Universidad ORT Uruguay
Facultad de Ingeniería

Diseño de Aplicaciones 2

Obligatorio 1

Descripción del Diseño

Entregado como requisito de la materia Diseño de Aplicaciones 2

2021

Diego Scaffo - 196673
José Pedro Amado - 188885

Índice

1. Descripción general del trabajo

La solución implementada se encarga de la gestión de contenidos reproducibles y de la agenda de psicólogos para el sistema BetterCalm. Se desarrolló una WebAPI Rest que soporta las siguientes funcionalidades:

- **Manejo de sesiones:** Los usuarios pueden iniciar sesión, utilizando su email y contraseña. El sistema devuelve un token en caso de que las mismas sean válidas y ese token es utilizado en el Header de Authorization para las funcionalidades que requieren un rol específico. Asimismo, se contempló la asignación de roles, por lo que es viable tener más de un rol en el sistema por usuario, considerando además que el sistema controla que el token pertenezca a un usuario que tenga el rol adecuado para la ejecución de cada funcionalidad.
- **Mantenimiento de administradores:** Los administradores pueden visualizar, crear, modificar y eliminar a otros administradores a través de la WebAPI, que provee un recurso específico para este fin (/api/administrators). En este sentido, cabe destacar que en los datos iniciales, se tiene precargado un usuario que cuenta con el rol de administrador.
- **Agenda de consultas:** Se soporta que los usuarios agenden consultas con un psicólogo, enviando sus datos personales y la dolencia (illness) por la que consultan. El sistema es quien define qué psicólogo va a atender la consulta.
- **Visualización de categorías:** Se cuenta con categorías predefinidas y con un endpoint que permite visualizarlas, así como cuáles son los contenidos y las playlists asociadas a esas categorías.
- **Mantenimiento de contenidos:** Los usuarios que tienen el rol de administrador pueden crear, actualizar y eliminar contenidos reproducibles. También se puede asociar un contenido reproducible a una playlist existente o a una que no exista (la playlist se crea).
- **Visualización de dolencias (Illnesses):** Los usuarios tienen la posibilidad de consultar cuáles son las dolencias para las cuales pueden agendar consulta actualmente. Al igual que las categorías, las dolencias están precargadas en el sistema, aunque siendo entidades de la base de datos, se pueden cargar nuevos valores en cualquier momento.
- **Visualización de playlists**: Los usuarios pueden visualizar el contenido asociado a las playlists existentes, pueden obtener todas las playlists o una específica.
- **Mantenimiento de psicólogos**: Los usuarios con el rol de administrador pueden visualizar, crear, actualizar y eliminar psicólogos, con sus datos personales, su forma de atención (Presencial o remoto) y las dolencias en las que el mismo es experto.

Mejoras posibles:

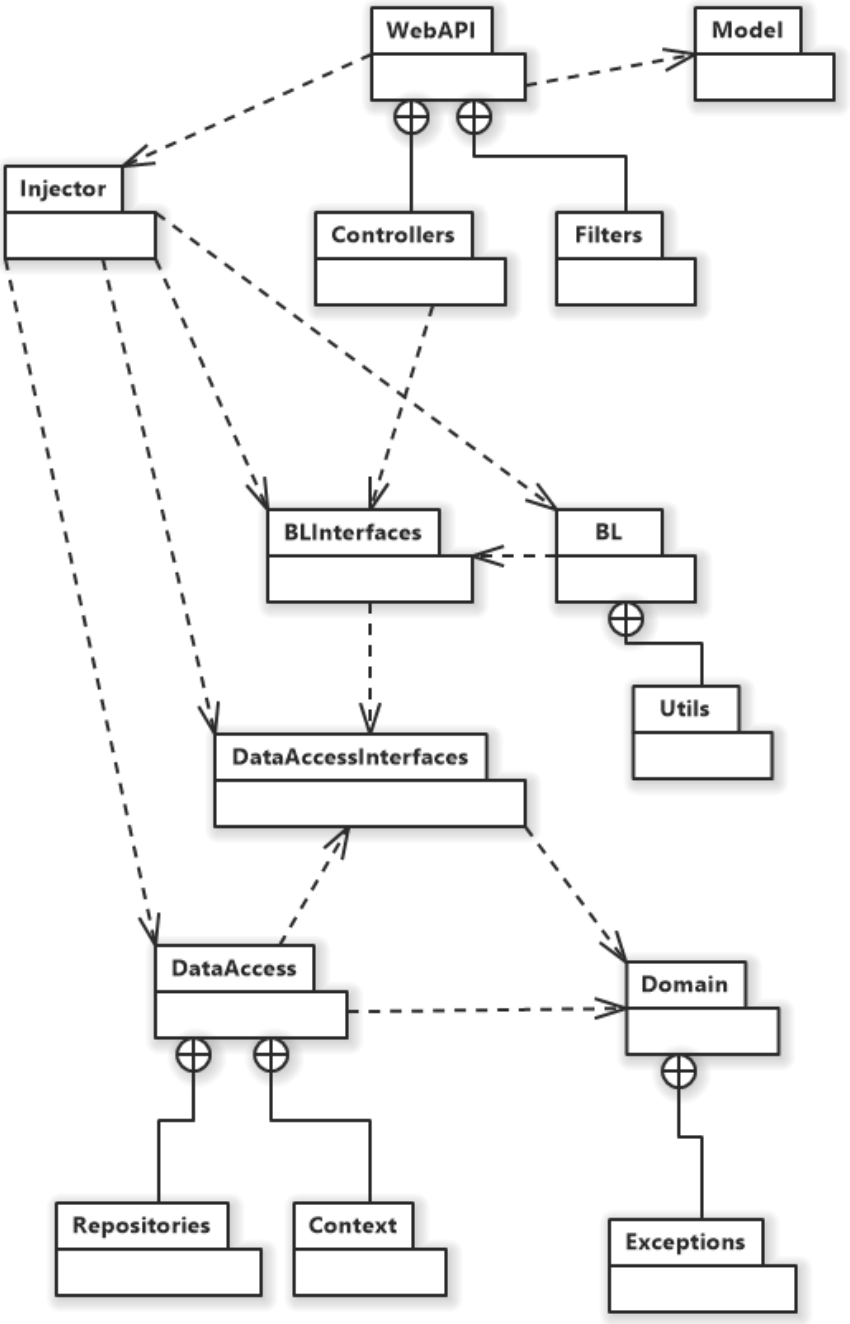
- Actualmente aquellos endpoints que devuelven contenidos reproducibles muestran como duración del mismo el ToString() del TimeSpan asociado, esto es bastante desprolijo de cara al usuario y además aumenta bastante el tamaño de la response.
- Si bien separamos el manejo de usuarios del manejo de roles, sólo proveemos un endpoint específico para el mantenimiento de administradores, dado que es lo requerido por el cliente. En futuras iteraciones sería deseable generalizar ese endpoint para recibir el rol que se le va a asignar al usuario.

Funcionalidades no implementadas:

- No identificamos funcionalidades que no hayan sido implementadas.

