



Diseño de aplicaciones 2

Obligatorio 2

<https://github.com/ORT-DA2/DA2-223143-205218>

Autores:

Francisco Rosello 223143

Maximiliano Pascale 205218

Docentes:

Ignacio Valle

Nicolás Fierro

Nicolás Blanco

2022

Declaración de autoría

Nosotros, Maximiliano Pascale y Francisco Rosello, 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 cursamos Diseño de aplicaciones 2.
- 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 ayudantías recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y que 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:

| | |
|--|----|
| Descripción general del trabajo | 5 |
| Errores conocidos | 5 |
| Funcionalidades no implementadas | 5 |
| Diagrama general de paquetes | 6 |
| Descripción de responsabilidades | 6 |
| Diagrama de clases asociado por paquete | 8 |
| ArenaGestor.API | 8 |
| ArenaGestor.APIContracts | 9 |
| ArenaGestor.Domain | 10 |
| ArenaGestor.BusinessFactory | 10 |
| ArenaGestor.BusiessHelpers | 10 |
| ArenaGestor.Business | 11 |
| ArenaGestor.BusinessInterface | 12 |
| ArenaGestor.DataAccess | 13 |
| ArenaGestor.DataAccessFactory | 13 |
| ArenaGestor.DataAccessInterface | 14 |
| ArenaGestor.Extensions | 14 |
| Descripción de jerarquías | 15 |
| Modelo de tablas | 15 |
| Diagrama de despliegue | 16 |
| Diagramas de interacción | 16 |
| Login | 16 |
| Export | 17 |
| Diagrama de implementación | 17 |
| Extensibilidad del sistema | 18 |
| Justificación y explicación del diseño | 19 |
| Seguridad | 19 |
| Excepciones | 20 |
| Acceso a datos | 20 |
| Validaciones | 20 |
| Patrones y principios de diseño utilizados | 21 |
| Patrón Strategy | 21 |

| | |
|---------------------------------|----|
| Patrón Factory | 21 |
| Patrón Facade | 22 |
| Patrones GRASPs: | 22 |
| Experto | 22 |
| Alta Cohesión | 23 |
| Controlador | 23 |
| Polimorfismo | 23 |
| Calidad del diseño | 24 |
| Anexo | 29 |
| Datos de prueba | 29 |
| Datos globales de la aplicación | 29 |
| Usuarios y sus roles | 30 |
| Artistas, solistas y bandas. | 32 |
| Conciertos | 34 |
| Tickets | 36 |
| Documentación de la API | 37 |
| Informe de cobertura | 38 |
| Test | 39 |
| Manual de instalación | 40 |
| Códigos de error | 42 |

Descripción general del trabajo

La solución entregada permite gestionar la venta de tickets para los eventos del Antel Arena. La misma permite registrar las bandas (Solistas o conjuntos musicales), el género de cada uno, sus conciertos y la venta de tickets para cada concierto. Para cada concierto permite saber quién compró cada ticket así como los usuarios pueden ver todos los conciertos a los cuales asistieron.

También permite a los funcionarios del Antel Arena vender los tickets y marcar los tickets como “Usados” cuando un usuario ingresa al recinto para ver un concierto.

En esta entrega se provee de una Web Api y la correspondiente interfaz web para interactuar con el sistema a cualquier usuario.

Errores conocidos

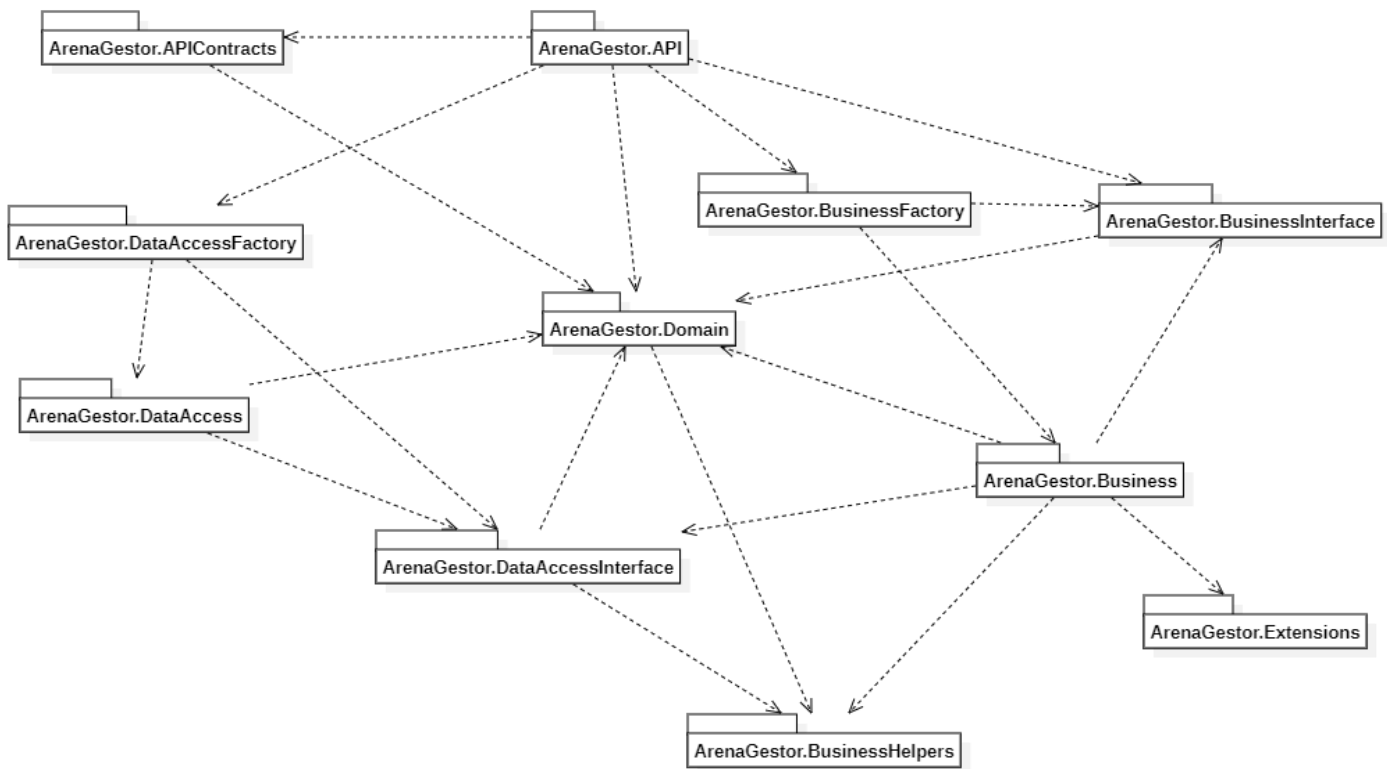
Hemos realizado las correcciones de la primera entrega donde por falta de tiempo y experiencia manejando la tecnología Entity Framework el proyecto contenía los siguientes problemas que **ya fueron solucionados**.

- Al modificar una banda, no se modifican los artistas que pertenecen a esa banda (No se puede agregarle o quitarle artistas a una banda)
- Al modificar un usuario, no se agregan nuevos roles ni se quitan.

Funcionalidades no implementadas

Se han desarrollado todos los requerimientos solicitados, así como las definiciones establecidas en el foro de la materia.

