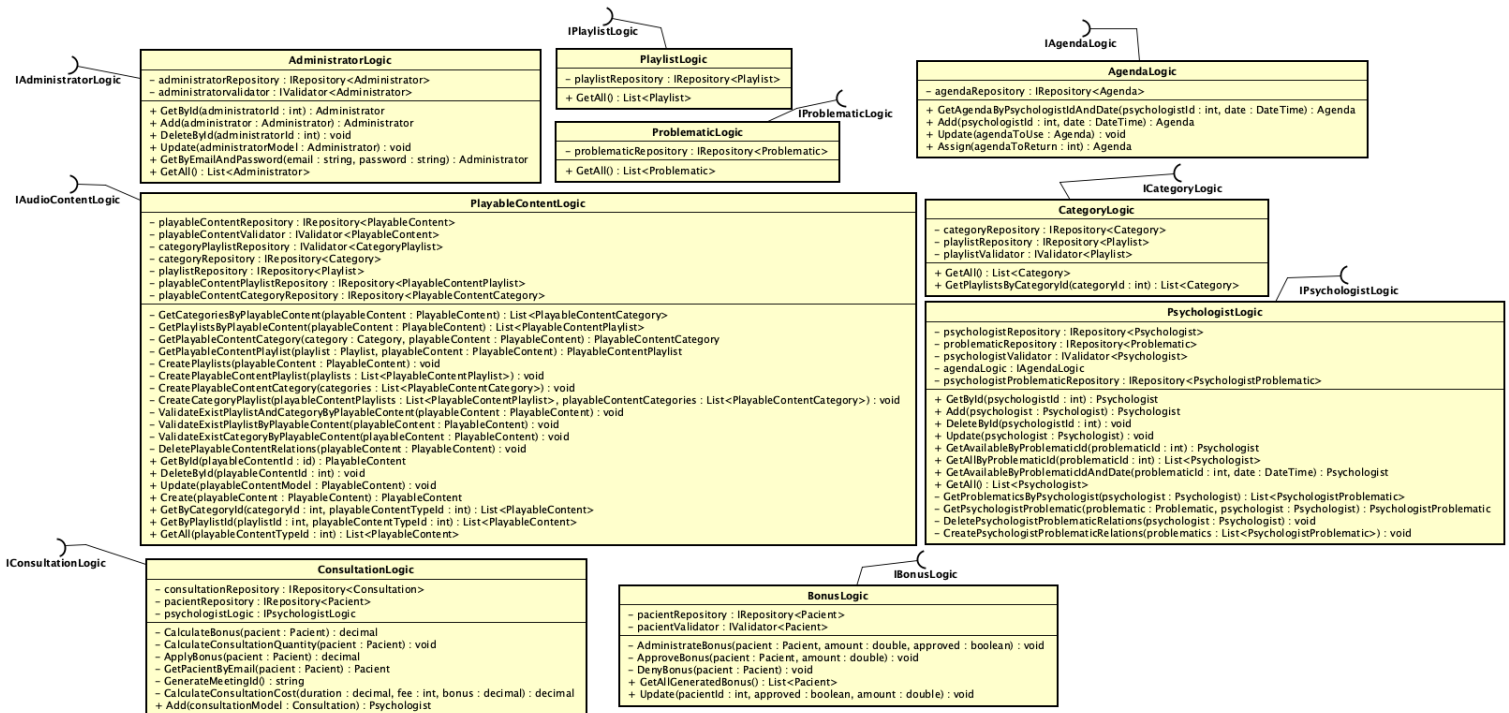


Diagrama de clases del paquete ImporterLogic

Posee la implementación de la clase del paquete ImporterLogicInterface.