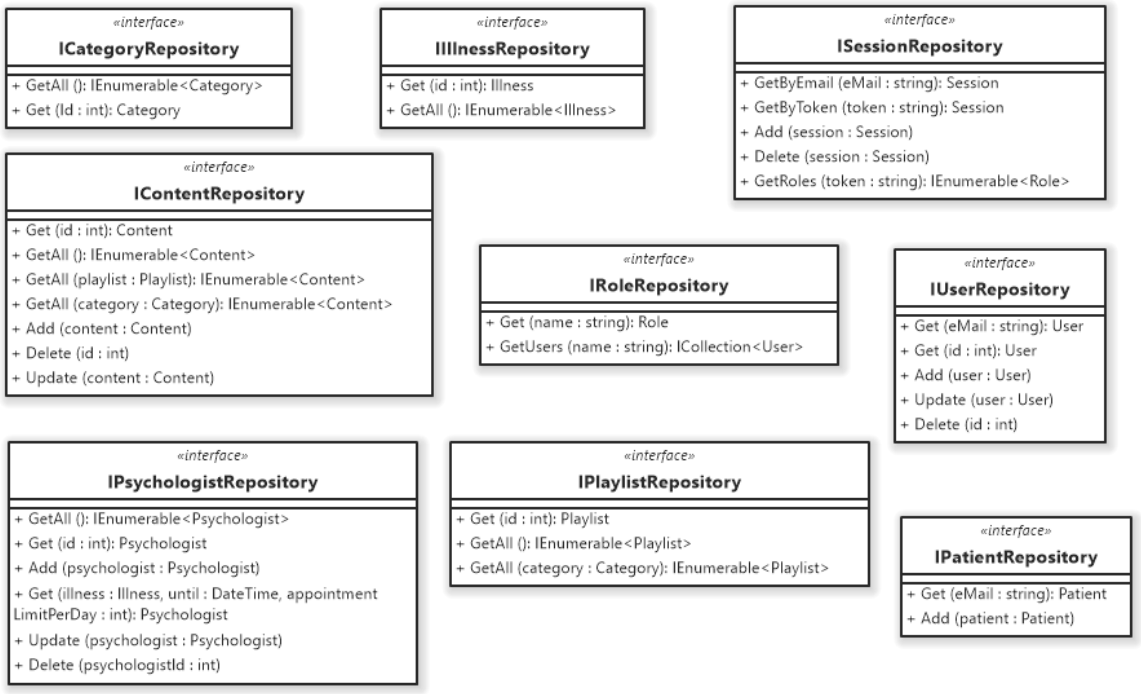
2. Diagrama general de paquetes

A continuación se presenta el diagrama general de paquetes correspondiente a la solución implementada:



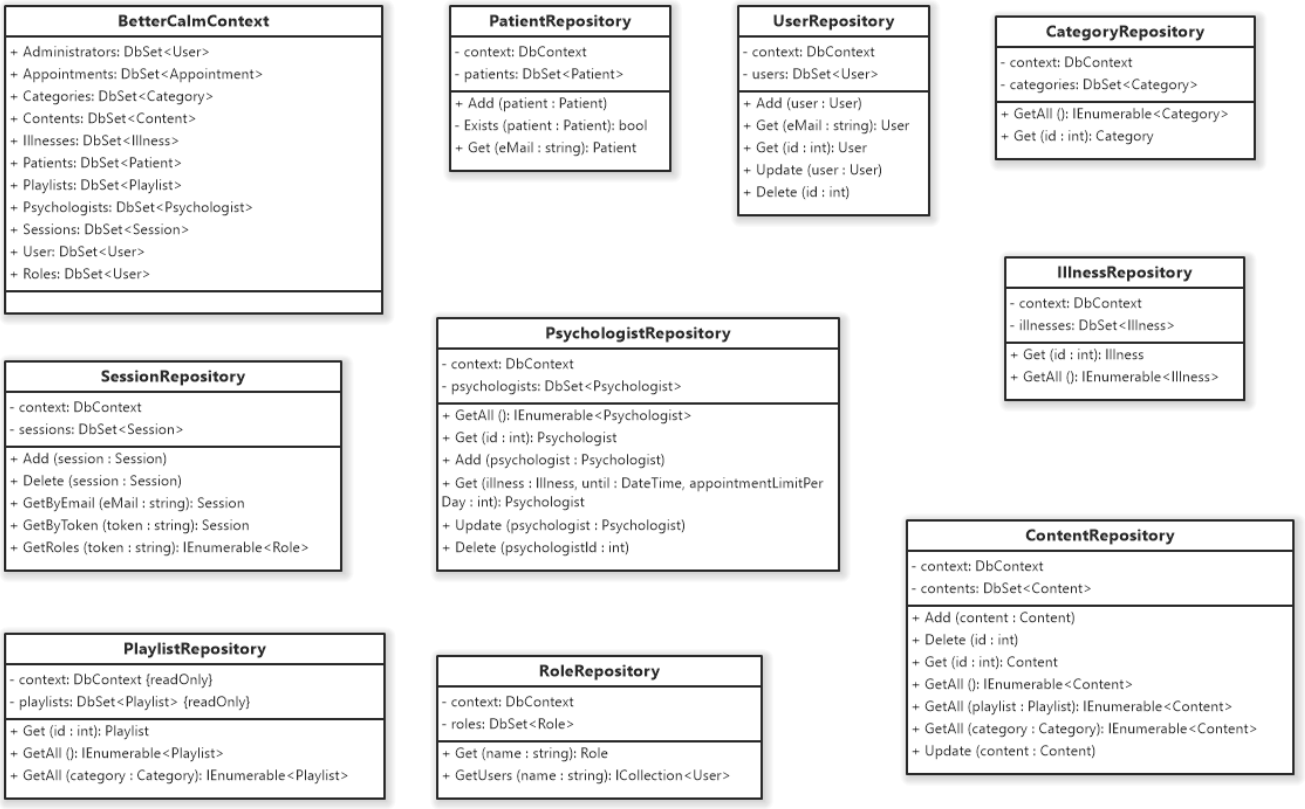
2.1. Descripción de paquetes

DataAccessInterfaces



Este paquete está destinado a definir las interfaces de DataAccess, que contemplan las operaciones de acceso a datos para cada uno de los repositorios pertenecientes a la solución. Este paquete es el que será referenciado por aquellos paquetes que necesiten consumir alguna implementación concreta de DataAccess pero que no deben estar acoplados a la misma.

DataAccess

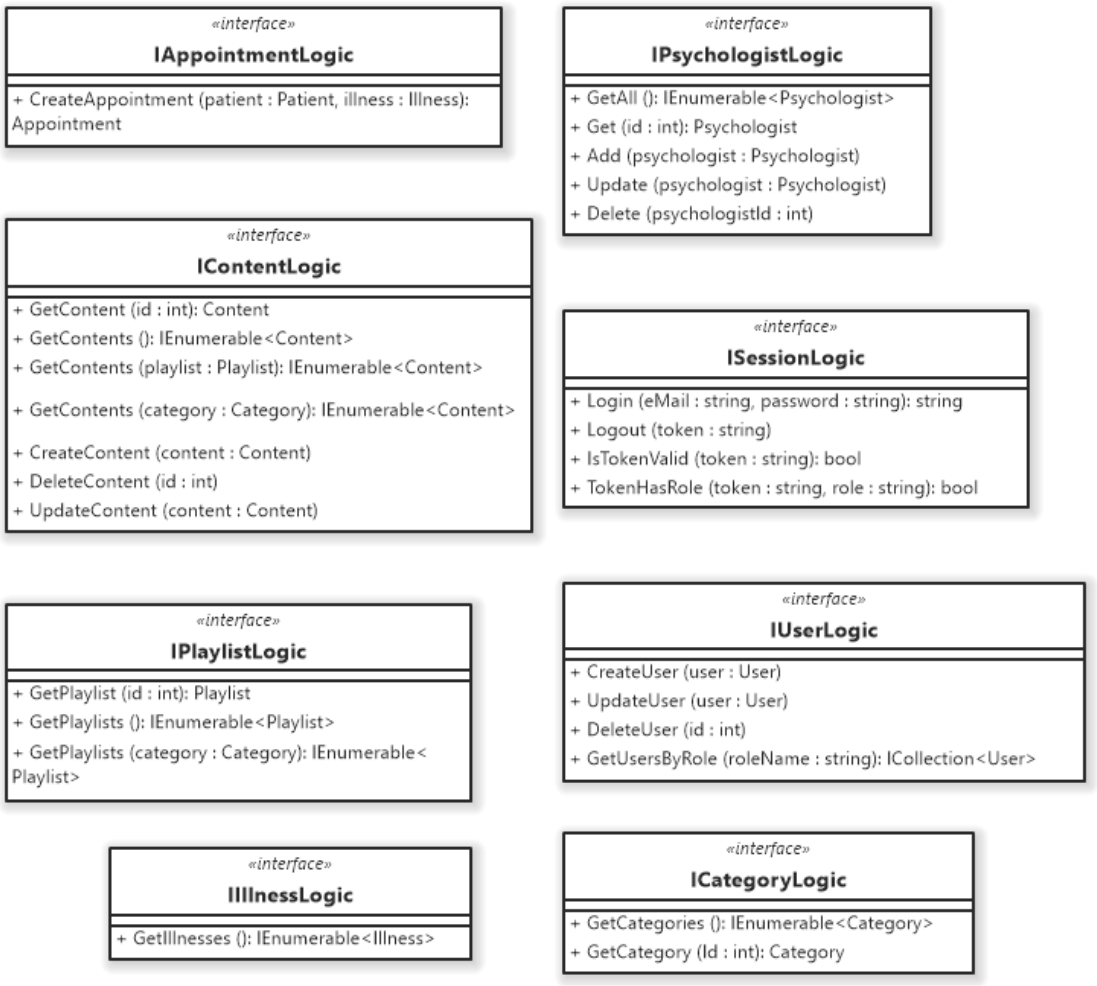


El paquete de DataAccess es el encargado de brindar una implementación concreta para las interfaces de DataAccess. En este caso, se utilizó Entity Framework Core para el acceso a datos, utilizando CodeFirst como approach al manejo de la base de datos.