Diagrama general de paquetes



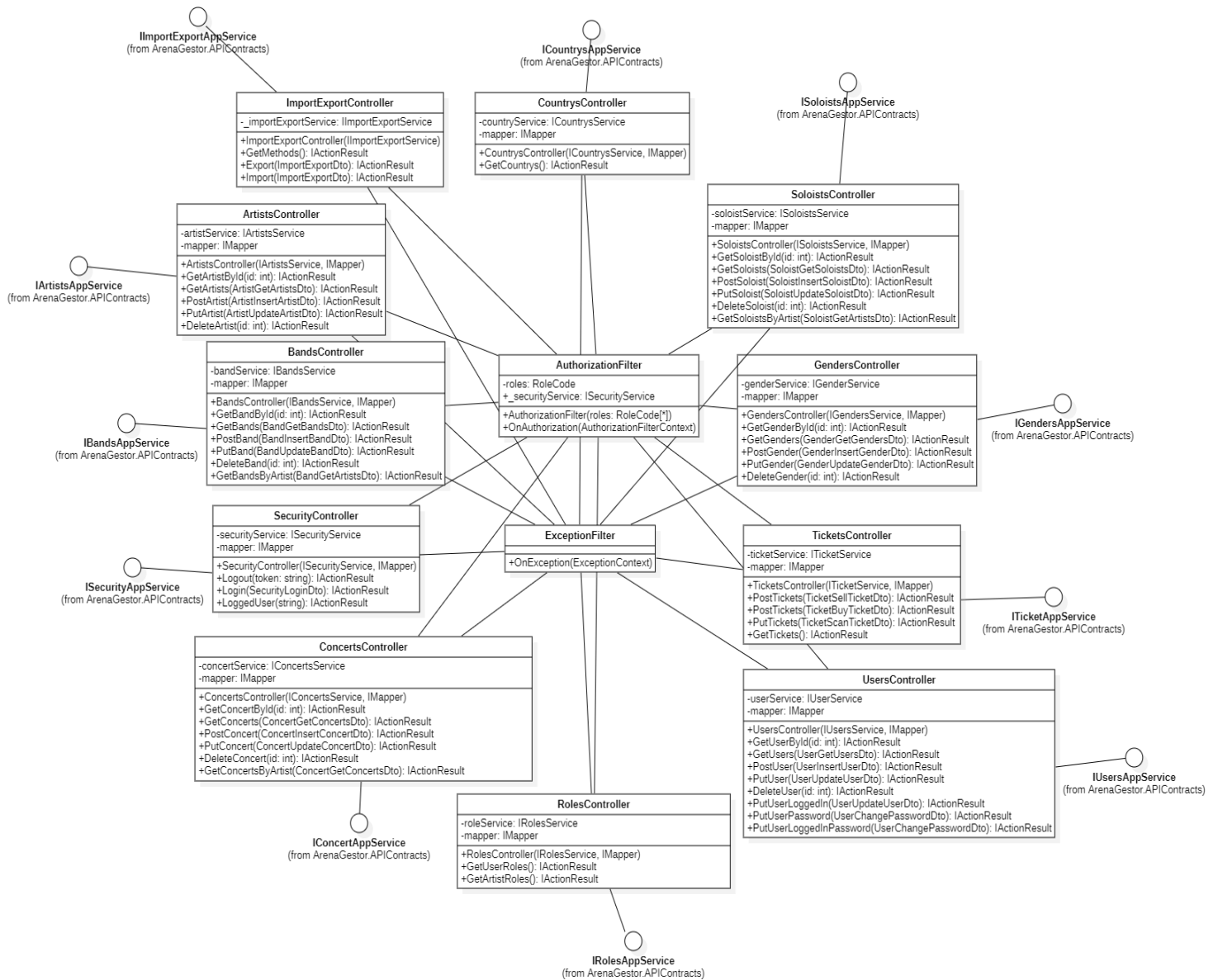
Descripción de responsabilidades

- **ArenaGestor.API**: Este paquete es el encargado de proveer a los usuarios de una Web API para obtener, crear, modificar, borrar los datos que mantiene la aplicación.
- **ArenaGestor.APIContracts**: Este paquete contiene todas las interfaces que deben cumplir los controladores y también contiene todas las clases “DTO” las cuales establecen los datos que se deben enviar o devolver en los request y de esta forma no exponer directamente las clases de dominio.
- **ArenaGestor.Domain**: Este paquete contiene todas las clases del dominio del proyecto las cuales son usadas en todos los otros paquetes de la solución.
- **ArenaGestor.Business**: Este paquete encapsula toda la lógica de negocios de la aplicación. La misma es la encargada de validar los datos que se ingresan al sistema y comunicarse con los otros paquetes para, por ejemplo, persistir los datos.

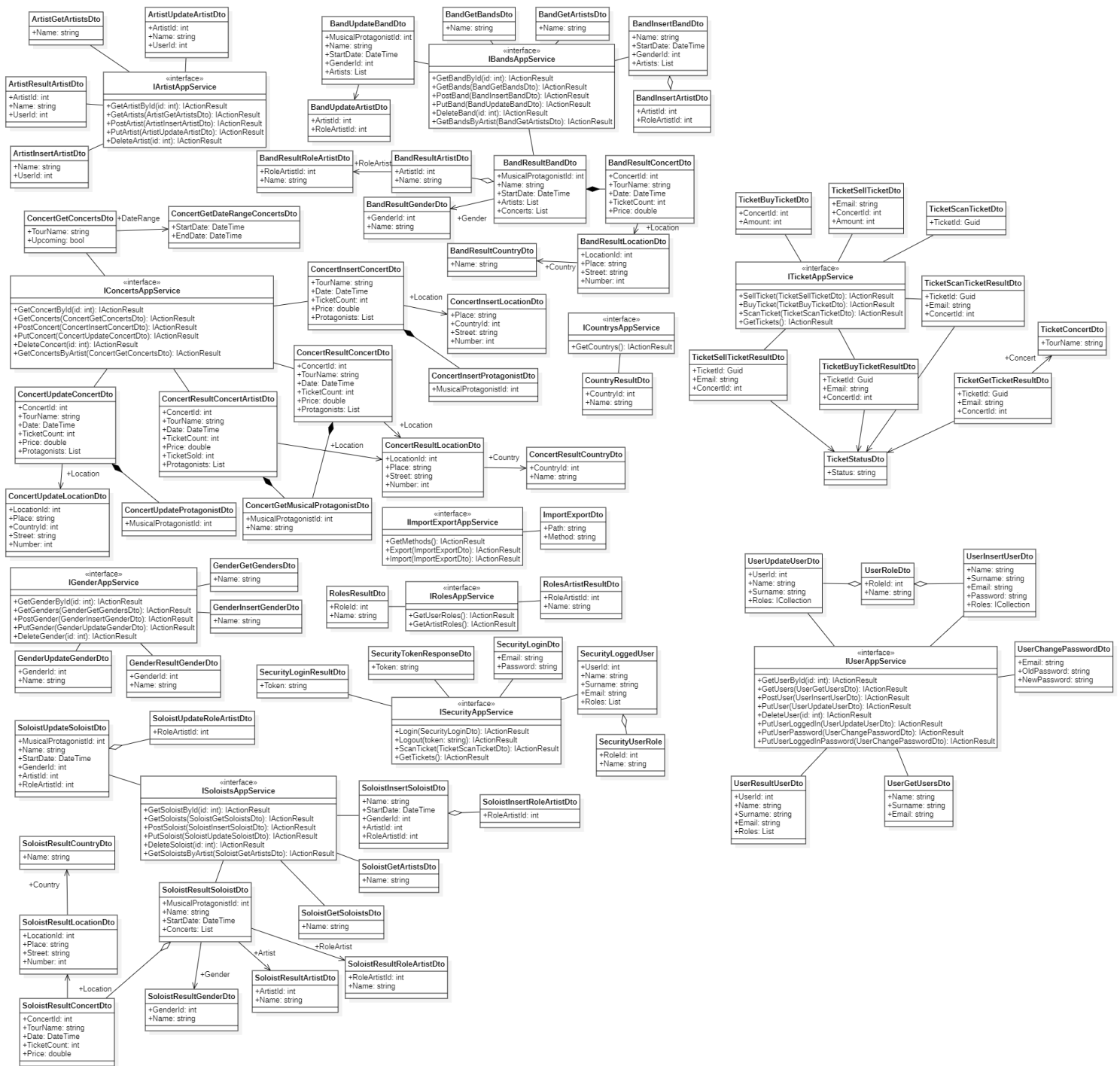
- ArenaGestor.BusinessInterface: Provee todas las interfaces de servicios que debe cumplir el paquete Business.
- ArenaGestor.BusiessHelpers: Provee varias funciones de ayuda para validar datos, por ejemplo que un email sea correcto.
- ArenaGestor.BusinessFactory: Este paquete es el encargado de registrar los servicios para inyectar dependencias del paquete Business. Este paquete provee una forma sencilla de cambiar cuál es la implementación del servicio simplemente modificando una línea de código sin tener que modificar la implementación de otras clases.
- ArenaGestor.DataAccess: Este paquete es el encargado de persistir los datos. Contiene todos los servicios para que los otros paquetes puedan obtener, crear, modificar o borrar registros.
- ArenaGestor.DataAccessInterface: Provee todas las interfaces de servicios que debe cumplir el paquete DataAccess.
- ArenaGestor.DataAccessFactory: Este paquete es el encargado de registrar los servicios para inyectar dependencias del paquete DataAccess.
- ArenaGestor.Extensions: Este paquete es el encargado de definir las interfaces para que terceros puedan desarrollar la extensión al sistema. Este cuenta con lo referido a la importación/exportación de conciertos.