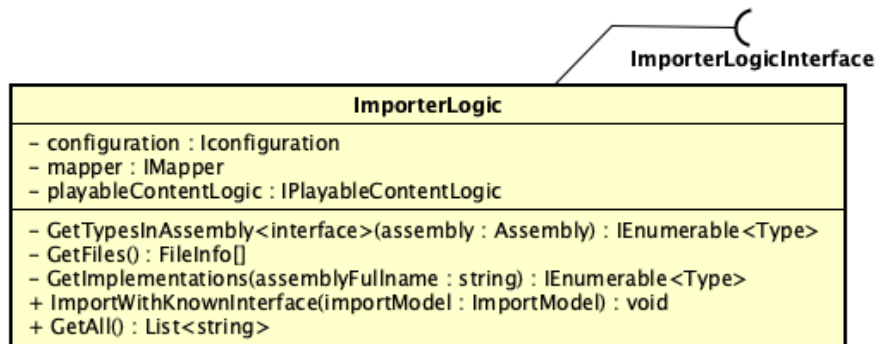


Diagrama de clases del paquete Domain

Posee nuestras entidades. Estas mismas persisten en la base de datos.

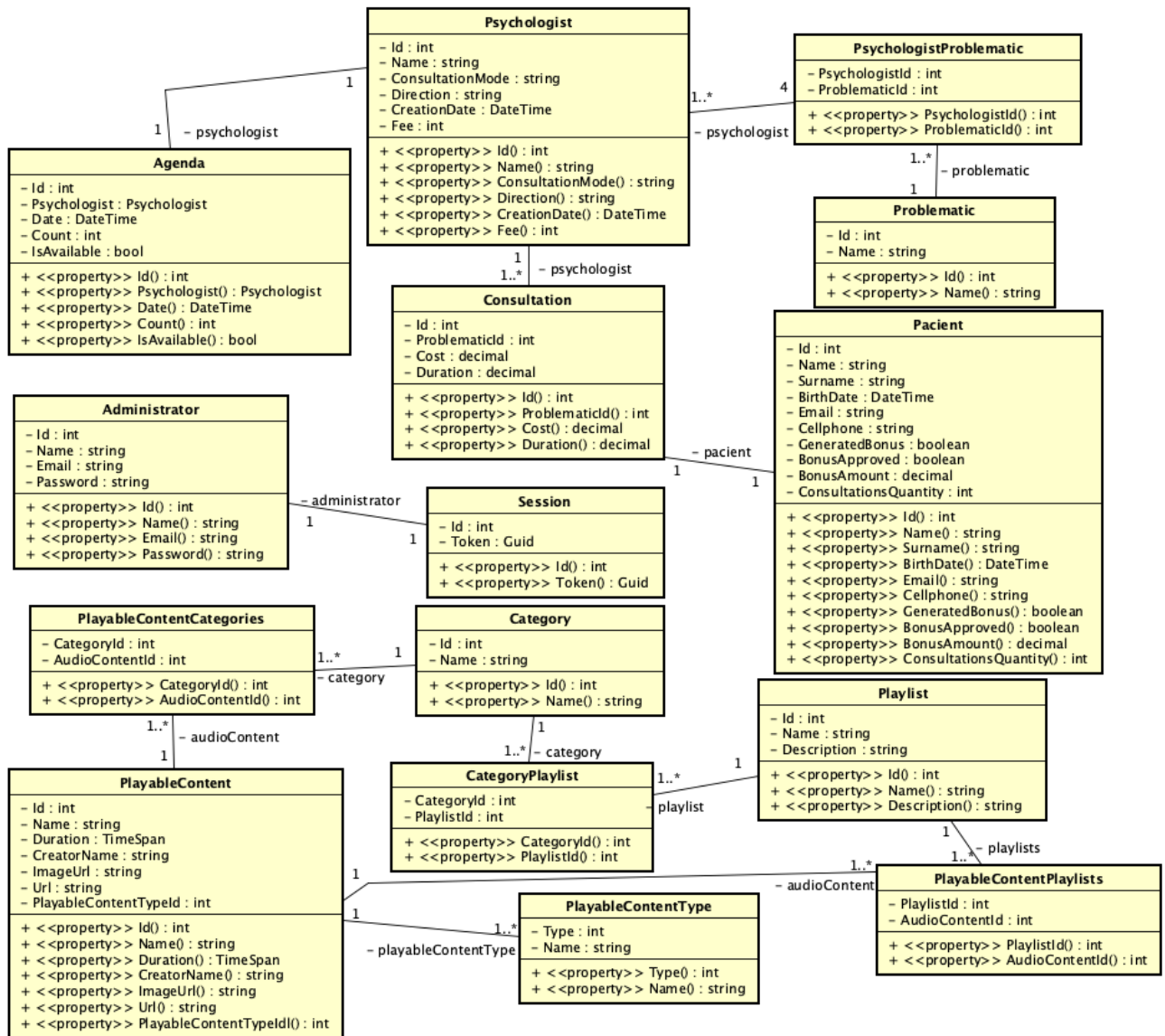
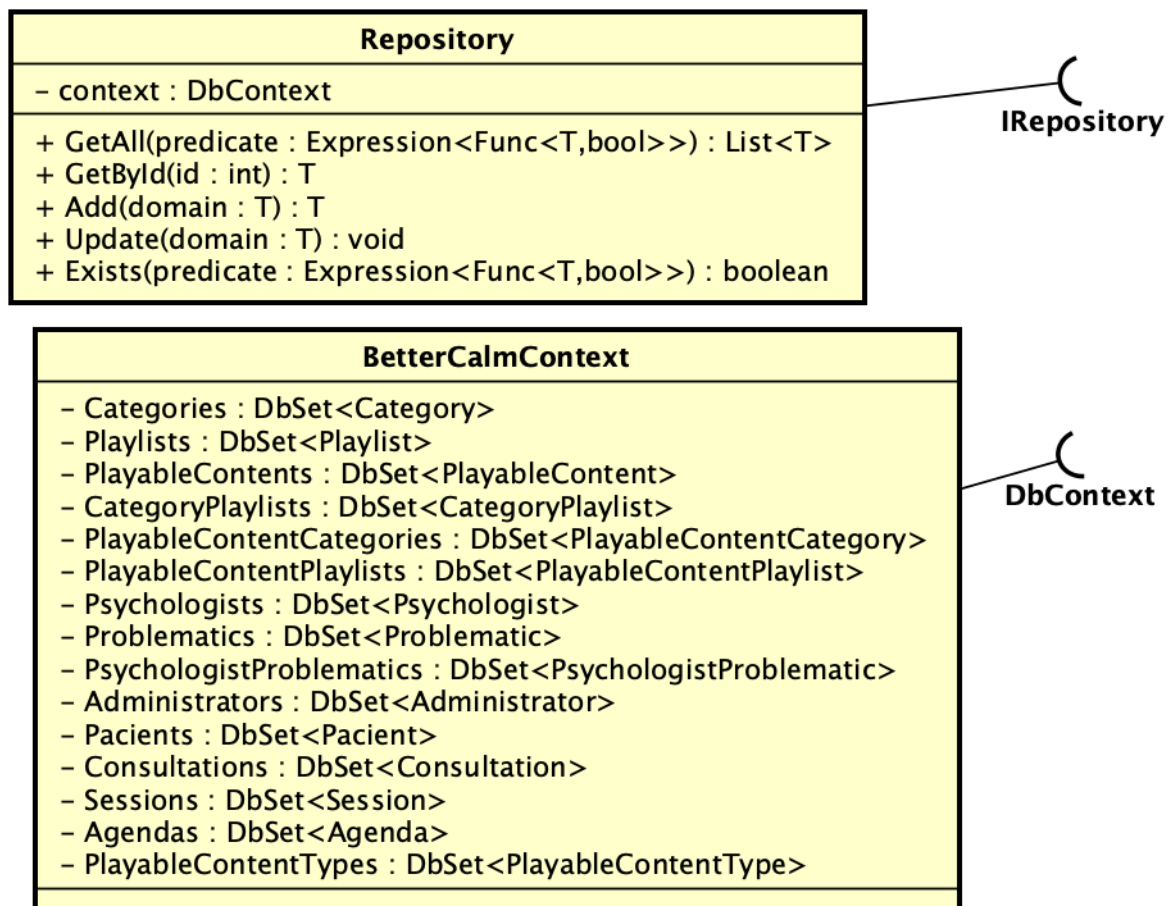


