Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio Diseño Aplicaciones 2

Descripcion del Diseño

Luis Sempolis 185664

Entregado como requisito de la materia Diseño Aplicaciones 2

6 de mayo de 2021

Declaraciones de autoría

Nosotros, Luis Sempolis y Martin Vergara ; declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

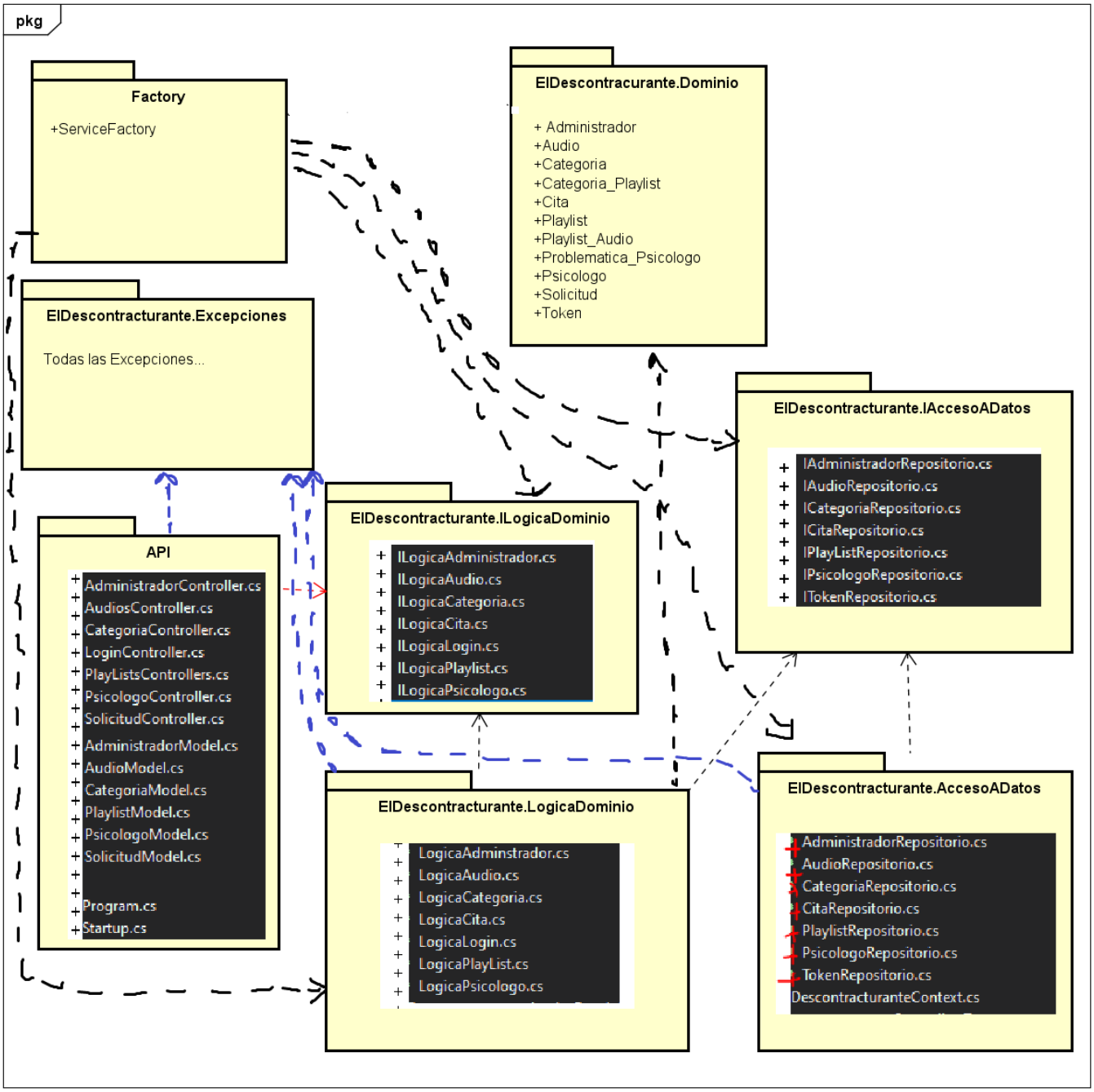
- La obra fue producida en su totalidad mientras realizábamos el primer obligatorio de Probabilidad y Estadística;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