Diagrama de clases asociado por paquete

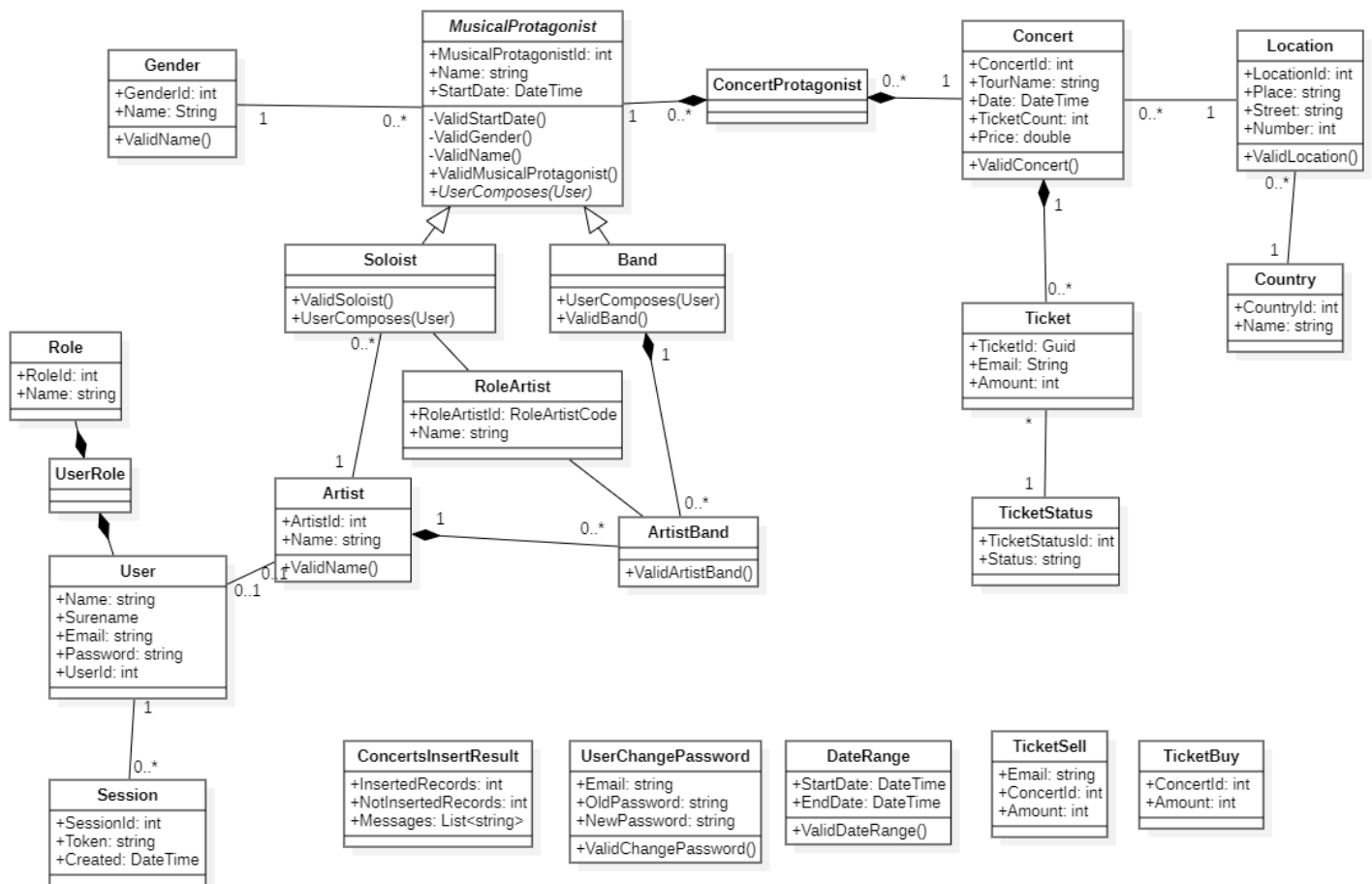
ArenaGestor.API



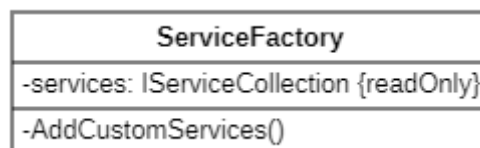
ArenaGestor.APIContracts



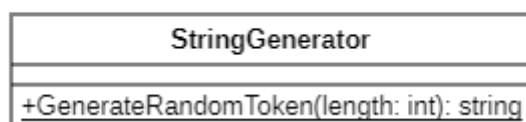
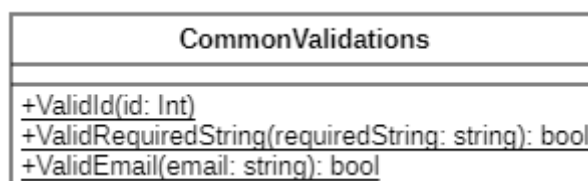
ArenaGestor.Domain



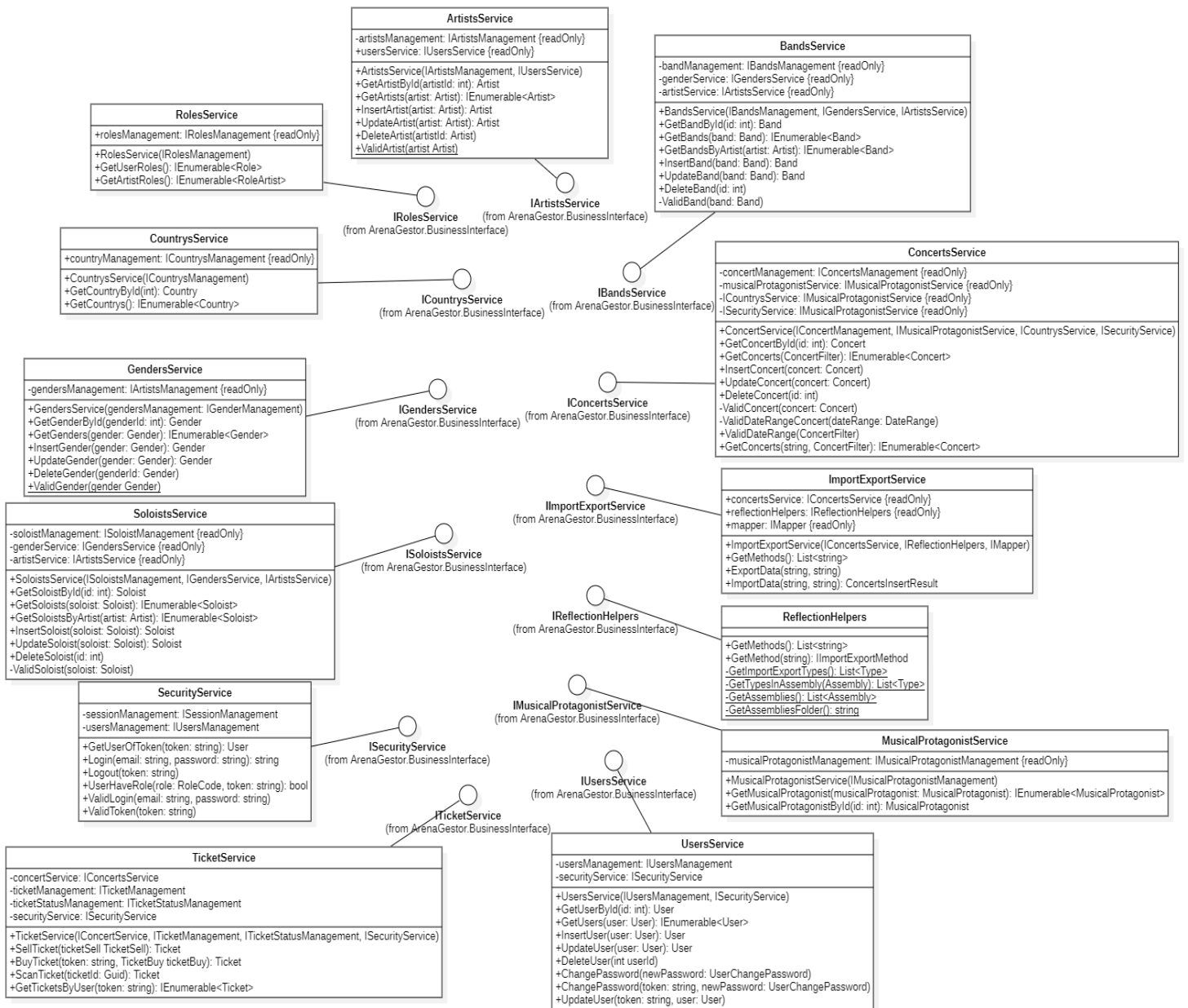
ArenaGestor.BusinessFactory



ArenaGestor.BusiessHelpers



ArenaGestor.Business



ArenaGestor.BusinessInterface

| «interface» IArtistsService |
|---|
| +GetArtistById(artistId: int): Artist +GetArtists(artist: Artist): IEnumerable<Artist> +InsertArtist(artist Artist): Artist +UpdateArtist(artist Artist): Artist +DeleteArtist(artistId: int) |

| «interface» IGendersService |
|--|
| +GetGenderById(genderId: int): Gender +GetGenders(gender: Genders): IEnumerable<Gender> +InsertGender(gender Gender): Gender +UpdateGender(gender Gender): Gender +DeleteGender(genderId: int) |

| «interface» IConcertsService |
|---|
| +GetConcertById(concertId: int): Concert +GetConcerts(concert: Concert): IEnumerable<Concert> +InsertConcert(concert Concert): Concert +UpdateConcert(concert Concert): Concert +DeleteConcert(concertId: int) +GetConcerts(string, ConcertFilter): IEnumerable<Concert> +InsertConcerts(List<Concert>): ConcertsInsertResult |

| «interface» ITicketService |
|---|
| +SellTicket(ticketSell: TicketSell): Ticket +BuyTicket(token: string, ticketBuy: TicketBuy): Ticket +ScanTicket(ticketId: Guid): Ticket +GetTicketsByUser(): IEnumerable<Ticket> |

| «interface» IImportExportService |
|---|
| +GetMethods(): List<string> +ImportData(string, string): ConcertsInsertResult +ExportData(string, string) |

| «interface» ISecurityService |
|---|
| +GetUserOfToken(token: string): User +UserHaveRole(role: RoleCode, token: string): bool +Login(email: string, password: string): string +Logout(token: string) |

| «interface» IReflectionHelpers |
|---|
| +GetMethods(): IEnumerable<string> +GetMethod(string): IImportExportMethod |

| «interface» IBandsService |
|---|
| +GetBandById(bandId: int): Band +GetBands(band: Band): IEnumerable<Band> +InsertBand(band Band): Band +UpdateBand(band Band): Band +DeleteBand(bandId: int) +GetBandsByArtist(artist: Artist): IEnumerable<Band> |

| «interface» IMusicalProtagonistService |
|---|
| +GetMusicalProtagonistById(musicalProtagonistId: int): MusicalProtagonist +GetMusicalProtagonist(musicalProtagonist MusicalProtagonist): IEnumerable<MusicalProtagonist> |