Diagrama de clases del paquete Data Access

Posee la implementación de la clase del paquete DataAccessInterface. También contiene el contexto de nuestra base de datos.



Descripciones de la API

Mecanismo de autenticación

Nos había quedado pendiente dar una explicación de cómo resolvimos dicho proceso. Para esto implementamos la interfaz del Filter Authorization. En su interfaz define el método `OnAuthorization`, aquí nosotros evaluamos si en el header de la llamada se encuentra un token. Si se encuentra presente utilizamos una función provista por `ISessionLogic` para evaluar su validez.

Decidimos utilizar un filtro ya que nos permite ejecutar código antes de procesar la solicitud entrante y también evitar la duplicación de código ya que una gran cantidad de endpoints necesitan de un token. Particularmente este filtro es el primero que se ejecuta en el pipeline por lo que nos generará una rápida respuesta al cliente.

Si al momento de ejecutar la solicitud no se envía un token se le retorna al cliente un status code 401 y el mensaje “Must contain a token to access Api”, en el caso de existir pero no ser válido se le retorna un status code 403 y el mensaje “Forbidden, ask for permission”.

Modificaciones

Para la segunda entrega debimos realizar modificaciones en la API, ya sea por nuevos requerimientos como mejoras funcionales.

Se solicitó que haya un nuevo tipo de contenido reproducible, el tipo de este es video. Por lo que tuvimos que agregar un nuevo controller *VideoContentController*. También se solicitó agregar una tarifa al psicólogo y una duración a la consulta, esto impacto sobre el *ConsultationController*. Además de esto ahora los pacientes que realizan más de cinco consultas pueden obtener una bonificación sobre el costo de la siguiente, por lo que debimos crear un nuevo controller *BonusController* para realizar el manejo de estas. Por

último se solicitó la opción de importar contenido reproducible para los usuarios por lo que generamos un *ImporterController* para realizar las importaciones.

Además de los nuevos requerimientos a la hora de empezar a implementar el front end decidimos agregar ciertos endpoints para realizar una mejor experiencia de usuario. Los controllers afectados fueron *AdministratorController*, *AudioContentController*, *PsychologistController* y además se agregó también un *PlaylistController*. Estos cambios fueron meramente pensados para facilitarle al usuario el uso de la web y presentarle mayor información.

Nuevos controllers

Video Content

GET	/api/videoContents	Obtains the information of all existing video contents.
Obtains the information of all existing video contents.		
Parameters		Try it out
No parameters		
Responses		
Code	Description	Links
200	Success. Returns the requested object.	No links
500	InternalServerError. Server problems, unexpected error.	No links

GET	/api/videoContents/categories/{id}	Obtains all existing video contents for a given category.
Obtains all existing video contents for a given category. An existing category id is required.		
Parameters		Try it out
Name	Description	
id * required integer(int32) (path)	<input type="text" value="id"/>	
Responses		
Code	Description	Links
200	Success. Returns the requested object.	No links
404	NotFound. There is no category registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Video Content

GET

/api/videoContents/playlists/{id} Obtains all existing video contents for a given playlist.

Obtains all existing video contents for a given playlist. An existing playlist id is required.

Parameters

Try it out

Name	Description
id * required integer(int32) (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
404	NotFound. There is no playlist registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

GET

/api/videoContents/{id} Obtains the information of an video content by the id given.