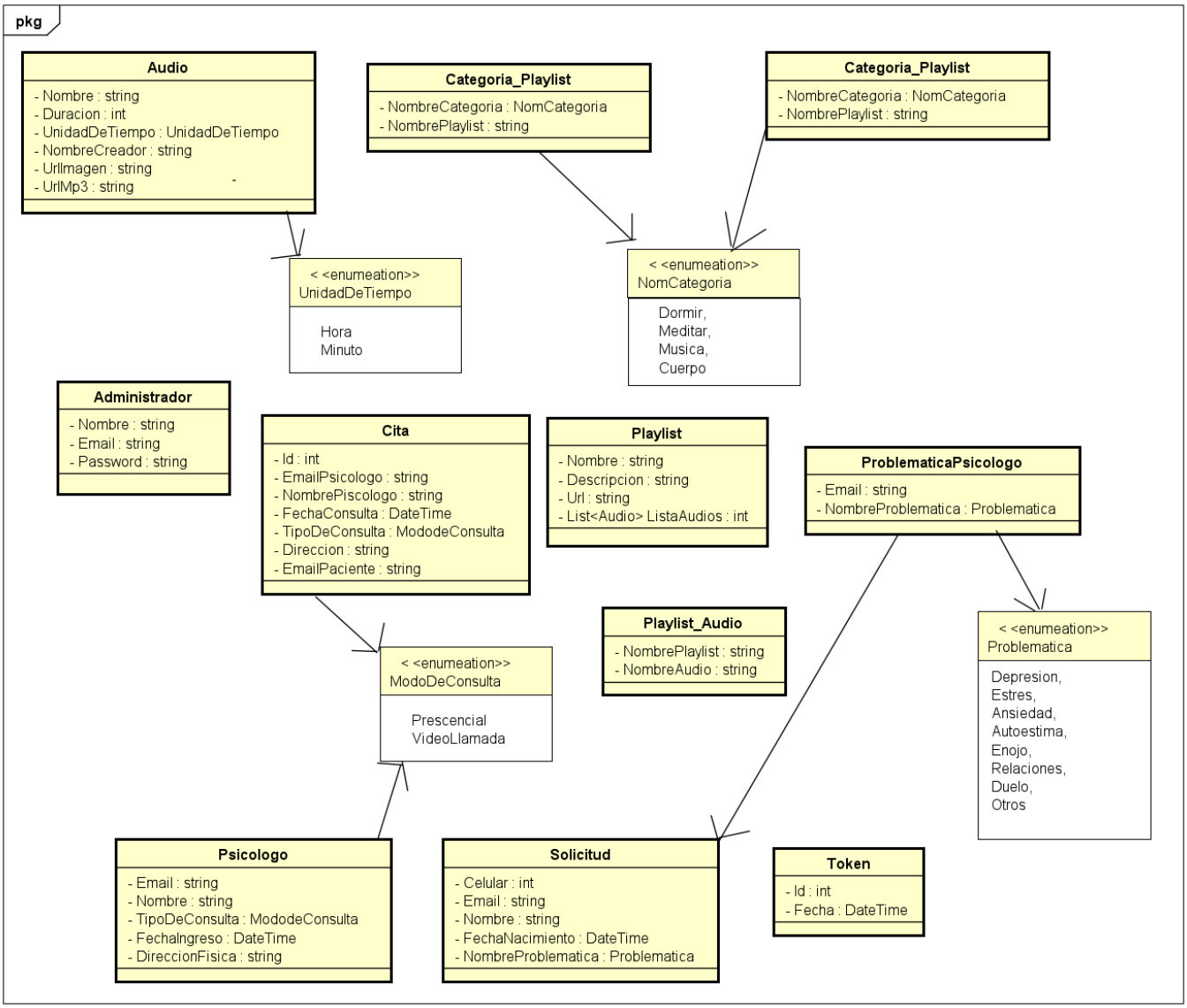
0.1. Descripción General Del Trabajo

Nuestra solución es básicamente el backend de un Sistema de gestión de problemáticas psicológicas. Brindamos un conjunto de definiciones de operaciones mediante una API REST que creemos son suficientes para solventar la problemática planteada.
Cabe recalcar que cualquier tipo de cliente que pueda utilizar HTTP REQUEST podrá utilizar nuestra aplicación haciendo de esta multi plataforma es decir no depende de una implementación en particular.
Creamos un mecanismo de autenticación mediante Tokens por lo cual podemos dar el privilegio de ciertas a operaciones a un grupo exclusivo de clientes, estos deben demostrar que cumplen con dicho privilegio.
Creemos haber implementado todas las funcionalidades planteadas por el cual a lo largo del documento se explicara cuales son las definiciones que nos referimos y como es el acceso a nuestra aplicación Backend.