DataAccess.Test

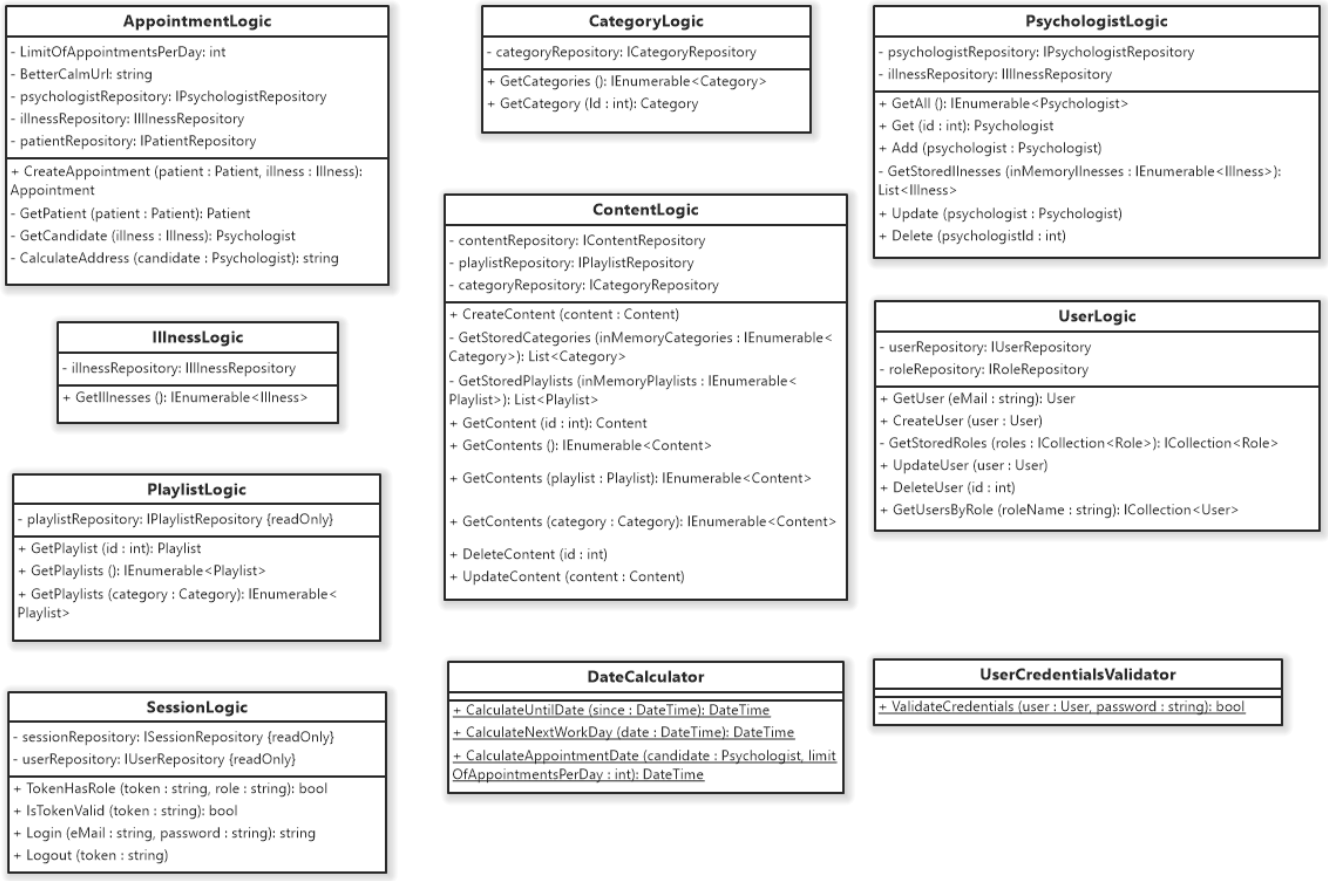
Este paquete se encarga de probar la implementación concreta del DataAccess.

BLInterfaces



Tal como en el caso de DataAccess, contamos con un paquete que se encarga de definir las interfaces necesarias para interactuar con la BL del sistema. Se definen las operaciones necesarias para cumplir con los requerimientos funcionales.