Obtains an video content related by the id given.

Parameters

Try it out

Name	Description
id * required integer(int32) (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
404	NotFound. Video content not exist for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Video Content

DELETE

/api/videoContents/{id}

Deletes an video content.

Deletes the video content with the id given. An administrator token is required.

Parameters

Try it out

Name	Description
id * required integer(int32) (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
200	Success	No links
204	No content.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
404	NotFound. Video content not exist for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

POST

/api/videoContents

Creates an video content.

Creates the video content send in the body. An administrator token is required.

Parameters

Try it out

No parameters

Request body

application/json

Example Value

Schema

```
{
  "id": 0,
  "name": "string",
  "duration": "00:00:00",
  "creatorName": "string",
  "videoUrl": "string",
  "categories": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "playlists": [
    {
      "id": 0,
      "name": "string",
      "description": "string"
    }
  ]
}
```

Responses

Code	Description	Links
200	Success	No links
201	Created.	No links
400	Error. The video content must contain a category.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Video Content

PUT

/api/videoContents

Updates an video content.

Updates the video content with the content send in the body. An administrator token is required.

Parameters

Try it out

No parameters

Request body

application/json

Example Value

Schema

```
{
  "id": 0,
  "name": "string",
  "duration": "00:00:00",
  "creatorName": "string",
  "videoUrl": "string",
  "categories": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "playlists": [
    {
      "id": 0,
      "name": "string",
      "description": "string"
    }
  ]
}
```

Responses

Code	Description	Links
200	Success	No links
204	No content.	No links
400	Error. The video content must contain a category.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
404	NotFound. There is no video registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Bonus

GET

/api/bonuses

Obtains the information of all existing patient with generated bonus.

Obtains the information of all existing patient with generated bonus. An administrator token is required.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
500	InternalServerError. Server problems, unexpected error.	No links

PUT

/api/bonuses

Updates an generated bonus for a patient.

Updates an generated bonus for a patient send in the body. An administrator token is required.

Parameters

Try it out

No parameters

Request body

application/json

Example Value

Schema

```
{
  "patientId": 0,
  "approved": true,
  "amount": 0
}
```

Responses

Code	Description	Links
200	Success	No links
204	No content.	No links
400	Error. The bonus amount value is invalid.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
404	NotFound. There is no patient registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Importer

GET

/api/importers

Obtains the information of all available importers by the configured path.

Obtains the information of all available importers by the configured path. An administrator token is required.

Parameters

No parameters

Try it out

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
400	Error. The configured path is not valid.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
500	InternalServerError. Server problems, unexpected error.	No links

POST

/api/importers

Creates an administrator.

Creates the administrator with the information send in the body. An administrator token is required.

Parameters

No parameters

Try it out

Request body

application/json

Example Value

Schema

```
{  "filePath": "string",  "importerType": "string"}
```

Responses

Code	Description	Links
200	Success	No links
201	Created.	No links
400	Error. The given path is not valid.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Playlists

GET

/api/playlists

Obtains all the playlists.

Obtains all the information of the playlists on the system.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Success. Returns the list of playlists.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Modificaciones o nuevos endpoints en controllers existentes

Consultation

Ahora se recibe un nuevo parámetro indicando la duración de la consulta.

POST

/api/consultations

Creates a consultation with a psychologist.

Creates a consultation with a psychologist. It will return the information of the psychologist and the direction or the link if it's virtual.

Parameters

Try it out

No parameters

Request body

application/json

Example Value

Schema

```
{  "problematicId": 0,  "patient": {    "name": "string",    "surname": "string",    "birthDate": "2021-06-13T22:39:43.714Z",    "email": "string",    "cellphone": "string"  },  "duration": 0}
```

Responses

Code	Description	Links
200	Success	No links
201	Created. Returns the information of the psychologist and the direction or the link if it's virtual	No links
404	Not found. There is no psychologist registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Audio content

GET `/api/audioContents` Obtains the information of all existing audio contents.