0.2. Diagrama De Paquetes

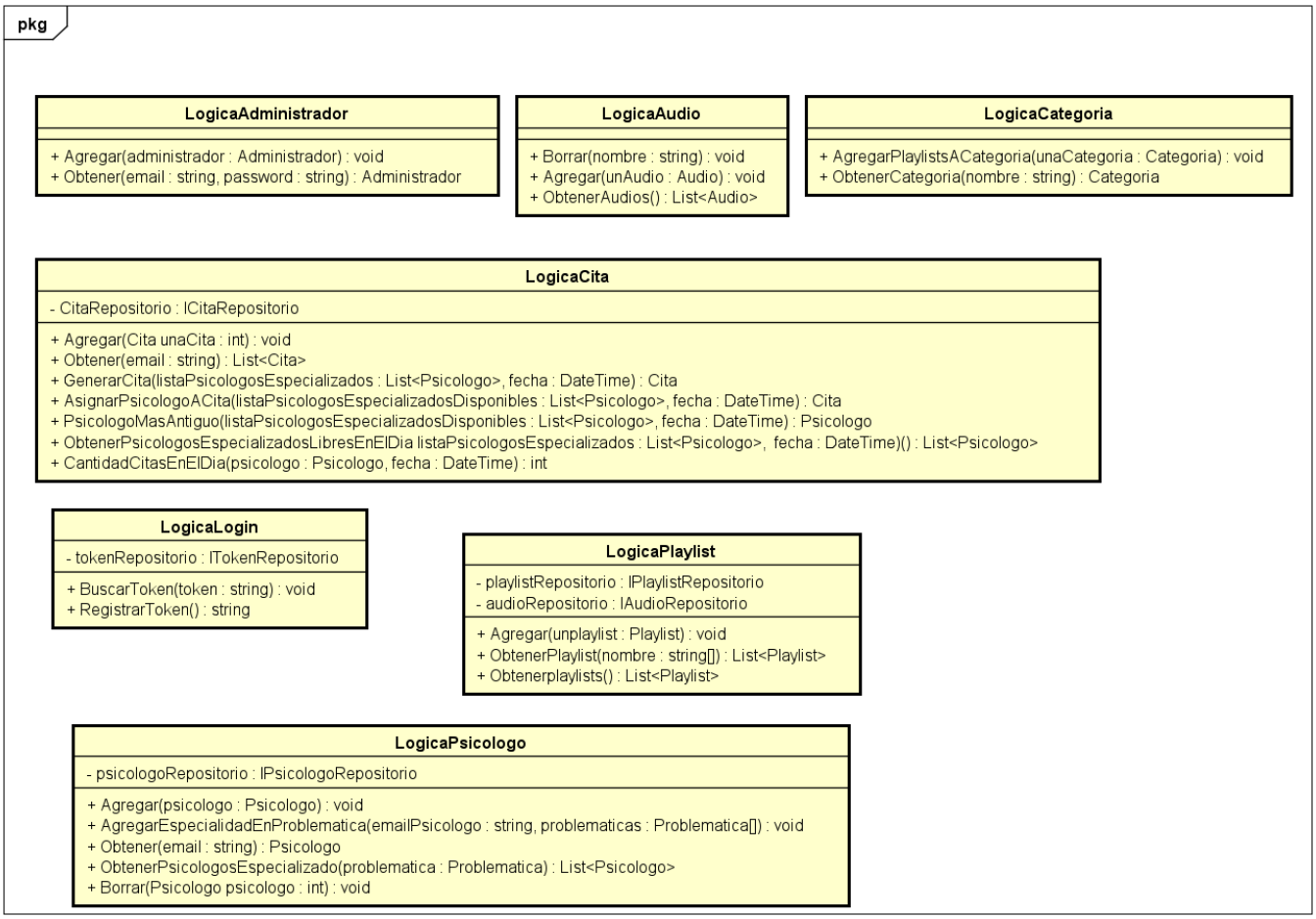


0.2.1. Diagrama De Clases Dominio



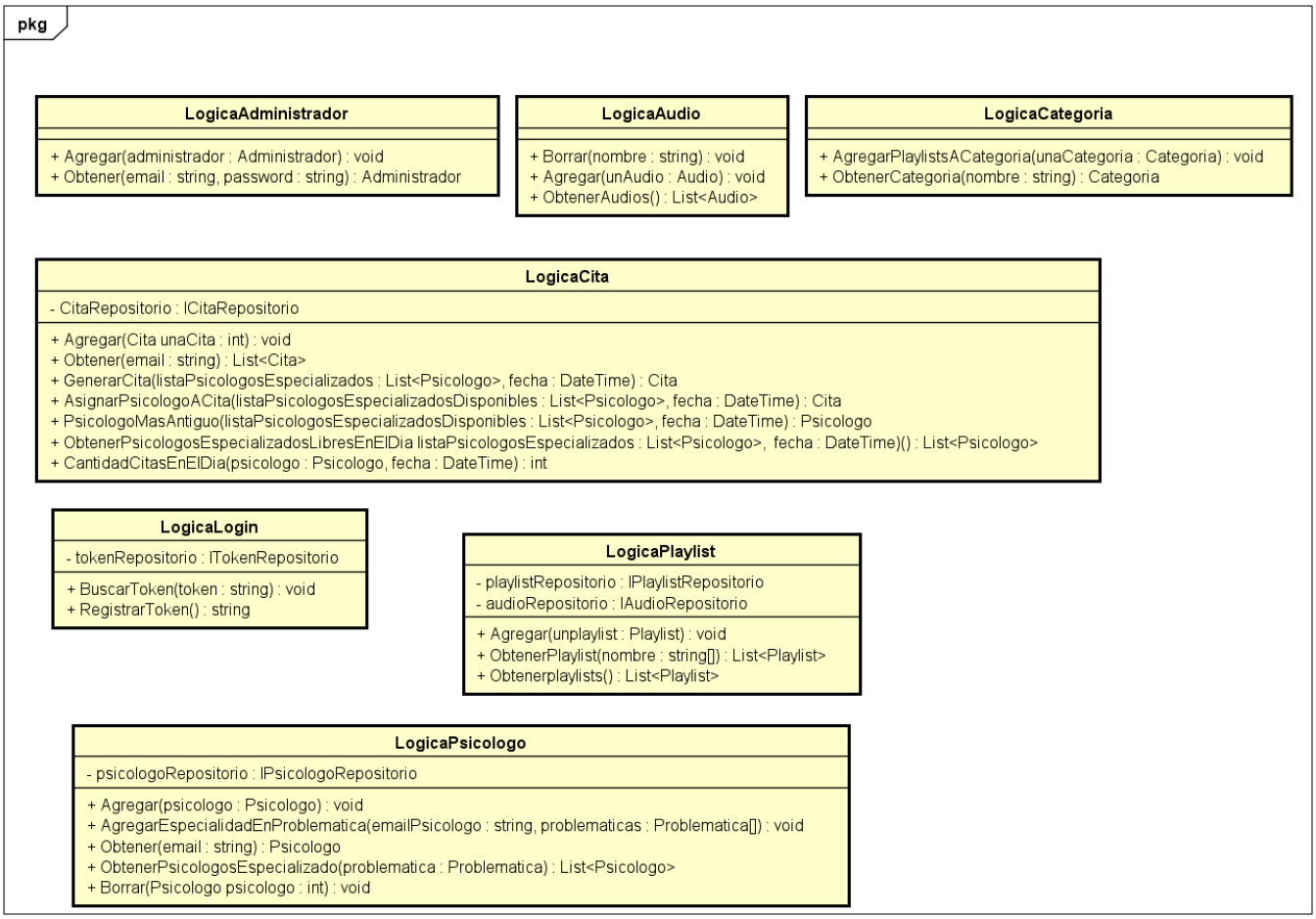
Cometido: Proveer una definición en común de las entidades que serán los objetos que interactuaran en nuestro sistema. Gracias a este paquete cuando nos referimos a un Objeto propio de nuestro Problema por ej. Audio sabremos que este se trata de una combinación especifica de atributos de tipos reconocidos previamente por el lenguaje. Int, String. List¡t¿o de otros objetos del Problema. Pero en fin conocemos estas combinaciones y mapeamos la problemática por la cual el Software fue creado en nuestro Proyecto.

0.2.2. Diagrama De Clases Lógica Dominio



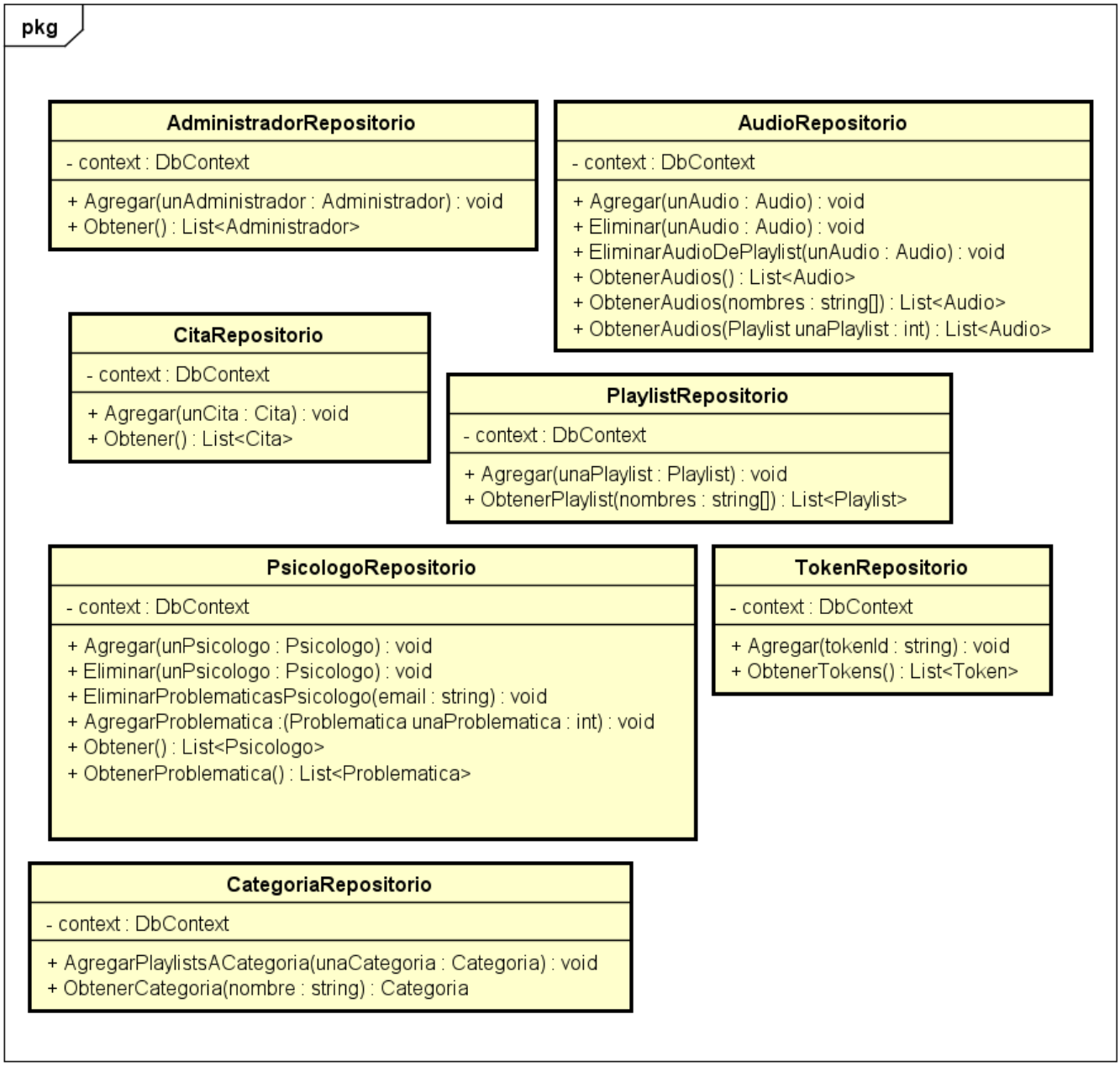
Brindar una implementación concreta para satisfacer los métodos que son necesarios para poder satisfacer nuestra problemática.