BL

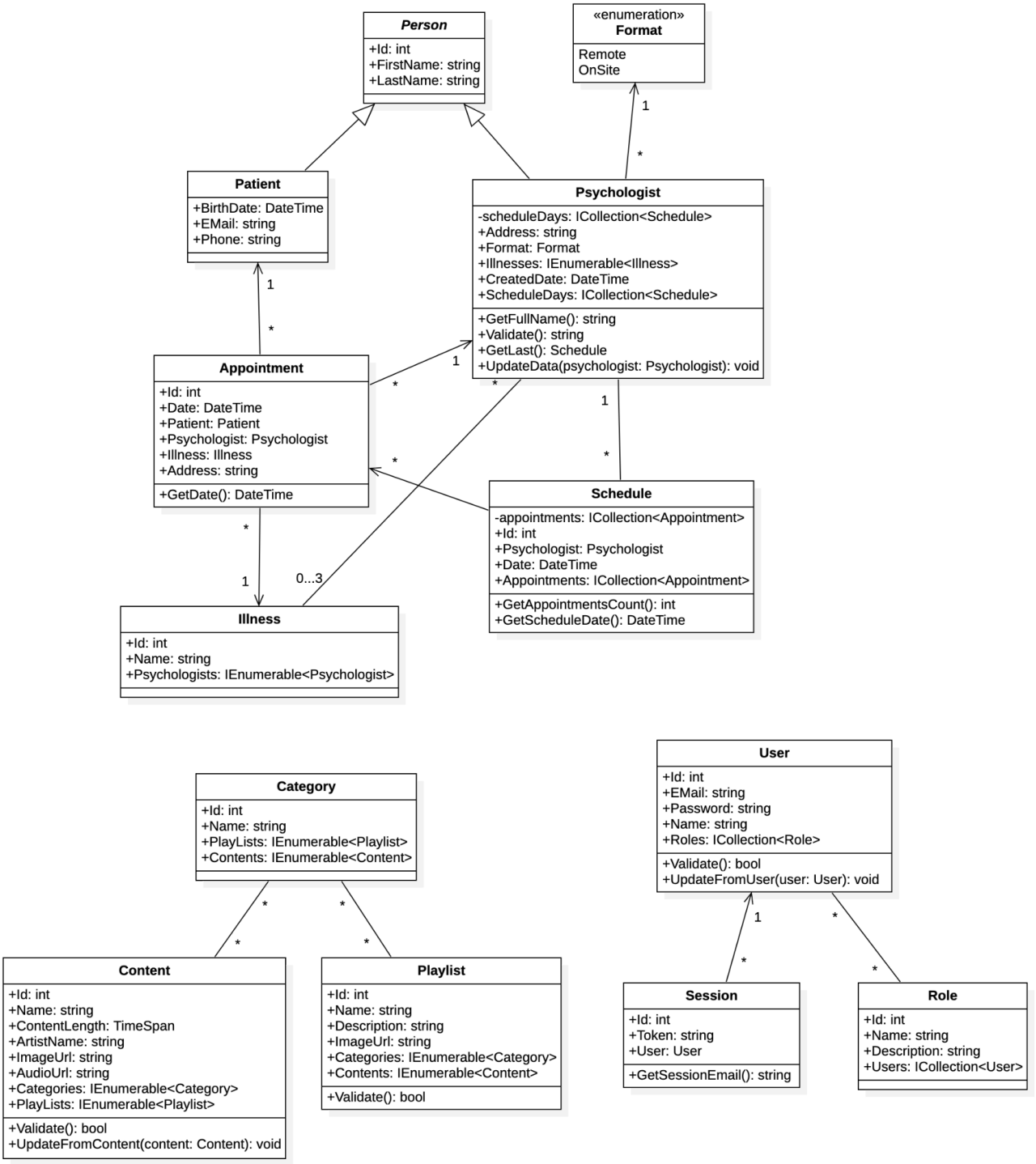


En BL se provee una implementación concreta para las interfaces de BL, los objetos de este paquete, en general, interactúan con las interfaces de DataAccess para poder llevar a cabo las diferentes funcionalidades.

BL.Test

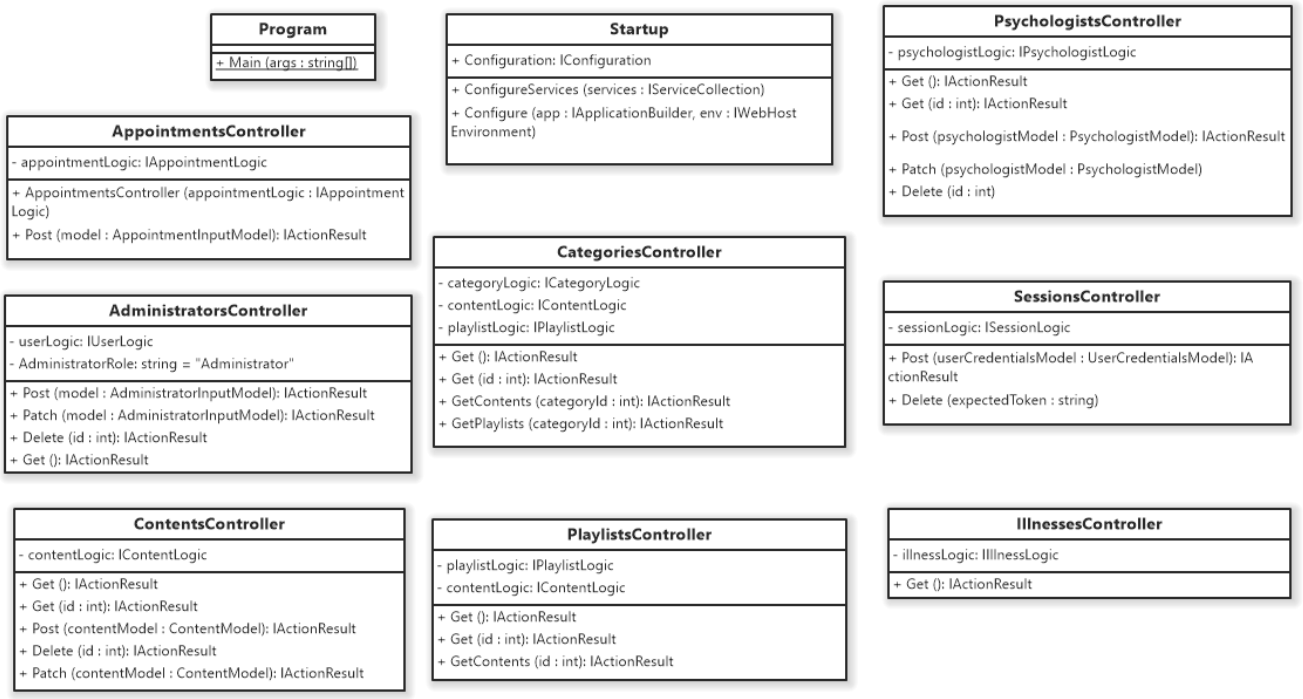
Paquete encargado de probar las funcionalidades expuestas por BL. Se utiliza un framework de mocking (Moq) para poder garantizar que lo que estamos probando es únicamente la lógica de la BL y evitar tener que definir todo lo necesario de DataAccess al probar.

Domain



Este paquete provee la definición de las clases del dominio de la solución.

WebAPI

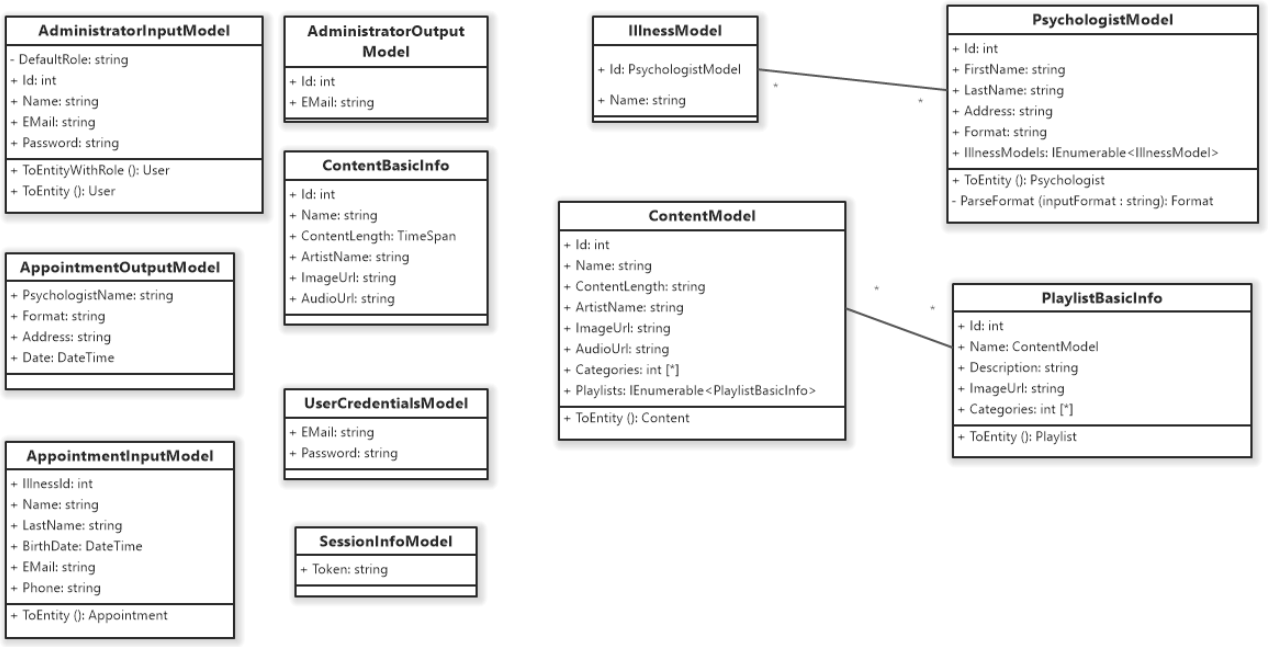


Encargado de exponer los servicios Rest que son soportados por la API a través de sus controllers. Además, cuenta con diferentes filters que permiten manejar las excepciones y lo relacionado a la authorization del usuario para la utilización de las diferentes funcionalidades.

WebAPI.Test

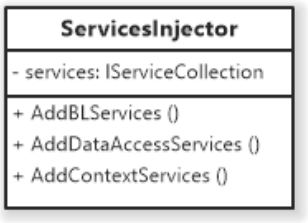
Paquete destinado a probar los diferentes endpoints de la WebAPI. También utiliza Mocks para simular respuestas de la BL.

Model



Define los objetos que serán utilizados para representar la entrada y la salida de los diferentes métodos disponibles en la Rest API, de forma tal que no siempre se tenga que recibir/responder con objetos del dominio.