Obtains the information of all existing audio contents.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
500	InternalServerError. Server problems, unexpected error.	No links

GET `/api/audioContents/categories/{id}` Obtains all existing audio contents for a given category.

Obtains all existing audio contents for a given category. An existing category id is required.

Parameters

Try it out

Name	Description
id • required <code>integer(\$int32)</code> <i>(path)</i>	<input type="text" value="id"/>

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
404	NotFound. There is no category registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

GET `/api/audioContents/playlists/{id}` Obtains all existing audio contents for a given playlist.

Obtains all existing audio contents for a given playlist. An existing playlist id is required.

Parameters

Try it out

Name	Description
id • required <code>integer(\$int32)</code> <i>(path)</i>	<input type="text" value="id"/>

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
404	NotFound. There is no playlist registered for the given data.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Administrators

GET

/api/administrators

Obtains the information of all existing administrators.

Obtains the information of all existing administrators. An administrator token is required.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
401	Unauthorized. Must contain a token to access Api.	No links
403	Unauthorized. Forbidden, ask for permission.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Psychologists

GET

/api/psychologists

Obtains the information of all existing psychologists.

Obtains the information of all existing psychologists.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Success. Returns the requested object.	No links
500	InternalServerError. Server problems, unexpected error.	No links

Análisis de cobertura de pruebas

Para la ejecución de la cobertura de pruebas decidimos utilizar [ExcludeFromCodeCoverage] en Startup, Program y Migrations. Ya que no son probados por lo que agregarlo en los resultados no aporta valor

Logramos obtener un gran porcentaje de cobertura fluctuando entre noventa y cien por ciento en la mayoría de los proyectos. El único proyecto que no tuvo los mismos resultados es Domain.

Hierarchy ▲	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
▶ juan.gallicchio_LAP0958 2021-06-17 12_...	462	3.31%	13484	96.69%
▶ adapter.dll	84	6.61%	1187	93.39%
▶ adapterexceptions.dll	0	0.00%	14	100.00%
▶ adaptertests.dll	83	3.16%	2547	96.84%
▶ businessexceptions.dll	0	0.00%	6	100.00%
▶ businesslogic.dll	62	7.13%	808	92.87%
▶ businesslogictests.dll	47	1.30%	3566	98.70%
▶ dataaccess.dll	28	3.61%	748	96.39%
▶ dataaccesstests.dll	0	0.00%	402	100.00%
▶ domain.dll	37	24.67%	113	75.33%
▶ model.dll	22	12.50%	154	87.50%
▶ sessionlogic.dll	0	0.00%	32	100.00%
▶ sessionlogictests.dll	4	1.38%	286	98.62%
▶ validator.dll	4	1.98%	198	98.02%
▶ validatortests.dll	33	6.78%	454	93.22%
▶ webapi.dll	7	2.89%	235	97.11%
▶ webapitests.dll	51	1.83%	2734	98.17%

Analizando los porcentajes que se encuentran debajo, creemos que más allá de que en el Dominio el porcentaje cubierto no es bueno en la totalidad, si fue cubierto lo que era relevante de probar y el porcentaje mostrado no es el que realmente importa. Igualmente se alcanzó una excelente cobertura general del proyecto lo que ayuda a poder realizar modificaciones en un futuro y asegurar que el cambio no genere problemas.

Cobertura Adapter:

Presenta una de las coberturas más bajas, sin embargo es un resultado totalmente satisfactorio y se encuentra en los rangos de excelencia.

Cobertura Business Logic:

Aunque no logramos la totalidad de cobertura es un muy buen resultado, creemos que cubrimos satisfactoriamente todo el código siendo la lógica de negocio una parte crucial del sistema.

Cobertura Data Access:

Al igual que en Business Logic, en la Data Access se logra una cobertura muy buena. Y también cumple un rol importante en el sistema.