0.2.3. Diagrama De Clases Interfaz Lógica Dominio



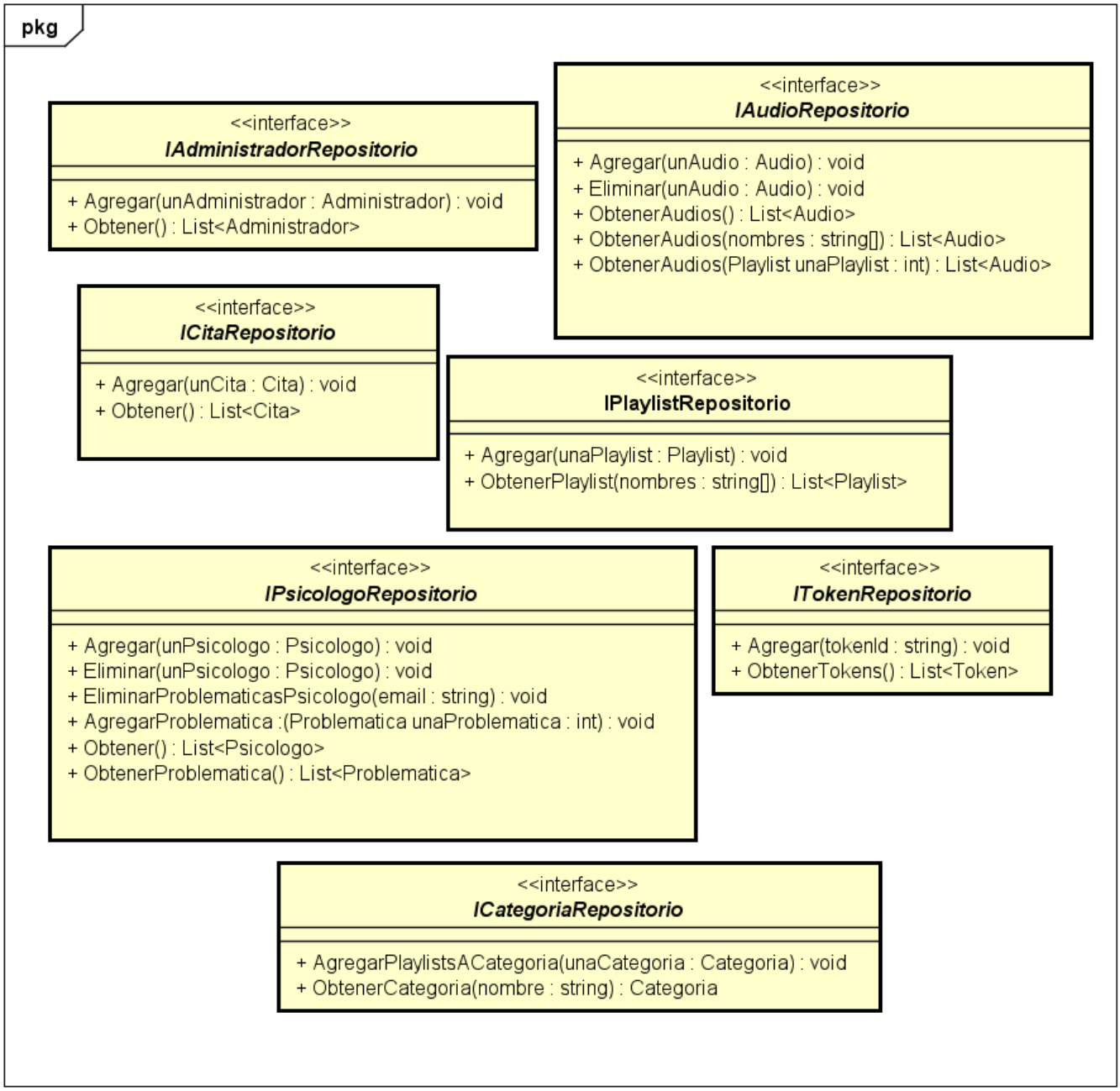
Cometido: Especifica qué se debe hacer pero no su implementación y de esta forma nos ayuda para comunicar objetos no relacionados entre si. Hace que la dependencias se realizan a "lo que se debe hacerz no a detalles de implementación. Por ejemplo al poseer la interfaz de la logica de Audio no dependemos de que repositorio utilizamos para almacenar los audios , ni que algoritmo de ordenación utilizamos para devolverlos,Sabemos que nuestra dependencia sera hacia una .ᵉˢpecificación de que debe hacer (agregar,obtener)z cualquier implementación que se de va a servir , incluso haciéndolas a estas intercambiables.