Injector



Utilizado para definir la inyección de dependencias a nivel general, se comporta como una factory, definiendo qué implementaciones concretas se asocian a cada interface definida en los diferentes paquetes que se relacionan con la WebAPI.

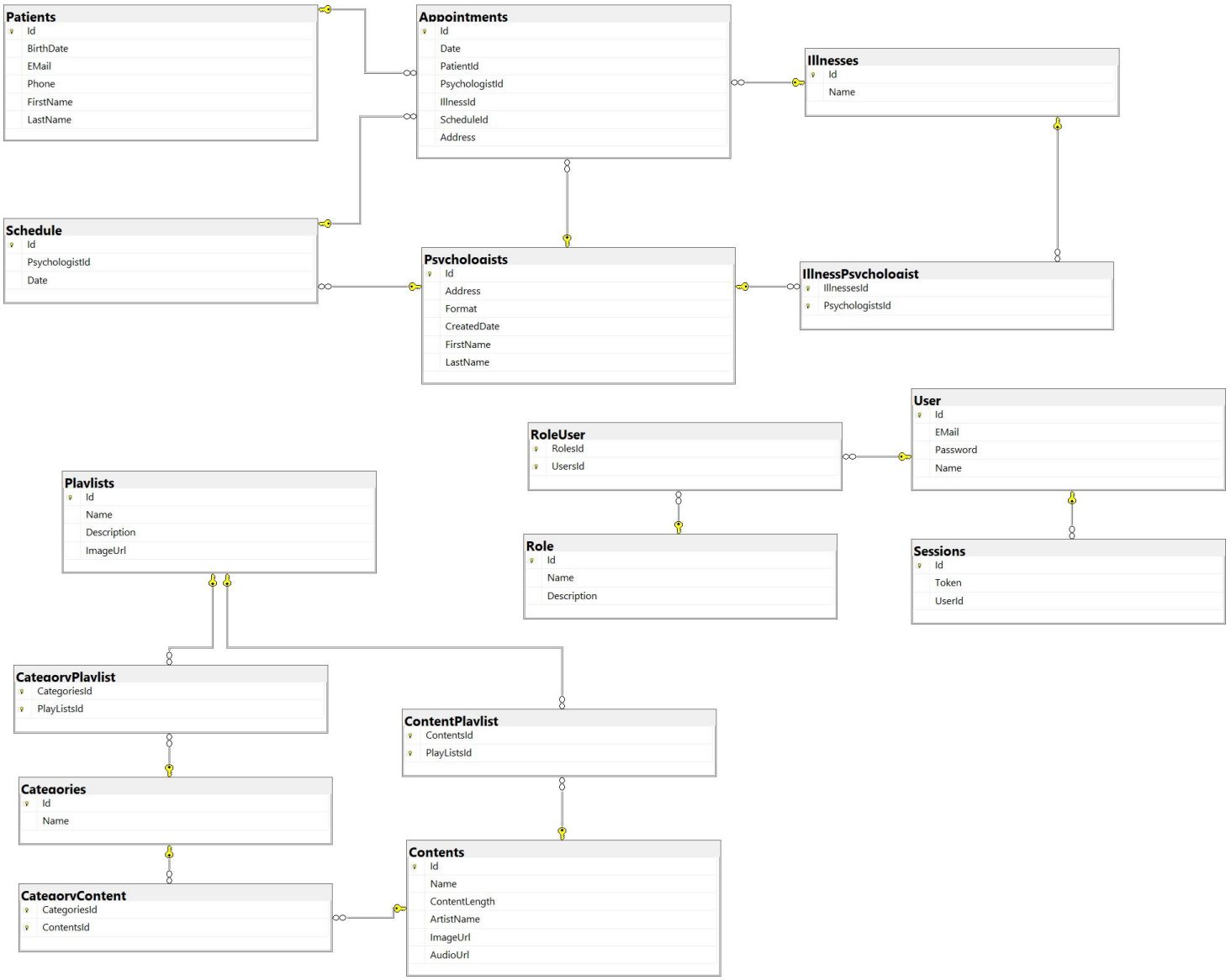
Si bien esto se puede hacer en el propio paquete de la WebAPI, nos pareció oportuno que esté en un paquete independiente con el fin de que la misma no conozca a ninguna implementación concreta (si lo hicieramos en WebAPI, tendríamos dependencias a paquetes de implementaciones concretas).

3. Descripción de jerarquías de herencia utilizadas

Se utilizó herencia solamente para el caso de la definición de Personas, en donde tanto Pacientes como Psicólogos heredan de esa clase en común para reutilizar sus datos. Esto se puede observar en el diagrama de clases del dominio.

4. Modelo de tablas de la estructura de la base de datos

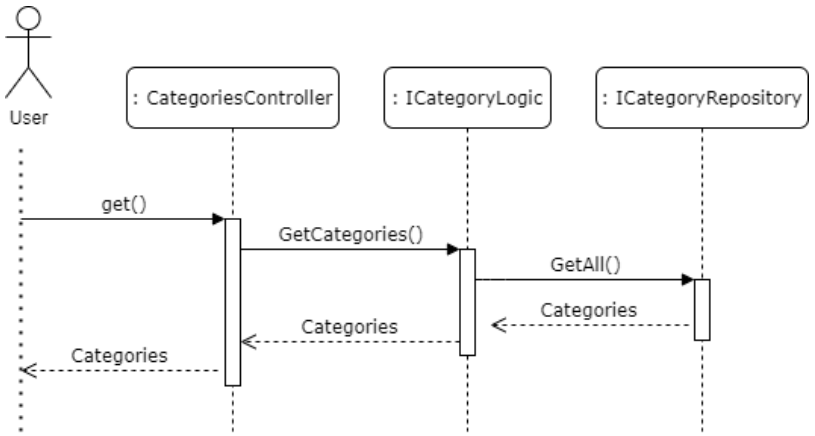
A continuación presentamos el modelo de tablas. Este es el resultado de aplicar las diferentes migraciones que fueron surgiendo a través del desarrollo del sistema, utilizando Entity Framework como Code First.



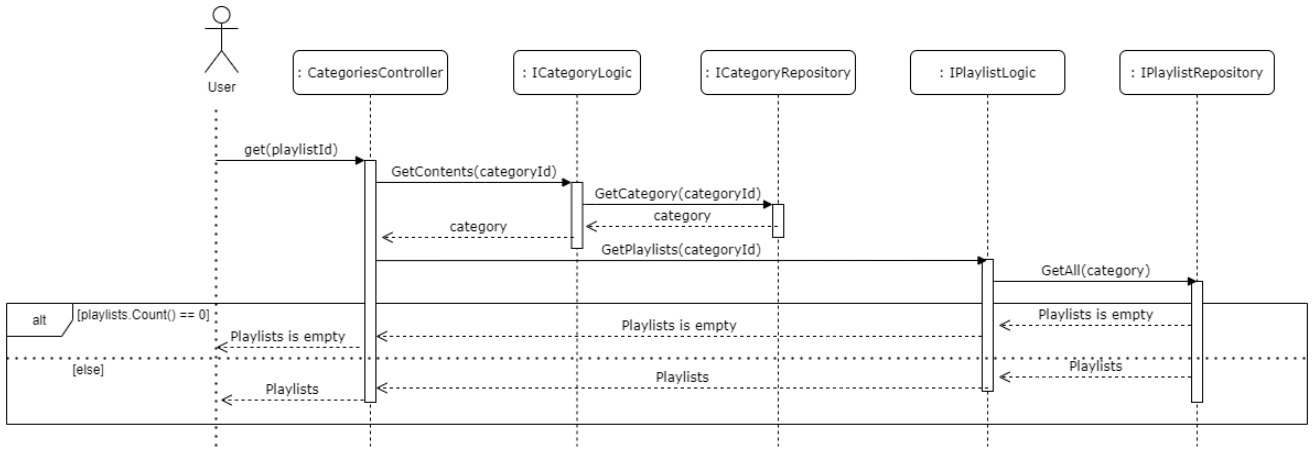
5. Diagramas de interacción

A continuación se detallan los diagramas de secuencia más significativos en cuanto a las funcionalidades principales. Además, se agregan comentarios a aquellos que consideramos que necesitan alguna aclaración en cuanto a sus interacciones.