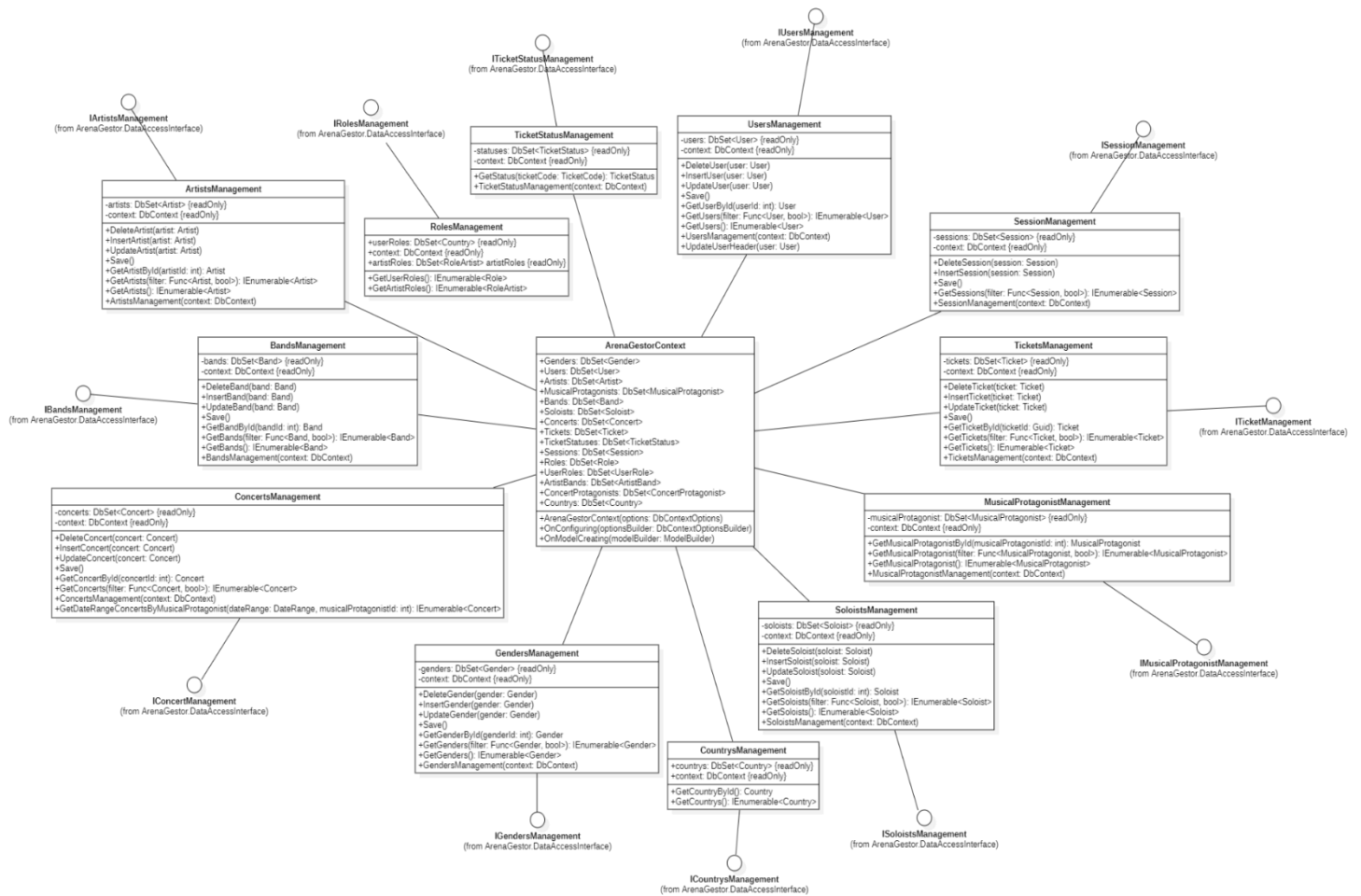
| «interface» ISoloistsService |
|--|
| +GetSoloistById(soloistId: int): Soloist +GetSoloists(soloist: Soloist): IEnumerable<Soloist> +InsertSoloist(soloist Soloist): Soloist +UpdateSoloist(soloist Soloist): Soloist +DeleteSoloist(soloistId: int) +GetSoloistsByArtist(artist: Artist): IEnumerable<Soloist> |

| «interface» IUsersService |
|---|
| +GetUserById(userId: int): User +GetUsers(user: User): IEnumerable<User> +InsertUser(user User): User +UpdateUser(user User): User +DeleteUser(userId: int) +ChangePassword(newPassword UserChangePassword) +ChangePassword(token: string, newPassword UserChangePassword) +UpdateUser(token: string, user User): User |

| «interface» ICountrysService |
|---|
| +GetCountryById(int): Country +GetCountrys(): IEnumerable<Country> |

| «interface» IRolesService |
|--|
| +GetUserRoles(): IEnumerable<Role> +GetArtistRoles(): IEnumerable<RoleArtist> |