0.2.4. Diagrama De Clases Acceso A Datos



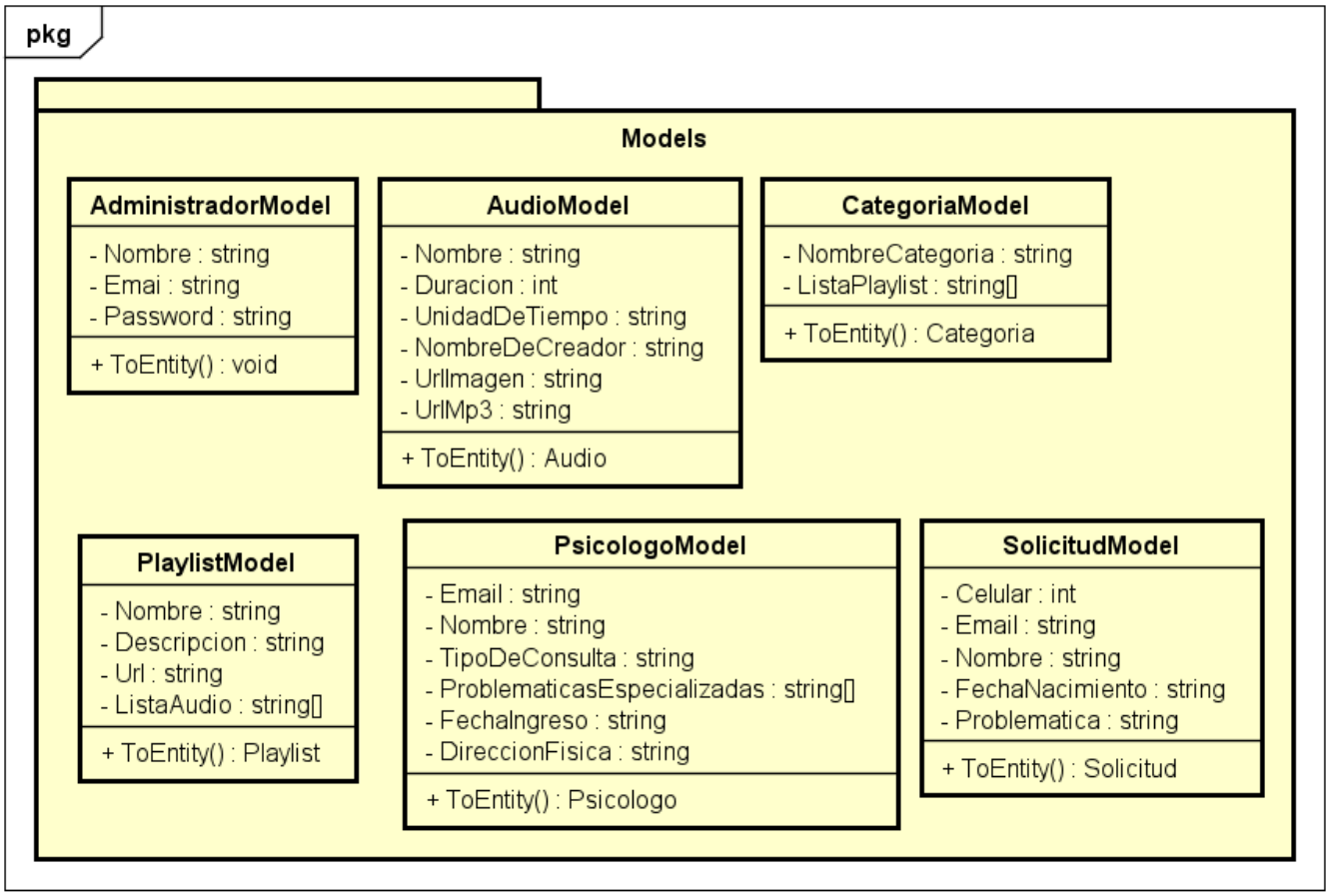
Cometido: Brindar una implementación concreta para satisfacer los métodos que son necesarios para poder satisfacer nuestra problemática del acceso a los datos y como deben ser devueltos estos. Aquí se manejaran métodos que son Concretos de cada motor de Base de Datos por lo que este paquete posiblemente no pueda ser reutilizado en caso de cambiar el motor de base de datos..

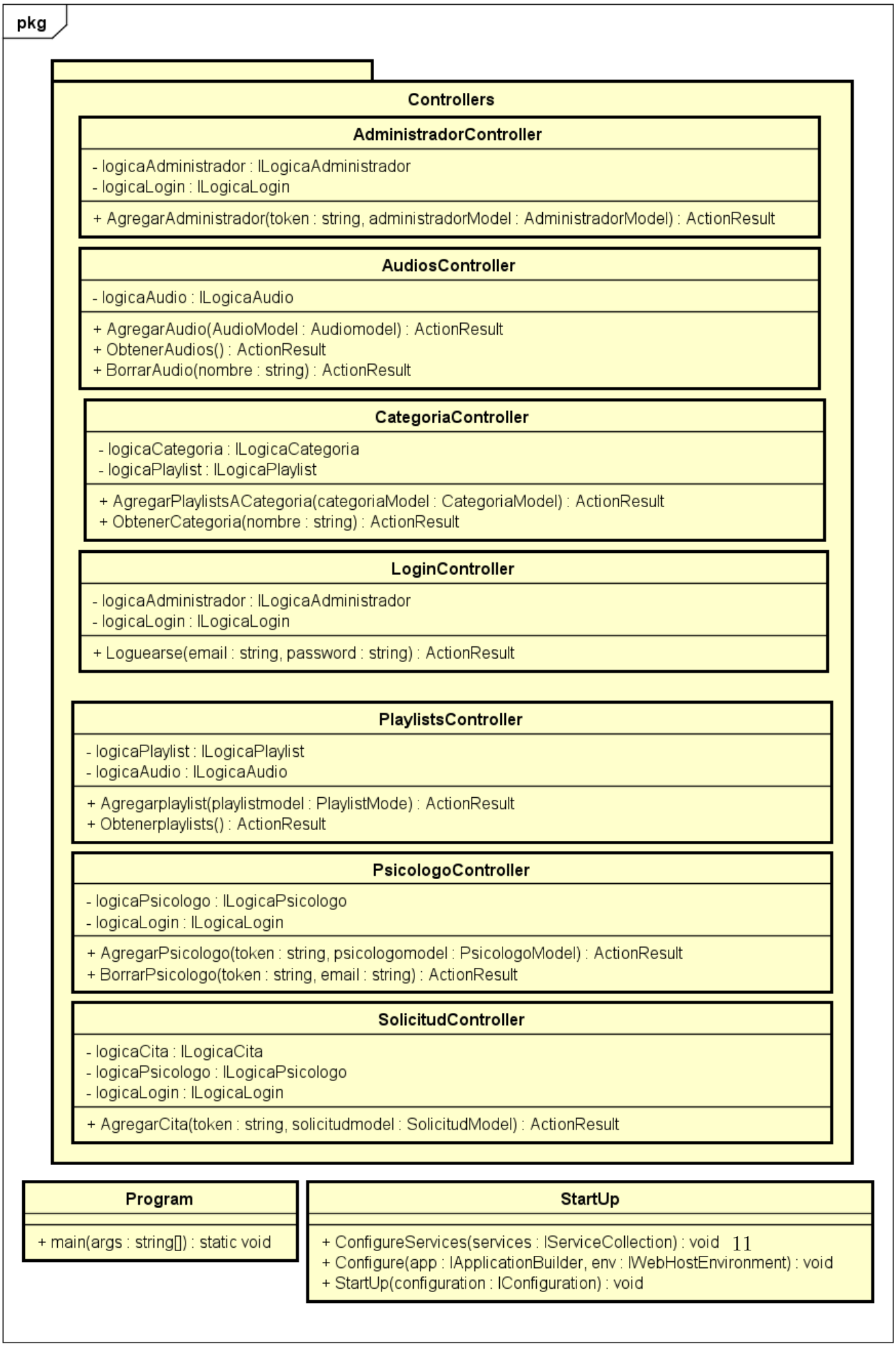
0.2.5. Diagrama De Clases Interfaz Acceso A Datos