Cobertura Session Logic:

Se obtuvo una excelente cobertura, le dimos importancia al set de pruebas de dicho proyecto ya que es el encargado de mantener las sesiones de usuario y realizar las validaciones correspondientes.

Cobertura Validator:

Obtuvimos un gran resultado de cobertura, es de suma importancia que su funcionamiento sea el correcto ya que es el encargado de validar las entidades del sistema.

Cobertura Paquetes de excepciones:

Obtuvimos la totalidad de cobertura, si bien no presentan demasiada lógica es importante saber que su funcionamiento es el esperado.

Cobertura WebApi:

Se obtuvo una excelente cobertura, le dimos importancia al set de pruebas de dicho proyecto.

Importar archivos desde la web

Para importar contenido reproducible se debe seguir cierto formato. Hoy en día existen dos tipos diferentes: json y xml. Debajo les dejaremos templates de ayuda, todo archivo a utilizar debe seguir dicho formato. De no ser así la importación no será satisfactoria.

Audio Json

```
{
  "name": "Un nombre",
  "duration": "00:02:50",
  "creatorName": "El creador",
  "imageUrl": "imageUrl",
  "url": "audioUrl",
  "categories": [
    {
      "categoryId": 1
    }
  ],
  "playlists": [
    {
      "playlist": {
        "description": "Descripcion",
        "name": "Audios json"
      }
    }
  ],
  "playableContentTypeId": 1
}
```

Video Json

```
{
  "name": "Un video json",
  "duration": "00:02:50",
  "creatorName": "El creador",
  "imageUrl": "",
  "url": "youtube.com/videoJson",
  "categories": [
    {
      "categoryId": 1
    }
  ],
  "playlists": [
    {
      "playlist": {
        "description": "Descripcion",
        "name": "Videos json"
      }
    }
  ],
  "playableContentTypeId": 2
}
```


Audio Xml

```
<contentImporterModel xmlns:json='false'>
  <url>audioUrl</url>
  <categories json:Array='true'>
    <categoryId>1</categoryId>
  </categories>
  <creatorName>El creador</creatorName>
  <duration>00:02:50</duration>
  <imageUrl>imageUrl</imageUrl>
  <name>Un nombre</name>
  <playableContentTypeId json:Type='Integer'>1</playableContentTypeId>
  <playlists json:Array='true'>
    <playlist>
      <description>Descripcion</description>
      <name>Audios Xml</name>
    </playlist>
  </playlists>
</contentImporterModel>
```

Video Xml

```
<contentImporterModel xmlns:json='false'>
  <url>youtube.com/Video</url>
  <categories json:Array='true'>
    <categoryId>2</categoryId>
  </categories>
  <creatorName>El creador</creatorName>
  <duration>00:07:30</duration>
  <imageUrl></imageUrl>
  <name>Un video</name>
  <playableContentTypeId json:Type='Integer'>2</playableContentTypeId>
  <playlists json:Array='true'>
    <playlist>
      <description>Descripcion</description>
      <name>Videos</name>
    </playlist>
  </playlists>
</contentImporterModel>
```

Cohesión relacional

Assemblies 	Relational Cohesion
Adapter v1.0.0.0	0.5
AdapterExceptions v1.0.0.0	0.14
AdapterInterface v1.0.0.0	0.09
BusinessExceptions v1.0.0.0	0.33
BusinessLogic v1.0.0.0	0.2
BusinessLogicInterface v1.0.0.0	0.11
DataAccess v1.0.0.0	0.91
DataAccessInterface v1.0.0.0	1
Domain v1.0.0.0	1.47
Factory v1.0.0.0	1
ImporterInterface v1.0.0.0	1
ImporterLogic v1.0.0.0	0.75
ImporterLogicInterface v1.0.0.0	1
JsonContentImporter v1.0.0.0	1
Migrations v1.0.0.0	1
Model v1.0.0.0	0.59
SessionInterface v1.0.0.0	1
SessionLogic v1.0.0.0	1
Validator v1.0.0.0	0.07
ValidatorInterface v1.0.0.0	1
WebApi v1.0.0.0	0.94
XmlContentImporter v1.0.0.0	1