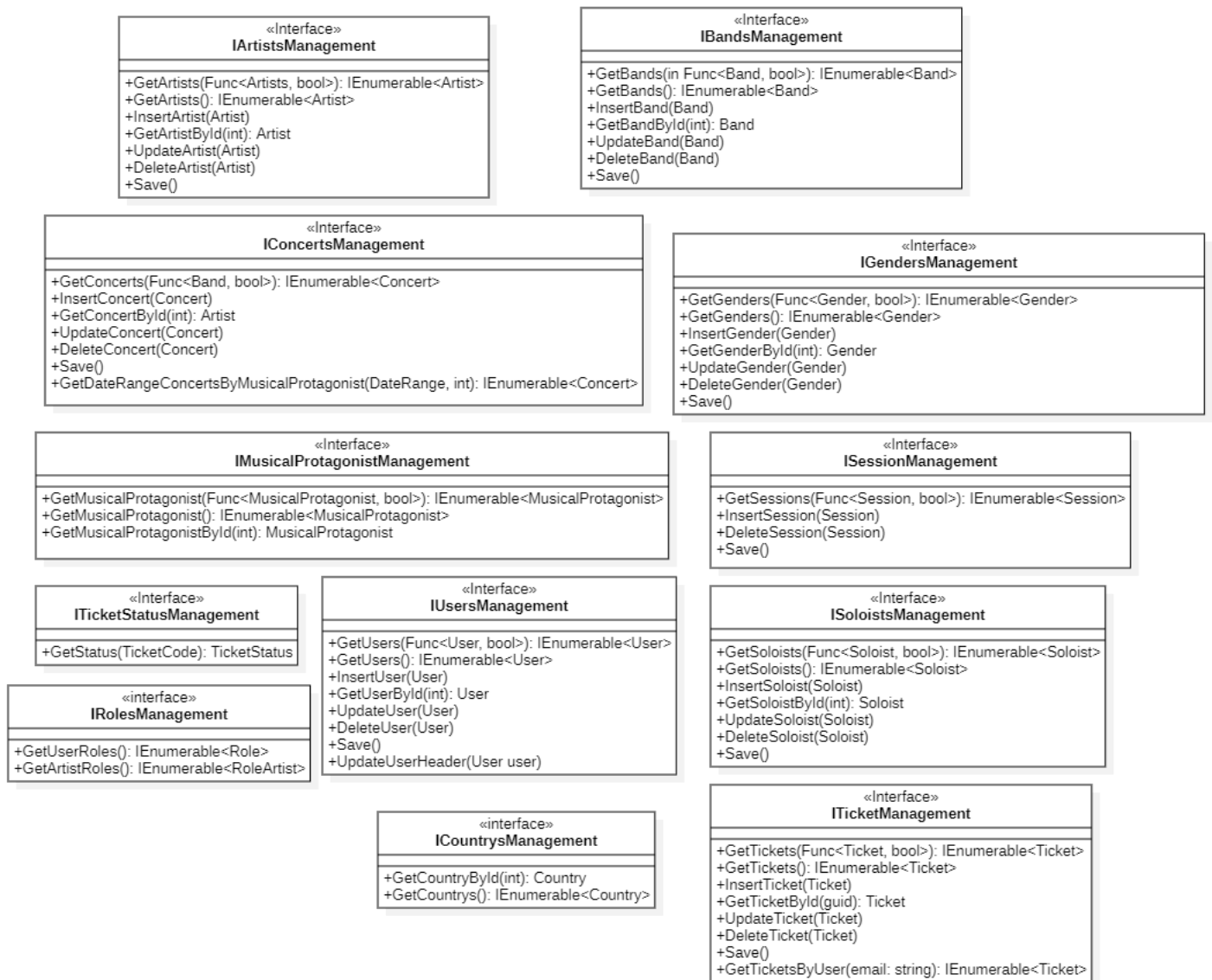
ArenaGestor.DataAccess



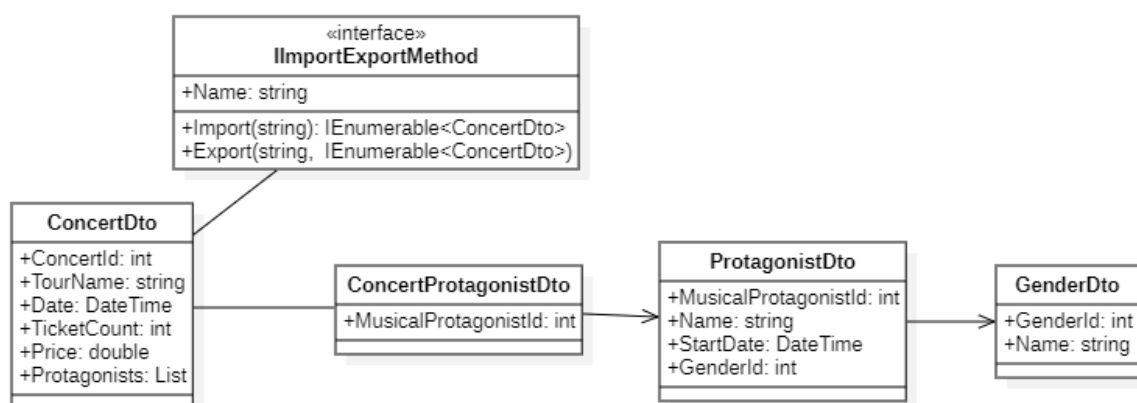
ArenaGestor.DataAccessFactory

| ManagementFactory |
|--|
| -service: IServiceCollection {readOnly} |
| +ManagementFactory(services: IServiceCollection) |
| -AddCustomManagement() |
| +AddDbContextService(connectionString: string) |

ArenaGestor.DataAccessInterface



ArenaGestor.Extensions

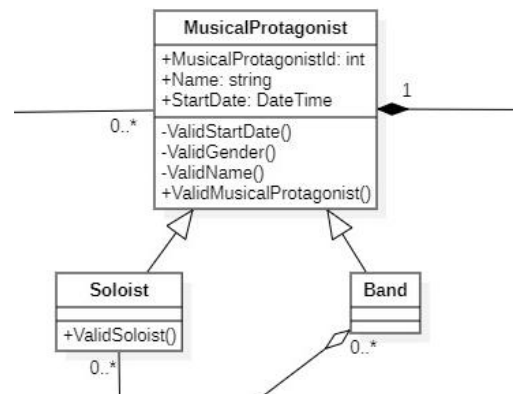


Descripción de jerarquías

En esta solución se implementó únicamente una herencia en el paquete Dominio donde la clase base es MusicalProtagonist y las clases que heredan de la misma son Soloist y Band.

De esta forma no es necesario asociar conciertos a Bandas o Solistas sino que se asocian a un protagonista independientemente del subtipo que sea.

Esto también permite que en el futuro se implemente un nuevo tipo de protagonista y que automáticamente ya se pueda asociar sus conciertos y género.



Modelo de tablas

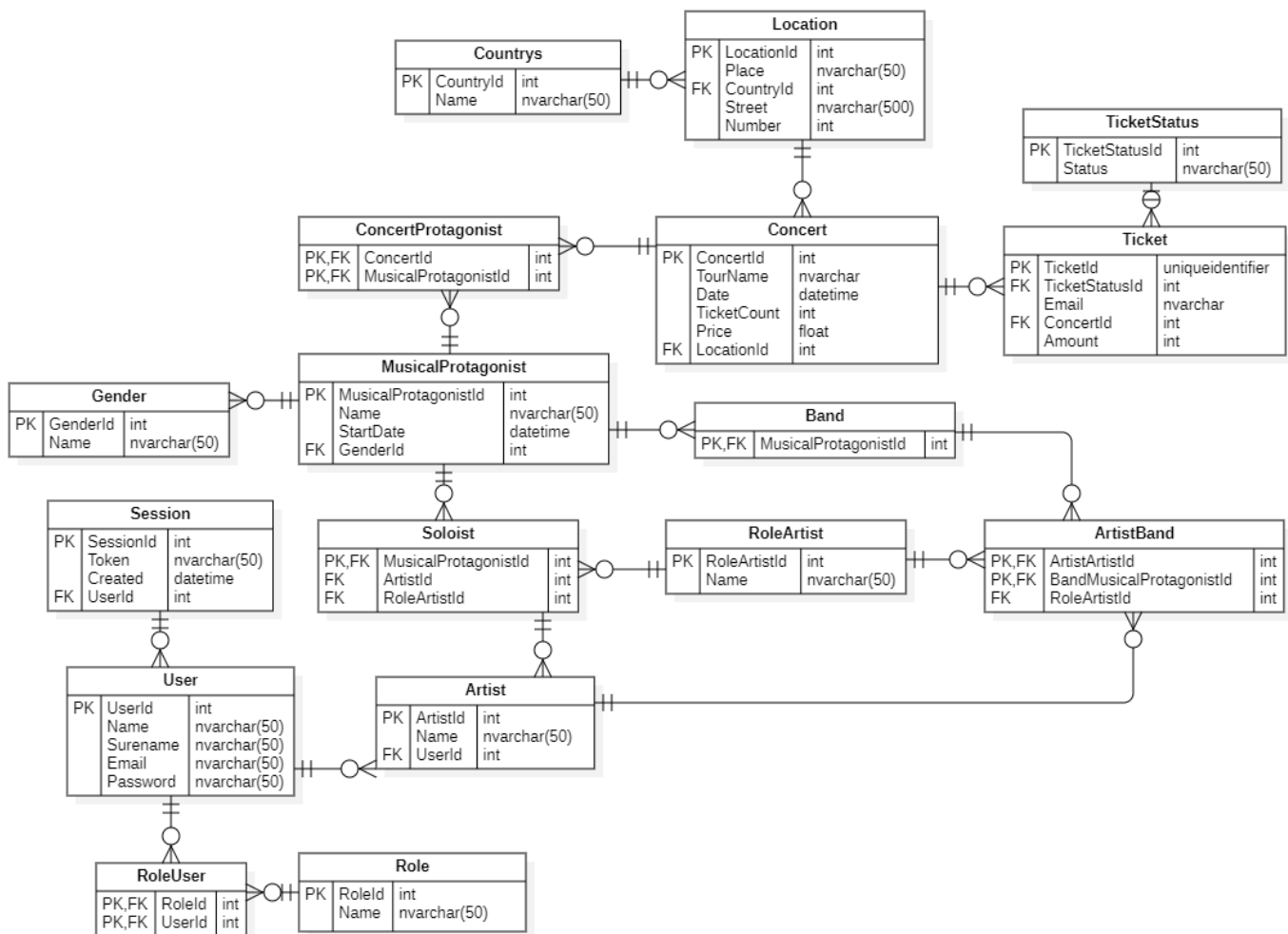
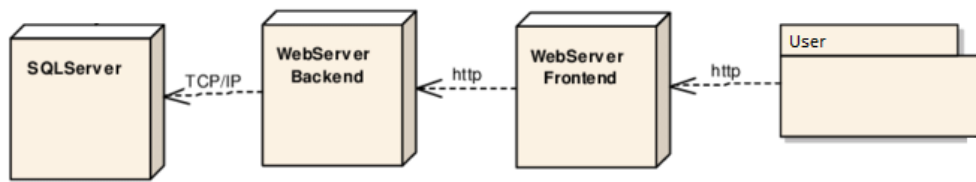