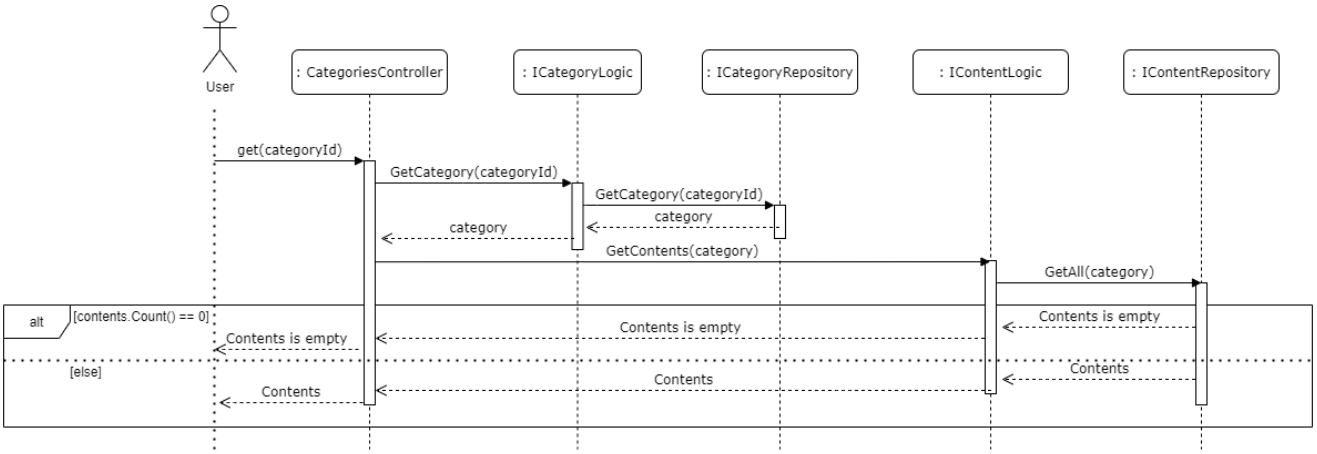
Ver categorías



Ver Playlists de las categorías



Ver contenidos de las categorías



Agendar consultas con un psicólogo

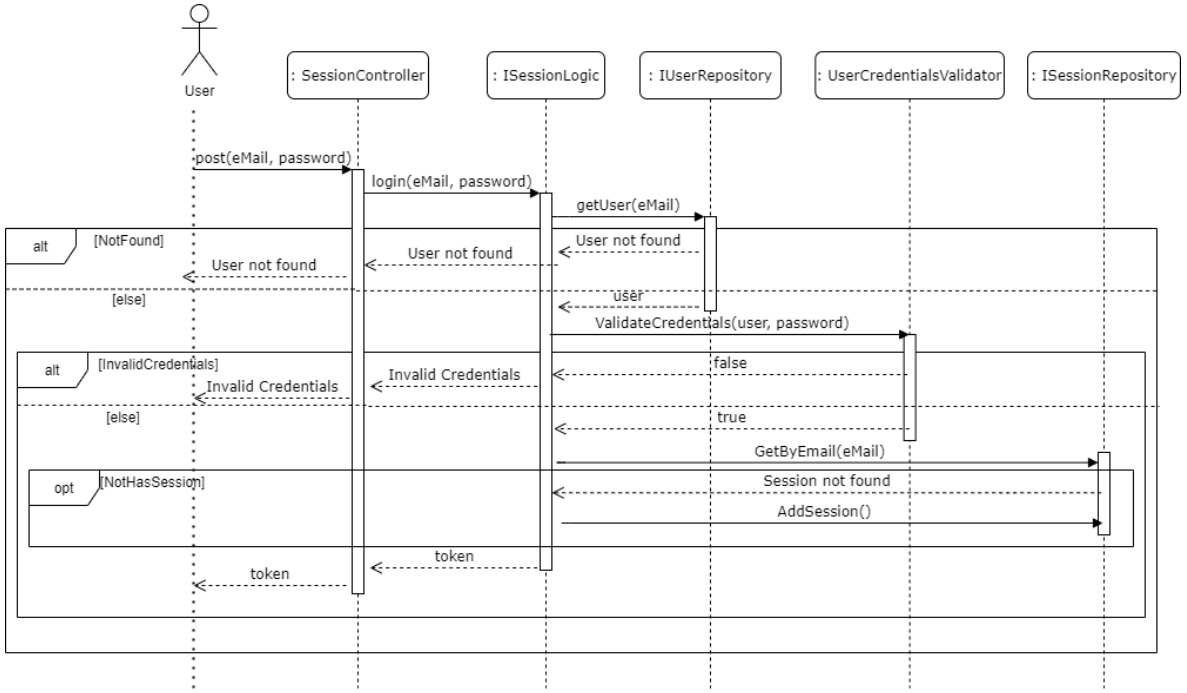
Ver diagrama en Anexo I. Diagramas de secuencia - Agendar consultas con un psicólogo

Para este caso se decidió que la mayor parte de la consulta la resuelva el propio repositorio de psychologist, que es quién conoce a los psicólogos y estos tienen acceso a su agenda. De esta manera, el DataAccess nos puede facilitar la consulta de los psicólogos que atienden determinada dolencia, con un límite de consultas agendadas por día y una fecha límite.

El caso de la fecha límite fue considerado por el requerimiento de que se tiene que controlar la disponibilidad semana a semana, es decir, el psicólogo disponible más antiguo en la semana (iterando semana a semana hasta encontrar uno libre). En este caso también aplicamos fabricación pura para desacoplar los métodos necesarios para el cálculo de fechas (calcular la fecha de la cita en base a las reglas de negocio), que en un principio los teníamos en la AppointmentLogic, atentando contra SRP.

Inicio de sesión

El inicio de sesión es una funcionalidad requerida por algunas de las funcionalidades claves de la solución.

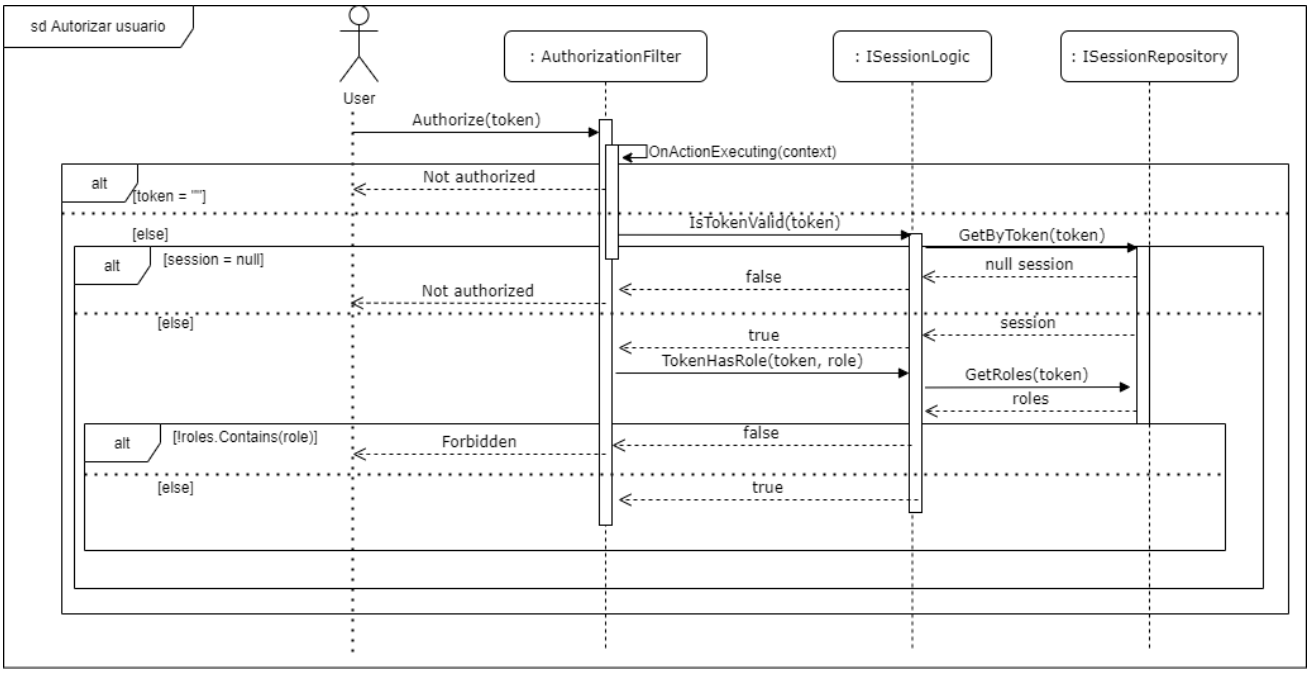


El proceso de inicio de sesión consiste en el envío de email y contraseña por parte del usuario, estas credenciales serán validadas por la SessionLogic (BL), luego de solicitar el usuario al UserRepository (DataAccess).