Especifica qué se debe hacer para que una implementación concreta con un Motor de base de datos cualesquiera sea intercambiable con nuestro sistema. Si logra cumplir con la especificación es valido.

0.2.6. Diagrama De Clases WebAPI




pkg
Controllers
AdministradorController
- logicaAdministrador : ILogicaAdministrador
- logicaLogin : ILogicaLogin
+ AgregarAdministrador(token : string, administradorModel : AdministradorModel) : ActionResult
AudiosController
- logicaAudio : ILogicaAudio
+ AgregarAudio(AudioModel : Audiomodel) : ActionResult
+ ObtenerAudios() : ActionResult
+ BorrarAudio(nombre : string) : ActionResult
CategoriaController
- logicaCategoria : ILogicaCategoria
- logicaPlaylist : ILogicaPlaylist
+ AgregarPlaylistsACategoria(categoriaModel : CategoriaModel) : ActionResult
+ ObtenerCategoria(nombre : string) : ActionResult
LoginController
- logicaAdministrador : ILogicaAdministrador
- logicaLogin : ILogicaLogin
+ Loguearse(email : string, password : string) : ActionResult
PlaylistsController
- logicaPlaylist : ILogicaPlaylist
- logicaAudio : ILogicaAudio
+ Agregarplaylist(playlistmodel : PlaylistMode) : ActionResult
+ Obtenerplaylists() : ActionResult
PsicologoController
- logicaPsicologo : ILogicaPsicologo
- logicaLogin : ILogicaLogin
+ AgregarPsicologo(token : string, psicologomodel : PsicologoModel) : ActionResult
+ BorrarPsicologo(token : string, email : string) : ActionResult
SolicitudController
- logicaCita : ILogicaCita
- logicaPsicologo : ILogicaPsicologo
- logicaLogin : ILogicaLogin
+ AgregarCita(token : string, solicitudmodel : SolicitudModel) : ActionResult
Program
+ main(args : string[]) : static void
StartUp
+ ConfigureServices(services : IServiceCollection) : void
+ Configure(app : IApplicationBuilder, env : IWebHostEnvironment) : void
+ StartUp(configuration : IConfiguration) : void

Es el punto de entrada de la aplicación , publicara las formas de acceso a la información mediante la cual un cliente se comunicara para obtenerla o modificarla .

0.2.7. Factory

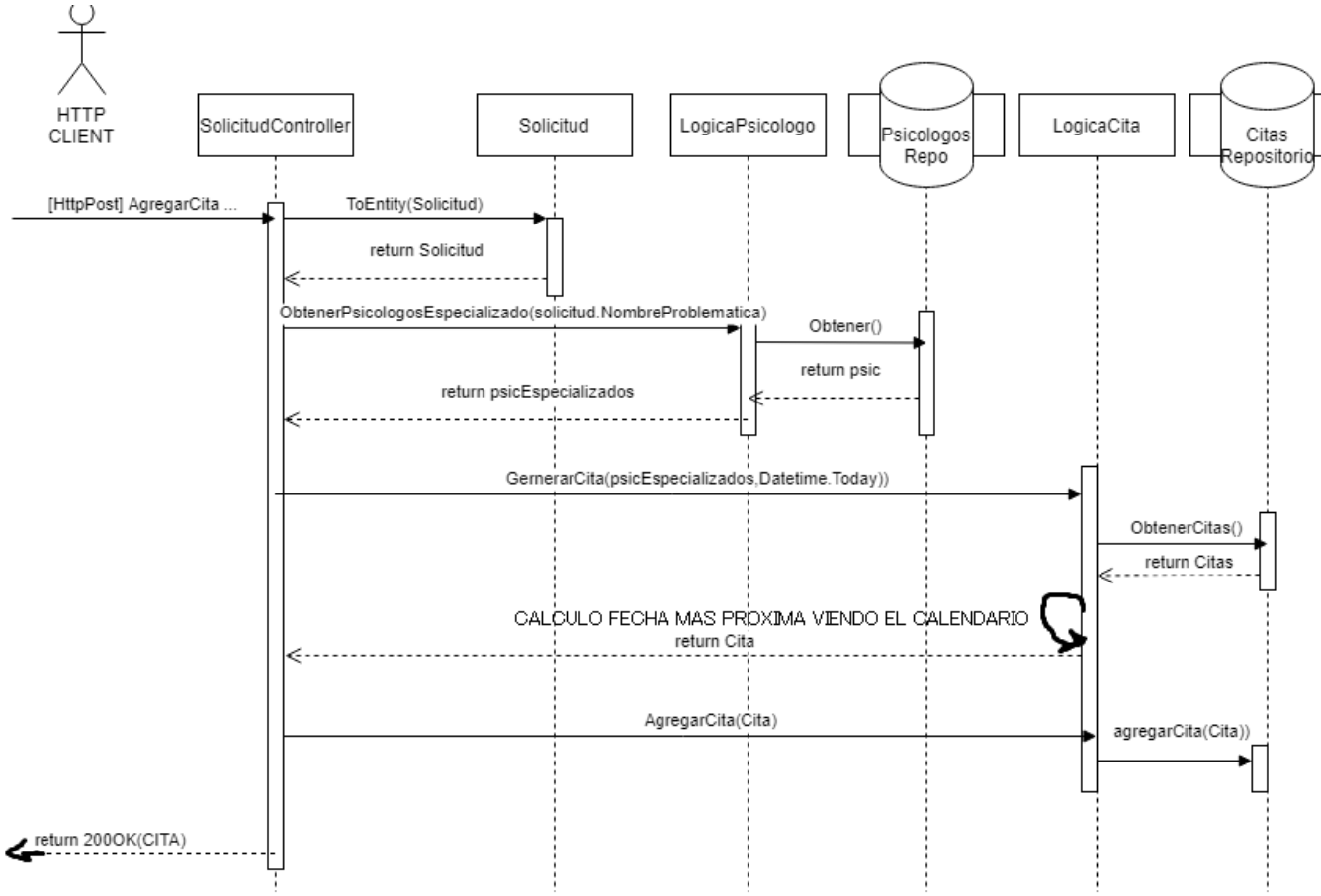
Posee solo la clase Service Factory no lo creímos necesario.
Encargado de inyectar las dependencias para romper el acoplamiento de implementación de nuestro sistema, el cual sucede entre la dependencia de dos clases concretas en cambio gracias a este paquete podemos depender de abstracciones.