Estabilidad/Inestabilidad

Assemblies	Afferent Coupling	Efferent Coupling	Relational Cohesion	Instability
Adapter v1.0.0.0	1	76	0.5	0.99
AdapterExceptions v1.0.0.0	19	0	0.14	0
AdapterInterface v1.0.0.0	23	18	0.09	0.44
BusinessExceptions v1.0.0.0	19	0	0.33	0
BusinessLogic v1.0.0.0	1	37	0.2	0.97
BusinessLogicInterface v1.0.0.0	21	9	0.11	0.3
DataAccess v1.0.0.0	1	84	0.91	0.99
DataAccessInterface v1.0.0.0	12	1	1	0.08
Domain v1.0.0.0	50	0	1.47	0
Factory v1.0.0.0	1	86	1	0.99
ImporterInterface v1.0.0.0	4	11	1	0.73
ImporterLogic v1.0.0.0	1	25	0.75	0.96
ImporterLogicInterface v1.0.0.0	3	1	1	0.25
JsonContentImporter v1.0.0.0	0	13	1	1
Migrations v1.0.0.0	0	4	1	1
Model v1.0.0.0	56	0	0.59	0
SessionInterface v1.0.0.0	4	1	1	0.2
SessionLogic v1.0.0.0	1	12	1	0.92
Validator v1.0.0.0	1	20	0.07	0.95
ValidatorInterface v1.0.0.0	27	0	1	0
WebApi v1.0.0.0	0	122	0.94	1
XmlContentImporter v1.0.0.0	0	18	1	1

Abstracción

Assemblies	Relational Cohesion	Instability	Abstractness	Distance
Adapter v1.0.0.0	0.5	0.99	0.04	0.02
AdapterExceptions v1.0.0.0	0.14	0	0	0.71
AdapterInterface v1.0.0.0	0.09	0.44	1	0.31
BusinessExceptions v1.0.0.0	0.33	0	0	0.71
BusinessLogic v1.0.0.0	0.2	0.97	0	0.02
BusinessLogicInterface v1.0.0.0	0.11	0.3	1	0.21
DataAccess v1.0.0.0	0.91	0.99	0	0.01
DataAccessInterface v1.0.0.0	1	0.08	1	0.05
Domain v1.0.0.0	1.47	0	0	0.71
Factory v1.0.0.0	1	0.99	0	0.01
ImporterInterface v1.0.0.0	1	0.73	0.12	0.1
ImporterLogic v1.0.0.0	0.75	0.96	0.25	0.15
ImporterLogicInterface v1.0.0.0	1	0.25	1	0.18
JsonContentImporter v1.0.0.0	1	1	0	0
Migrations v1.0.0.0	1	1	0	0
Model v1.0.0.0	0.59	0	0	0.71
SessionInterface v1.0.0.0	1	0.2	1	0.14
SessionLogic v1.0.0.0	1	0.92	0	0.05
Validator v1.0.0.0	0.07	0.95	0	0.03
ValidatorInterface v1.0.0.0	1	0	1	0
WebApi v1.0.0.0	0.94	1	0	0
XmlContentImporter v1.0.0.0	1	1	0	0

Link a repositorio

<https://github.com/ORT-DA2/CarbonellGallicchio.git>

Super-Admin

El usuario de correo: admin@gmail.com y contraseña 1234 tiene permisos super admin.

Urls de video content

Para que se logren visualizar correctamente los videos en el sistema es necesario proporcionar el link embedded proporcionado por el servidor de video. En el caso de youtube hay que seleccionar la opción Insertar y luego copiar el link que es indicado.