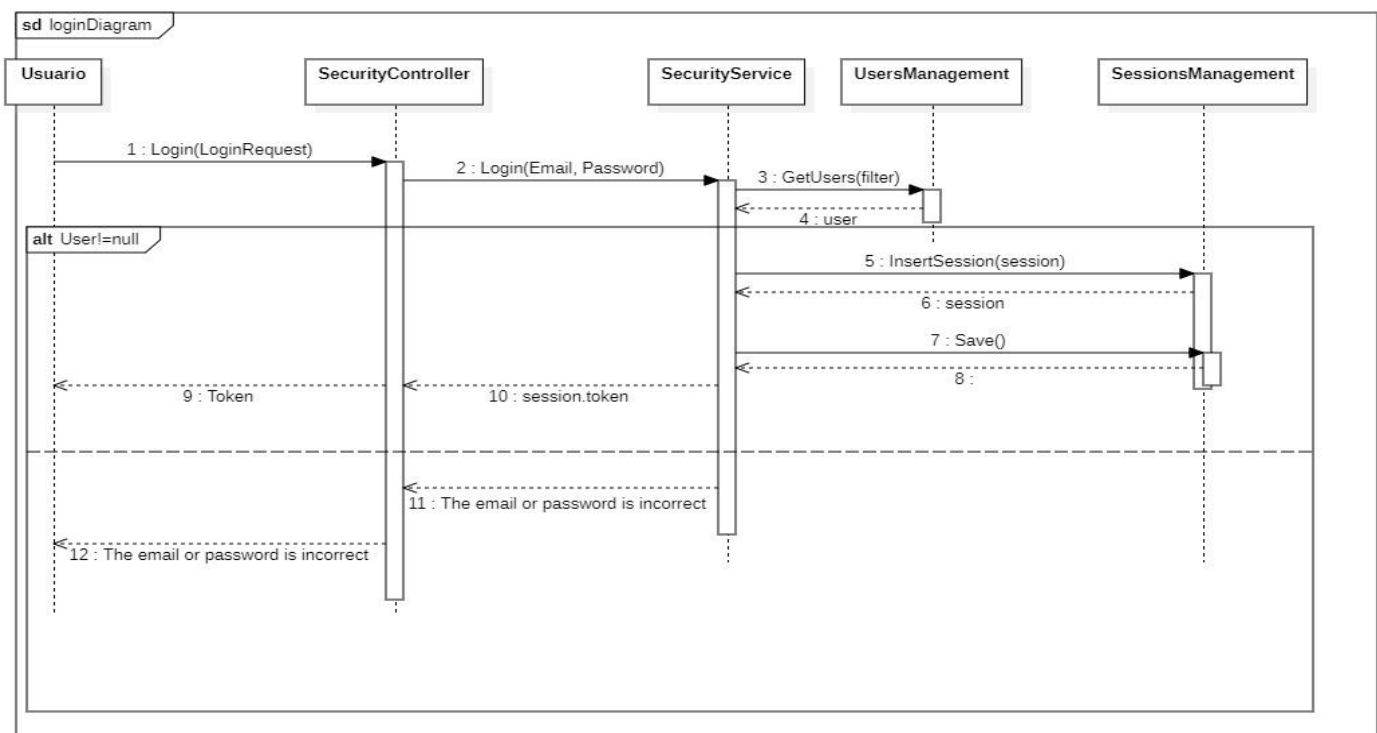
Diagrama de despliegue



El sistema está separado en 3 partes: base de datos, API Reset y FrontEnd desarrollado en Angular. Cada una de estas partes corre de forma independiente del otro. En el caso de este obligatorio el Frontend lo hemos alojado en un sitio de IIS, y en el mismo IIS hemos creado un sitio diferente con puerto diferente para correr la API REST, pero no necesariamente deben estar en el mismo servidor, se pueden hospedar en servidores diferentes, solamente debe haber conectividad entre ellos para poder separarlos. Finalmente el backend se comunica con un servidor SQL Server el cual puede estar en cualquier máquina a la que se tenga conectividad. Tanto la dirección a la que apunta el Frontend como la dirección de SQL Server a la cual apunta el Backend se puede configurar (Ver anexos)

Diagramas de interacción

Login



Export

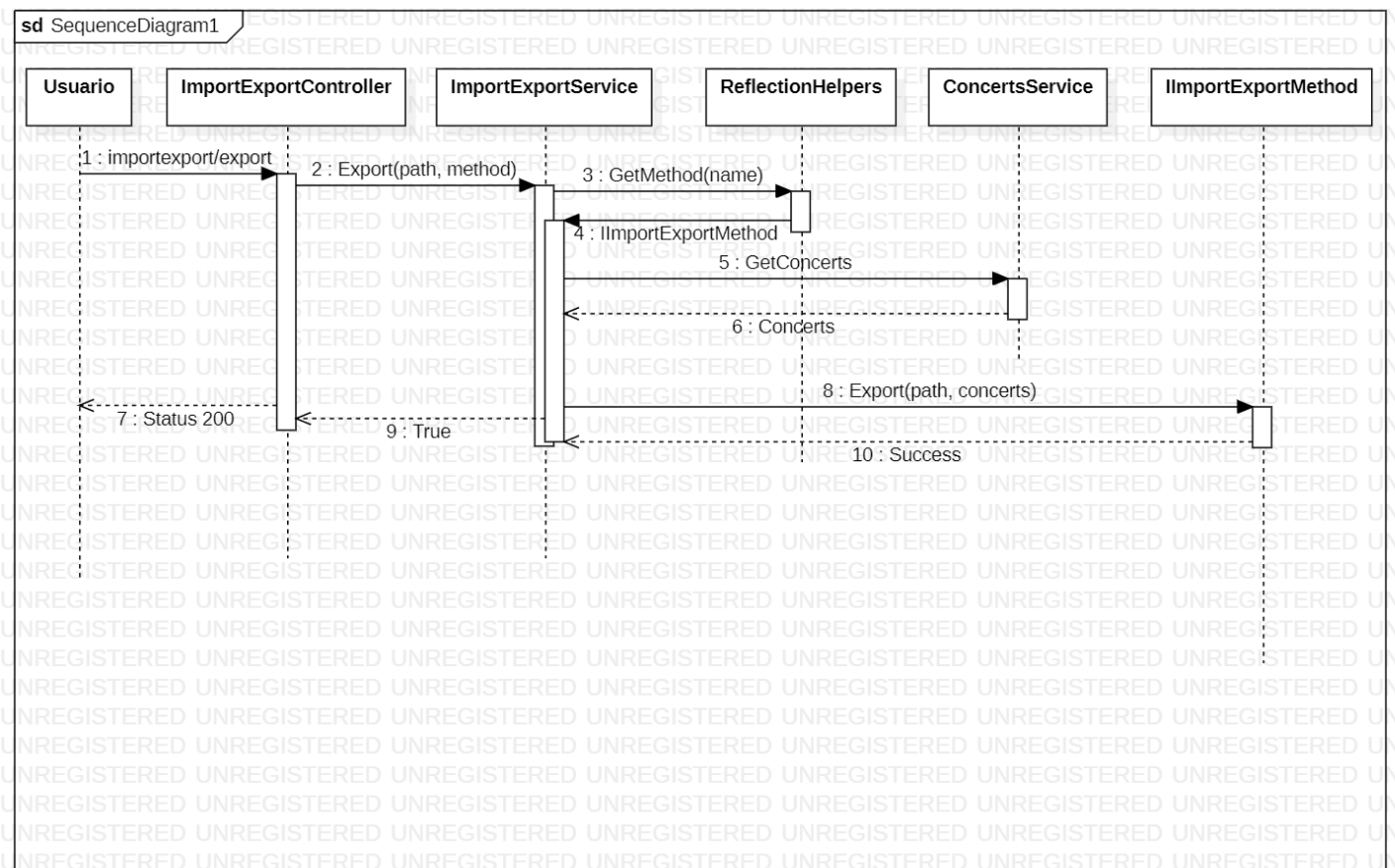
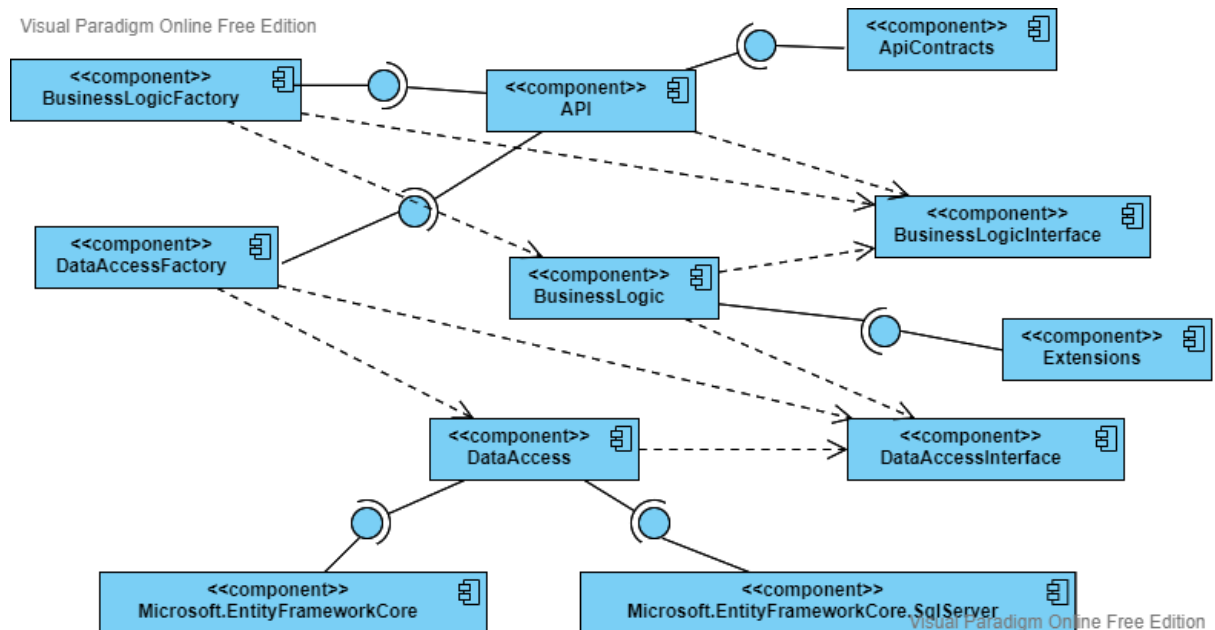


Diagrama de implementación



La solución se separó en diferentes componentes teniendo 3 principales: Business, API y Data Access. Esto permite que cada uno sea independiente del otro y se pueda reemplazar sin necesidad de estar acoplado con un mecanismo específico. Por ejemplo en el caso del componente data access, este fue implementado para leer/escribir sobre una base de datos SQL Server con Entity Framework. Pero podría cambiarse a otro componente que guarde los datos en memoria o una base de datos Documental.