0.2.8. Excepciones

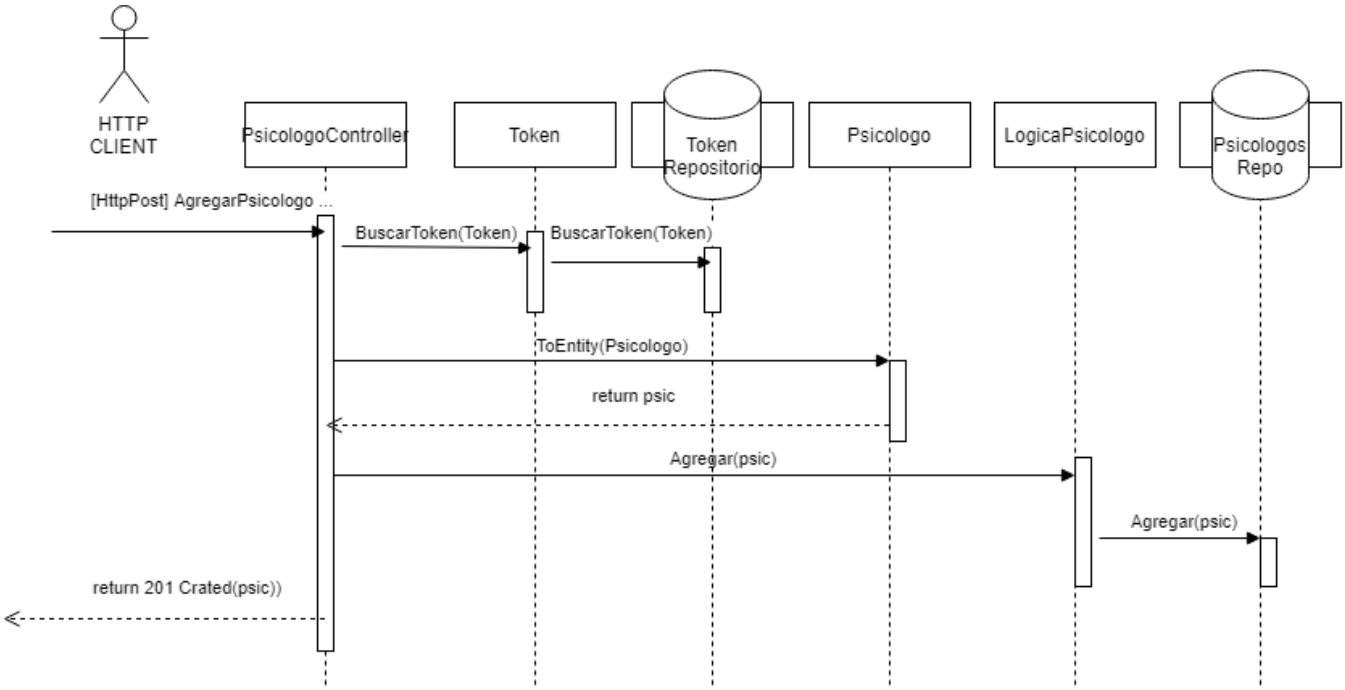
Contiene excepciones propias de nuestro sistema para el mejor reconomiento de los errores.