Para validar las credenciales, se cuenta con una clase UserCredentialsValidator que fue definida específicamente con ese fin, para poder desacoplar esa lógica de la SessionLogic, aumentando la cohesión en la misma y favoreciendo el desacoplamiento (se aplicó fabricación pura).

Una vez validadas las credenciales, se busca una sesión existente, en caso de no existir, se crea una nueva y se devuelve el token correspondiente al usuario. Ese token podrá ser utilizado en el Header Authorization en futuras request.

Autorizar usuario



Decidimos contar con un diagrama de secuencia específico para la autorización de los usuarios dado que es una funcionalidad utilizada por otras funcionalidades. De esta forma, podemos hacer uso del marco ref para reutilizarlo.

Alta contenido

Ver diagrama en Anexo I. Diagramas de secuencia - Alta contenido

Alta psicólogo

Ver diagrama en Anexo I. Diagramas de secuencia - Alta psicólogo

6. Justificación del diseño

En la solución diseñada buscamos apoyarnos en los principios SOLID, en los patrones de asignación de responsabilidades GRASP y en patrones de diseño.

Principalmente, buscamos dividir la solución de forma tal que los paquetes más importantes y más propensos a cambios se comuniquen entre ellos a través de interfaces, de esta manera, podemos cambiar sus implementaciones concretas sin afectar el funcionamiento del resto de los paquetes.

A su vez, intentamos que cada clase tenga un solo motivo para cambiar, lo que en términos generales cumplimos, salvo en los casos de los controladores, que siendo los encargados de recibir datos (que pueden ser modelos específicos, con necesidad de ser convertidos), soportar diferentes verbos y comunicarse con la BL, generalmente tienen más de un motivo para cambiar. También, las diferentes interfaces que definimos fueron pensadas para que quienes las implementen no tengan que implementar cosas que no necesitan, es decir, no se obliga a implementar métodos que van a quedar como NotImplemented.

Lo expuesto anteriormente nos permite cambiar fácilmente las implementaciones porque utilizamos inyección de dependencias, es decir, vamos inyectando diferentes dependencias y no las instanciamos en puntos particulares, sino que delegamos la creación de las instancias “concretas” a una Factory, que en nuestro caso está en el package Injector. Esto favorece enormemente a la mantenibilidad, dado que cambiar una implementación por otra solo implica tocar el paquete en cuestión y como mucho, actualizar la Factory del Injector.

Luego en base a patrones como experto, determinamos que algunos métodos como los de validación de existencia de datos o método de actualización de una instancia en base a otra del mismo tipo, sean potestad de las clases que más conocimiento tienen, en tal sentido, se puede ver en la solución como tenemos métodos como Validate o UpdateFromContent en la clase Content del dominio.

También nos basamos en fabricación pura para la creación de algunas clases que, si bien no representan una entidad del dominio, nos permiten encapsular cierto comportamiento y desacoplarlo de las clases en las que estaban, algunos casos ya los mencionamos anteriormente, como el UserCredentialsValidator o el DateCalculator.

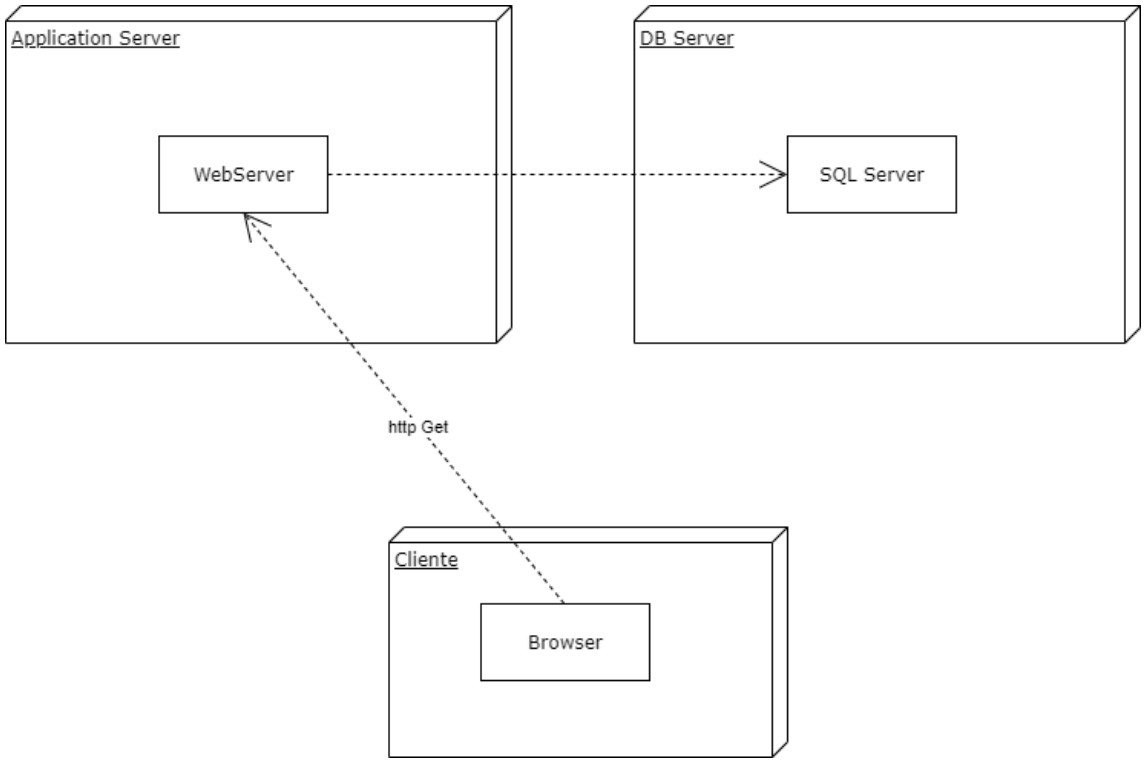
En cuanto a patrones de diseño, más allá de lo ya mencionado, intentamos incluir patrones como Builder, Strategy o Template Method, pero la realidad es que terminaban por complejizar aún más el diseño, siendo que la motivación de los patrones de diseño es justamente lo contrario. Luego el package de Injector, a través de su clase ServiceInjector, hace las veces de Factory dado que se encarga específicamente de definir qué objetos tienen que ser creados para cada interface. Por otra parte, no consideramos necesario usar singleton, dado que si bien podría aplicar para el caso del manejo de contexto, consideramos que al ser una Rest API, es más apropiado que el contexto sea scoped a las requests.

En relación al contexto, en cuanto al acceso a datos, utilizamos SQL Server y todas las consultas, creación de tablas, migraciones, etc, fueron manejadas a través de Entity Framework, utilizando Linq para las consultas. Intentamos en la mayoría de los casos llevar todas las consultas que implican recorridas y filtros a nivel de Data Access, con el fin de tener la mejor performance posible evitando tener que recorrer colecciones en memoria. Para muestra de esto tenemos el caso de la estructura schedule, que nos permite relacionar un psicólogo con sus citas por día, esta estructura nos permitió optimizar la consulta de “obtener psicólogo para determinada dolencia hasta determinada fecha que no tenga más de 5 consultas ya agendadas ese día” sin tener que recorrer una collection de psicólogos en memoria. Cabe destacar que tuvimos la precaución correspondiente de no llevar reglas del negocio a nivel del Data Access, es decir, esas consultas están debidamente parametrizadas.