El componente business es el encargado de ejecutar toda la lógica de negocios, esto permite que todos los que quieran interactuar con el sistema, lo hagan a través de este componente, por ejemplo hoy el componente que el usuario usa para acceder al sistema es el de WebApi, pero se podría implementar otro componente "Win forms" y que se comunique con el componente business sin necesidad de modificar nada del mismo.

Extensibilidad del sistema

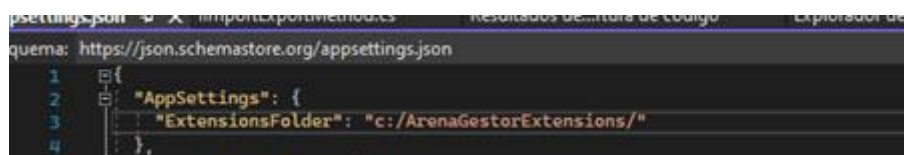
Para la funcionalidad de extensibilidad propuesta decidimos usar Reflection para permitir a terceros desarrollar sus propios componentes y puedan agregarlos de forma fácil al sistema.

En la solución se creó un paquete llamado ArenaGestor.Extensions el cual contiene la interfaz `IImportExportMethod` la cual tiene dos métodos y una variable de clase. La variable debe devolver el nombre del método.

Luego en la interfaz existen los métodos `import` y `export`. El primero recibe un string el cual es suministrado por el usuario mediante el frontend o la web api el cual es el parámetro que recibe este método. Que va a contener este parámetro es definido por el desarrollador de la DLL a desarrollar. En caso de un importador de XML puede ser el path del archivo a importar. Este método debe devolver una lista de `ConcertDTO` el cual es una clase representante de concert. Luego nuestra business logic se encarga de crear esos conciertos en nuestra base de datos.

En el caso del método de `export`, este recibe el string de parámetro y una lista de `ConcertDTO` con todos los conciertos de la base de datos. Luego el desarrollador elige qué hacer con esa lista.

Para leer estas dlls de terceros, se debe configurar en el archivo `appsettings.json` la carpeta de la cual leer estos componentes.

A screenshot of a code editor showing the configuration of an application settings file. The file is named `appsettings.json` and its schema is `https://json.schemastore.org/appsettings.json`. The configuration is as follows:

```
1 {  
2   "AppSettings": {  
3     "ExtensionsFolder": "c:/ArenaGestorExtensions/"  
4   },  
}
```

La business logic lee esta carpeta para reconocer mediante reflection todas las dlls que implementan `ImportExportMethod` en alguna clase.

En el caso de importar, cuando la business logic recibe la lista de conciertos, envía una a una al método de insert que ya teníamos el cual permite que se disparen todas las reglas de negocio previamente programadas. Si un concierto no cumple con alguna regla, ese concierto se ignora, se insertan el resto de conciertos y al usuario se le menciona cuales son los errores de validación en los conciertos no insertados. Para poder insertar un concierto, los protagonistas ya deben existir en la base de datos del sistema.

Este mecanismo podría mejorarse un poco y separar en dos interfaces: una para import y otra para export de forma que se puedan separar los mecanismos y que a alguien que solamente le interesa exportar conciertos no tenga que implementar el método de importar.

En el repositorio de Github se encuentran dos ejemplos de métodos de exportar e importar: uno de XML y otro de JSON. Los mismos se encuentran en la carpeta código. Dentro de esta se pueden ver las dos Dlls así como su código fuente en la carpeta `ArenaGestor.ImportExport`

Justificación y explicación del diseño

Seguridad

Para autenticar los requests es necesario hacer un llamado al endpoint `/Security/login` con el verbo POST y pasando en el body de la request el email y la contraseña del usuario a loguearse.

Esto devolverá un token que deberá especificar en el header de los requests que vaya a hacer en el futuro mediante el header "token"

A nivel técnico para esto creamos un **AuthorizationFilter** que se coloca en cada endpoint y recibe como parámetro cuales son los roles necesarios para consumir ese Endpoint.

El filter obtiene el usuario en base al token recibido y verifica que el usuario tenga alguno de los roles necesarios para consumir el endpoint, si es así, permite continuar con la ejecución del endpoint. En cualquier otro caso devuelve un código de error (401 o 403 según corresponda).

Todos los tokens son almacenados en la tabla Session donde se guarda cada token con el usuario al que corresponde y fecha de creación del mismo.

Excepciones

En el caso del manejo de excepciones, creamos un **ExceptionHandler** que se coloca en cada endpoint. Este filter aplica una política global a las excepciones que ocurren antes de que se cree el cuerpo de la respuesta. Esto nos permite capturar excepciones sin escribir bloques try, catch, implementando un mecanismo genérico de manejo de errores.

Acceso a datos

Para el acceso a los datos, utilizamos Entity Framework Core, que es un ORM que nos permite trabajar con bases de datos usando objetos .NET sin tener que crear código de acceso a datos.

Para conectarnos a la base de datos, creamos una clase contexto y las entidades del negocio.

La clase de contexto es la representación de la sesión con la base de datos que nos permitirá realizar consultas y guardar información. Primero usamos el código como una convención de trabajo, lo que primero diseñamos es el negocio con clases, para que luego EF Core cree el modelo de tabla para crear la base de datos y actualizarla.

Validaciones

Las validaciones referente a la lógica del negocio es realizada en el paquete ArenaGestor.Business donde cada clase Service es responsable de las validaciones globales y cada clase del dominio es el experto para validarse a sí mismo.

No ahondaremos en detalle en lo que respecta a las validaciones de tipo numérica, email, fecha, etc.

Validaciones más importantes:

Artistas

- Al crear/actualizar
 - Se chequea que no exista un artista con el mismo nombre
 - El usuario relacionado tenga el rol artista
- Baja/actualizar
 - Que el ID exista

Géneros

- Al crear
 - Se chequea que no exista un género con el mismo nombre
- Baja/actualizar
 - Que el ID exista

Bandas

- Al crear/actualizar
 - Se chequea que no exista una banda con el mismo nombre

- Se chequea que los artistas asociados existan
 - Se chequea que el género asociado exista
- Baja/actualizar
 - Que el ID exista

Solistas

- Al crear/actualizar
 - Se chequea que no exista una banda con el mismo nombre
 - Se chequea que el artista asociado exista
 - Se chequea que el género asociado exista
- Baja/actualizar
 - Que el ID exista

Conciertos

- Al crear/actualizar
 - Se chequea que los protagonistas asociados existan
 - Se chequea que el país asociado exista
- Baja/actualizar
 - Que el ID exista

Import/export

- Al importar/exportar
 - Se chequea que los métodos por reflection existan

Patrones y principios de diseño utilizados

Patrón Strategy

La intención es definir una familia de algoritmos, encapsularlos, y hacerlos intercambiables en tiempo de ejecución. Este patrón lo utilizamos en la funcionalidad de extensibilidad propuesta donde el algoritmo es encapsulable y se quiere poder intercambiar en tiempo de ejecución

`ImportExportMethod` declara una interfaz común para todos los algoritmos soportados y se implementa el algoritmo utilizando la interfaz `Strategy` el cual queda abierto para las diferentes implementaciones

`ImportExportService` funciona como `Context`, donde se configura con una `ImportExportMethod` en particular e interactúan para implementar el algoritmo elegido.

En este caso estamos usando, composición delegación, polimorfismo, fabricación pura, alto acoplamiento. Nos permite el reúso de algoritmos de una misma familia, extender el comportamiento usando composición-delegación, eliminamos las sentencias `if/else` o `switch` y permite elegir la implementación a usar en tiempo de ejecución.

Patrón Factory

En esta solución se usa la inyección de dependencias que provee `.Net`. Esto nos permite desde el constructor de nuestras clases, solicitar instancias de ciertas

interfaces, de forma que no nos interese su implementación, solamente conocemos qué funciones tiene.

La inyección de dependencias cumple el patrón Factory ya que se le delega la responsabilidad de crear las instancias de cada clase.

En la solución tenemos dos paquetes `ArenaGestor.DataAccessFactory` y `ArenaGestor.BusinessFactory` las cuales son las responsables de registrar cual es la implementación de una interfaz para que luego el mecanismo de inyección de dependencias cree dicha instancia.

Patrón Facade

Este proyecto implementa una API Rest, las api rest son fachadas para que cualquiera pueda comunicarse con nuestro sistema de una forma fácil. De esta forma el usuario externo solamente ve cómo comunicarse con el sistema mediante la documentación de la API sin necesidad de saber que ocurre internamente y poder partir el sistema en Subsistemas. Los servicios de la Business Logic también son una fachada donde la Web Api se comunica reduciendo la complejidad a la hora de implementar un controlador.

Patrones GRASPs:

Hemos realizado el diseño de la aplicación `ArenaGestor` enfocados en cumplir con estos patrones de asignación de responsabilidades

Experto

Hemos asignado las responsabilidades a la clase que tiene la información necesaria para cumplirla. Por ejemplo, cada objeto de dominio sabe si es válido para realizar operaciones.