0.3. Diagrama Secuencias Funcionalidades Claves

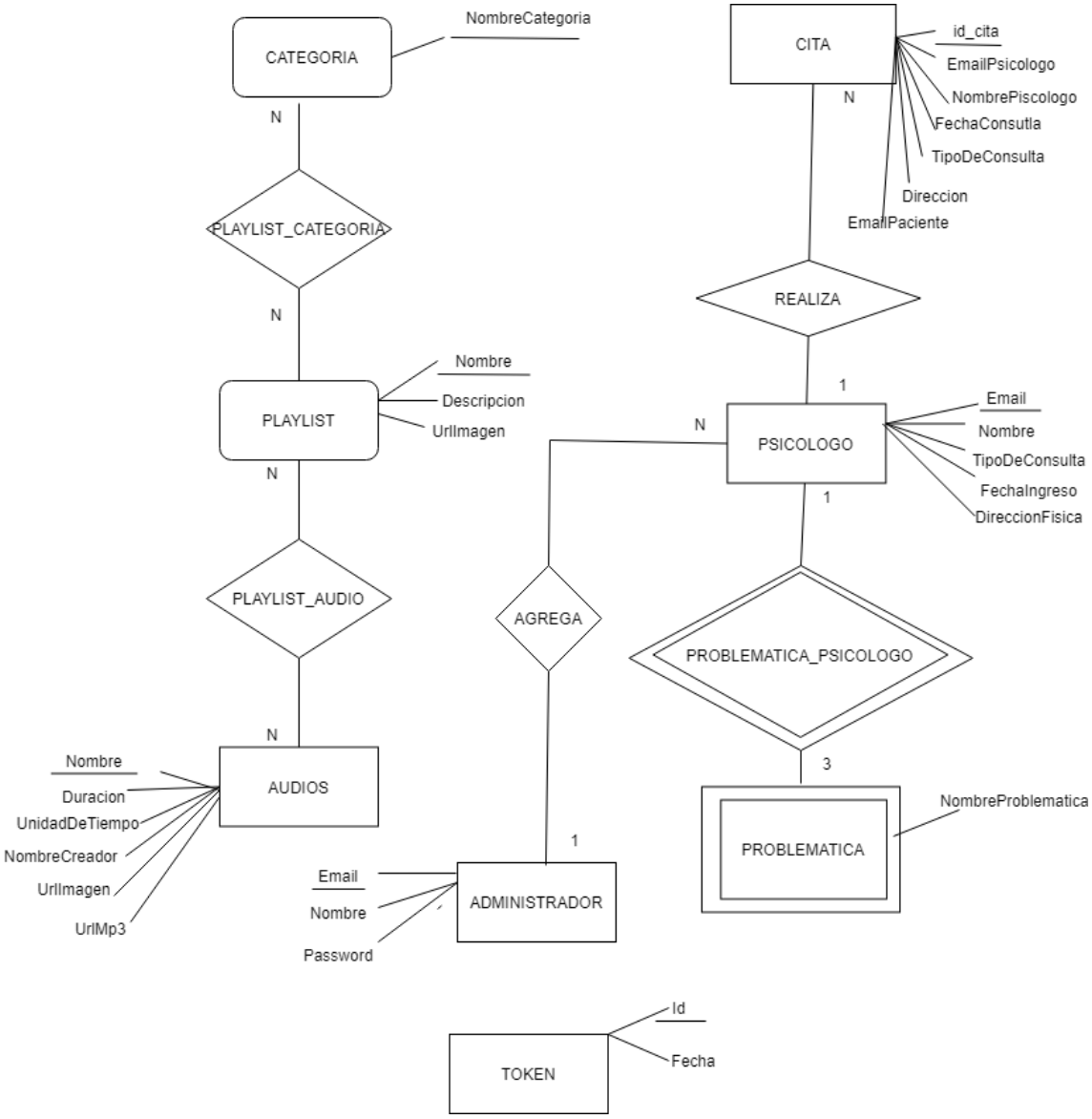
0.3.1. Agregar Cita



0.3.2. Agregar Psicólogo



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



0.4.1. Justificación del Diseño

0.4.2. Inyección de Dependencias

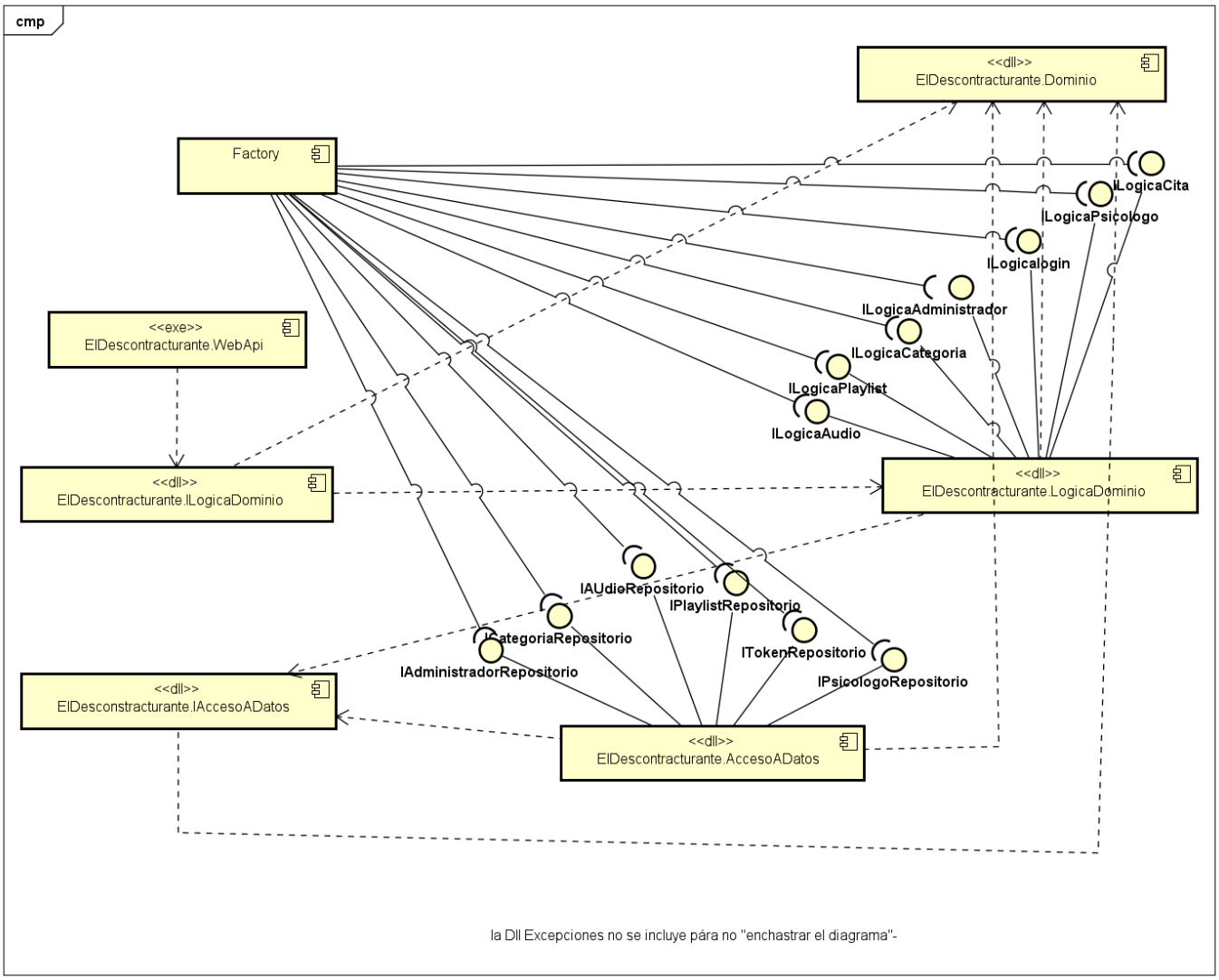
Como vimos en el diagrama de paquetes no tenemos dependencias de Implementación , es decir siempre dependemos de interfaces y no de un paquete en concreto para esto usamos el patrón de Diseño Inyección de Dependencias. Y de esta forma nuestros módulos son flexibles a utilizar otras implementaciones como puede ser el proveedor de base de datos.

Manejo de Excepciones

Se utilizaron excepciones propias para que a la hora de que se genere un error en el código tener idea como levantarnos del mismo, mapear excepciones propias del lenguaje a nuestras propias excepciones nos brinda una reacción al fallo mucha mas alta que si no la tuviéramos.

Mecanismos de Acceso a Datos Para esto utilizamos una interfaz que define las operaciones que cualquier motor de base de datos debe implementar. Haciendo esto nuestra lógica se basa en el uso de estas definiciones y no de la implementación en particular que le damos. Cualquier motor de base de datos que pueda implementar dicha lógica podrá ser intercambiable en nuestro sistema.

0.5. Diagrama de Componentes



Bibliografía

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, eight ed. McGraw-Hill, 2014.

[2] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.