En cuanto al manejo de excepciones, decidimos tener todas las excepciones en el paquete de dominio, dado que es el paquete que ya está relacionado con todos los demás y nos pareció coherente que las mismas se encuentren a ese nivel. Como política de manejo de excepciones, actualmente se están dejando subir las excepciones hasta el Exception Filter de la WebAPI, que es quien se encarga de hacer el catch y convertirlas en la response adecuada (salvo las que necesitan ser controladas internamente).

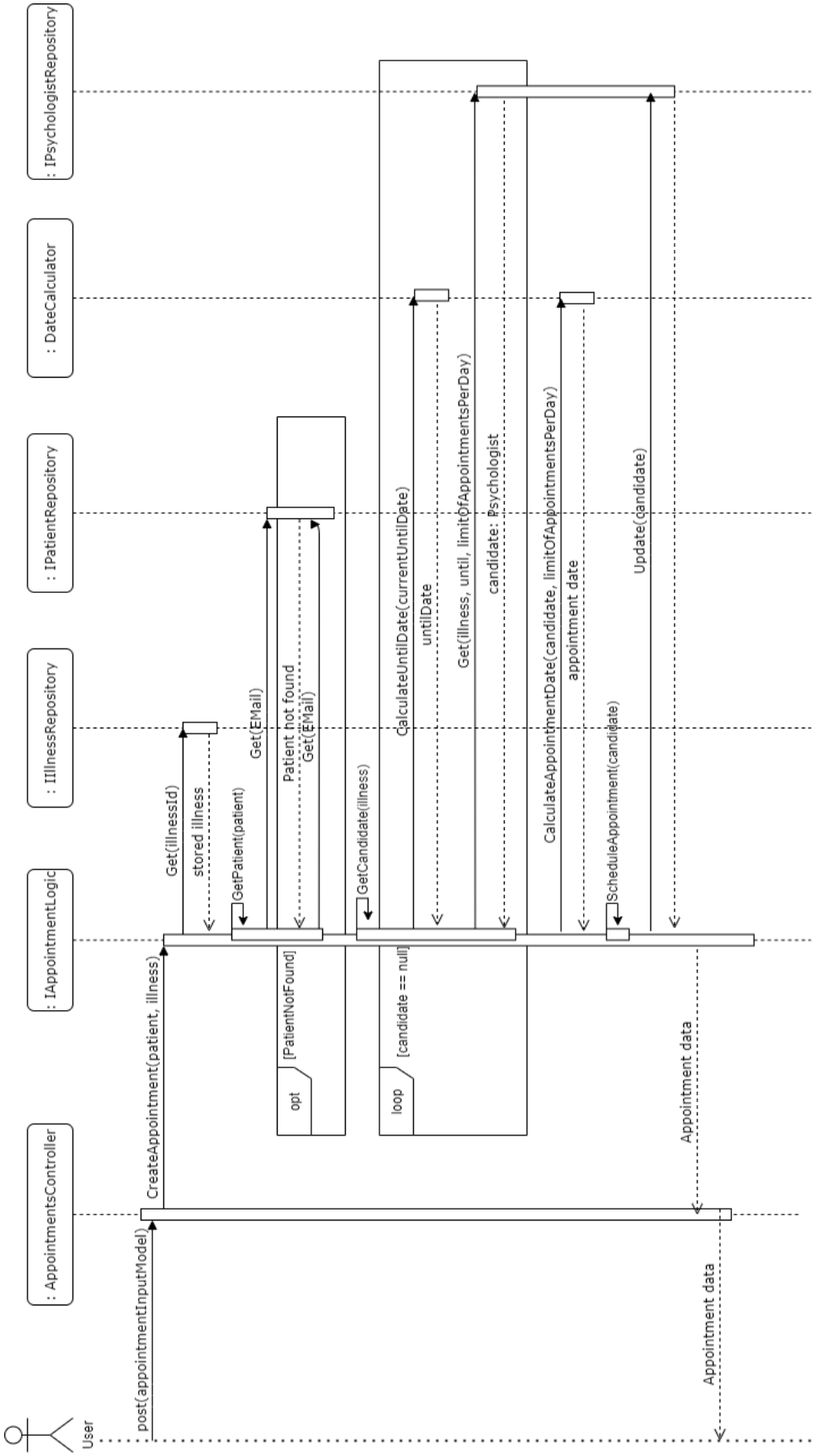
7. Diagrama de implementación

A continuación se presenta el diagrama de implementación, en donde se puede ver la separación en módulos, donde el cliente, a través de un browser se comunica con el web server, que es donde está deployada la WebAPI, que se comunica al DB Server para realizar las consultas correspondientes.

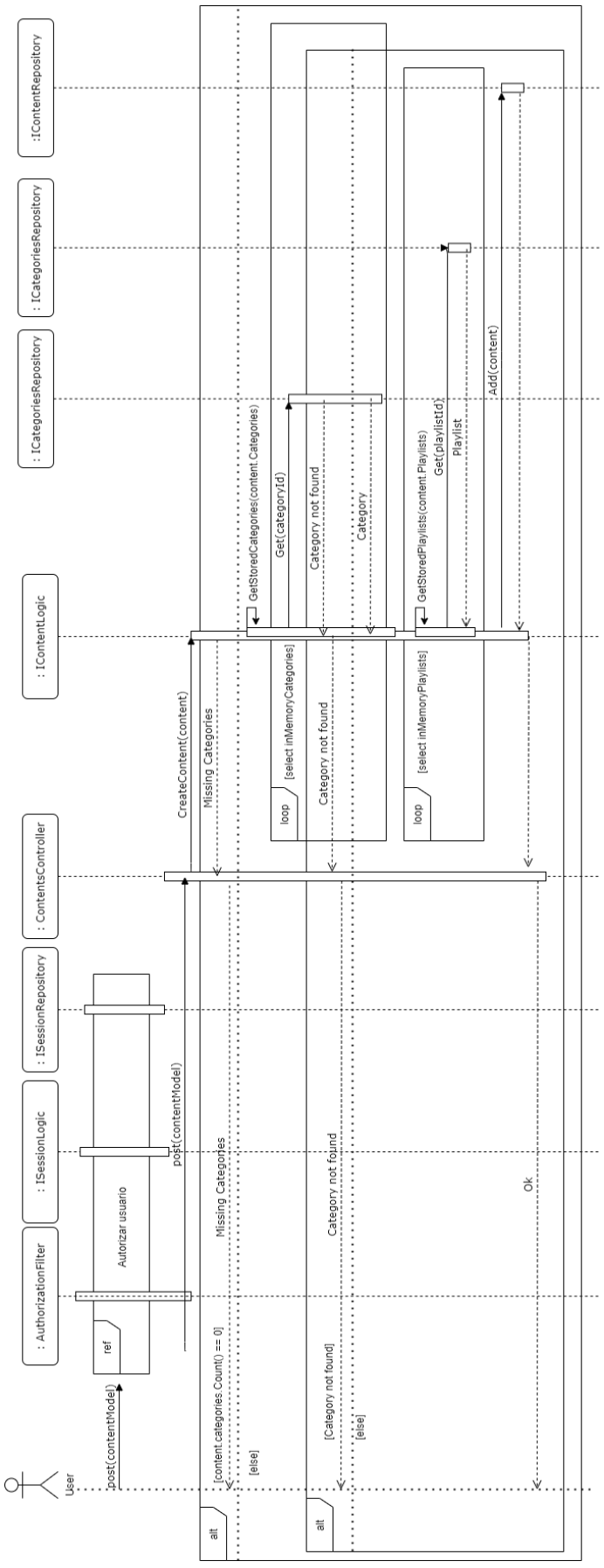


8. Anexo I. Diagramas de secuencia

Agendar consultas con un psicólogo



Alta contenido



Alta psicólogo