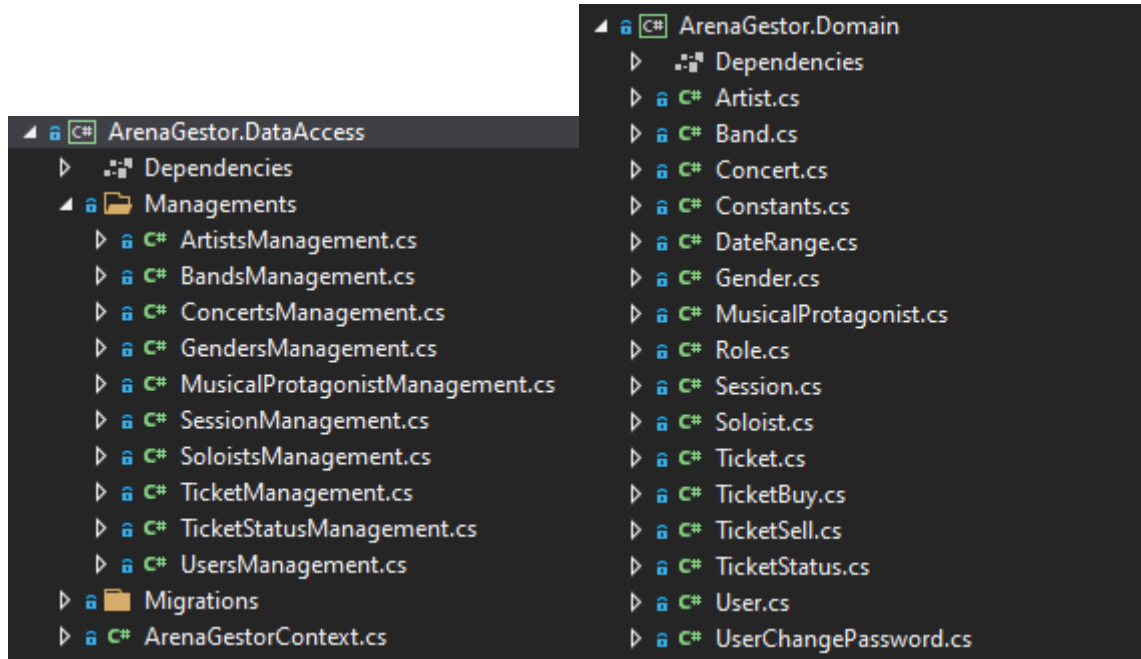
Por ejemplo, la clase `Concert` verifica que:

```
1 reference
public void ValidConcert()
{
    if (!CommonValidations.ValidRequiredString(this.TourName))
    {
        throw new ArgumentException("The name must have at least one character");
    }
    if (this.TourName.Length > 50)
    {
        throw new ArgumentException("Name must be less than 50 characters");
    }
    if (this.Date <= DateTime.Now)
    {
        throw new ArgumentException("The concert must start in the future");
    }
    if (this.Price <= 0)
    {
        throw new ArgumentException("The concert price must be higher than 0");
    }
    if (this.TicketCount <= 0)
    {
        throw new ArgumentException("The concert ticket count must be higher than 0");
    }
}
```

Alta Cohesión

Buscamos la alta cohesión, donde buscamos que están altamente relacionadas las responsabilidades de una clase. Si hay baja cohesión es difícil la reutilización y el mantenimiento.

En este caso, por ejemplo, hemos creado en cada Managements de acceso a datos la responsabilidad de las operaciones que están asociadas a el elemento del Dominio asociado.



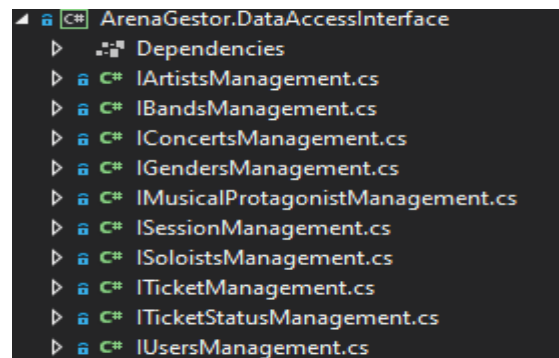
Controlador

Hemos utilizado el patrón controlador, que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, con eso la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. No debe tener lógica.

En este caso por ejemplo, los Controllers, reciben las peticiones de los usuarios y resuelven en que capa lógica debe resolverse dicha solicitud sin aplicar ningún tipo de lógica extra.

Polimorfismo

Hemos creado componentes de software pluggeables sin afectar al clientes, por ejemplo la persistencia de los datos la cual contiene todos los servicios para que los otros paquetes puedan obtener, crear, modificar o borrar registros puede ser modificado podría ser cambiado por una base de datos física o una base de datos en memoria implementando la interfaz correspondiente.

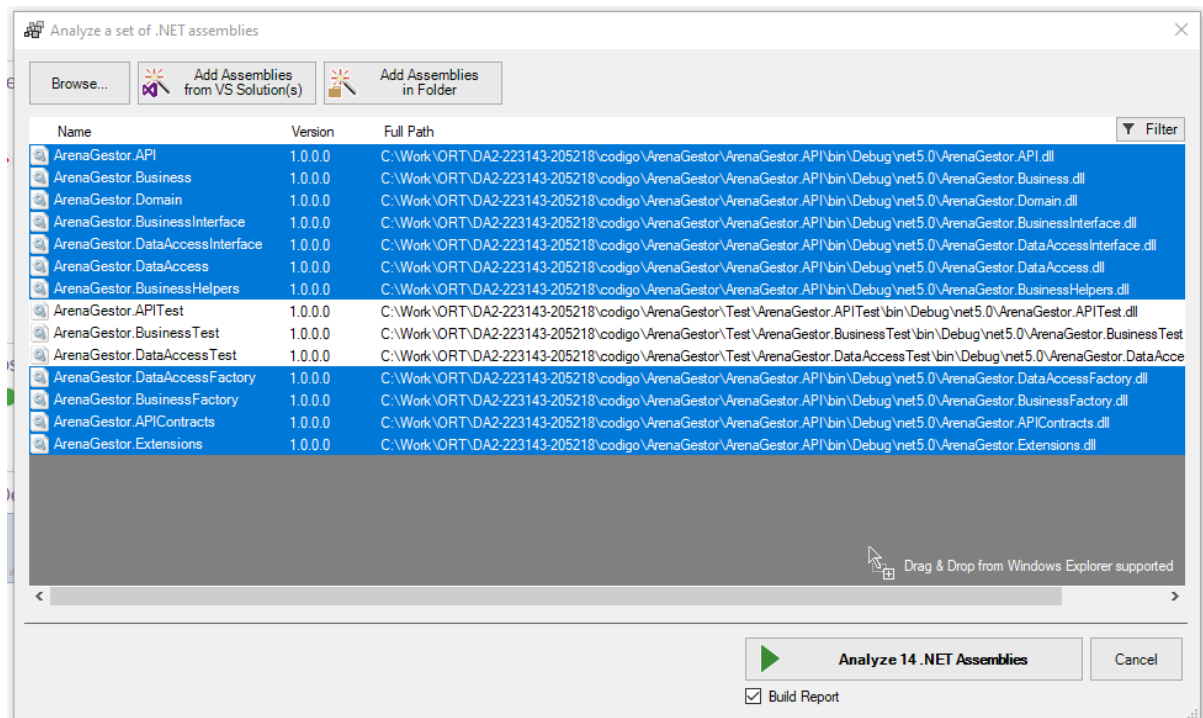


Calidad del diseño

A continuación realizaremos un informe sobre el diseño, basado en métricas, que explican los puntos fuertes del diseño y qué aspectos podrían ser mejorados.

Hemos utilizado NDepend para calcular las métricas.

Es importante aclarar, que para el informe de métricas hemos quitado los proyectos dedicados a los test.



Analizando los principios de diseño a nivel de paquete podemos informar lo siguiente.

A nivel de cohesión

Hemos seguido el **principio de clausura común (CCP)**, donde apuntamos al rápido mantenimiento, ya que se esperaban dos etapas de desarrollo con importantes cambios en la segunda de estas.

En función a la situación al escaso tiempo para el desarrollo y la certeza de cambios las clases candidatas a cambiar juntas se agrupan en un mismo paquete, por ejemplo el paquete *ArenaGestor.DataAccess* el cual maneja el acceso a datos, al ser necesario una nueva clase relacionadas con el acceso a datos, esta va donde están todos los demás de acceso a datos.

Utilizar CPP llevó a que algunos paquetes crezcan pero consideremos que tienen un tamaño razonable en función a la solución global.

En este sentido, consideramos que podríamos mejorar el diseño para permitir que el sistema se particione en paquetes reusables, por ejemplo, lo correspondiente a seguridad y gestión de usuarios lo separaríamos de lo que corresponde a bandas, solistas y conciertos, somos conscientes que no corresponde que convivan en los mismos paquetes y que podría reutilizarse si lo desacopláramos.

A nivel de acoplamiento

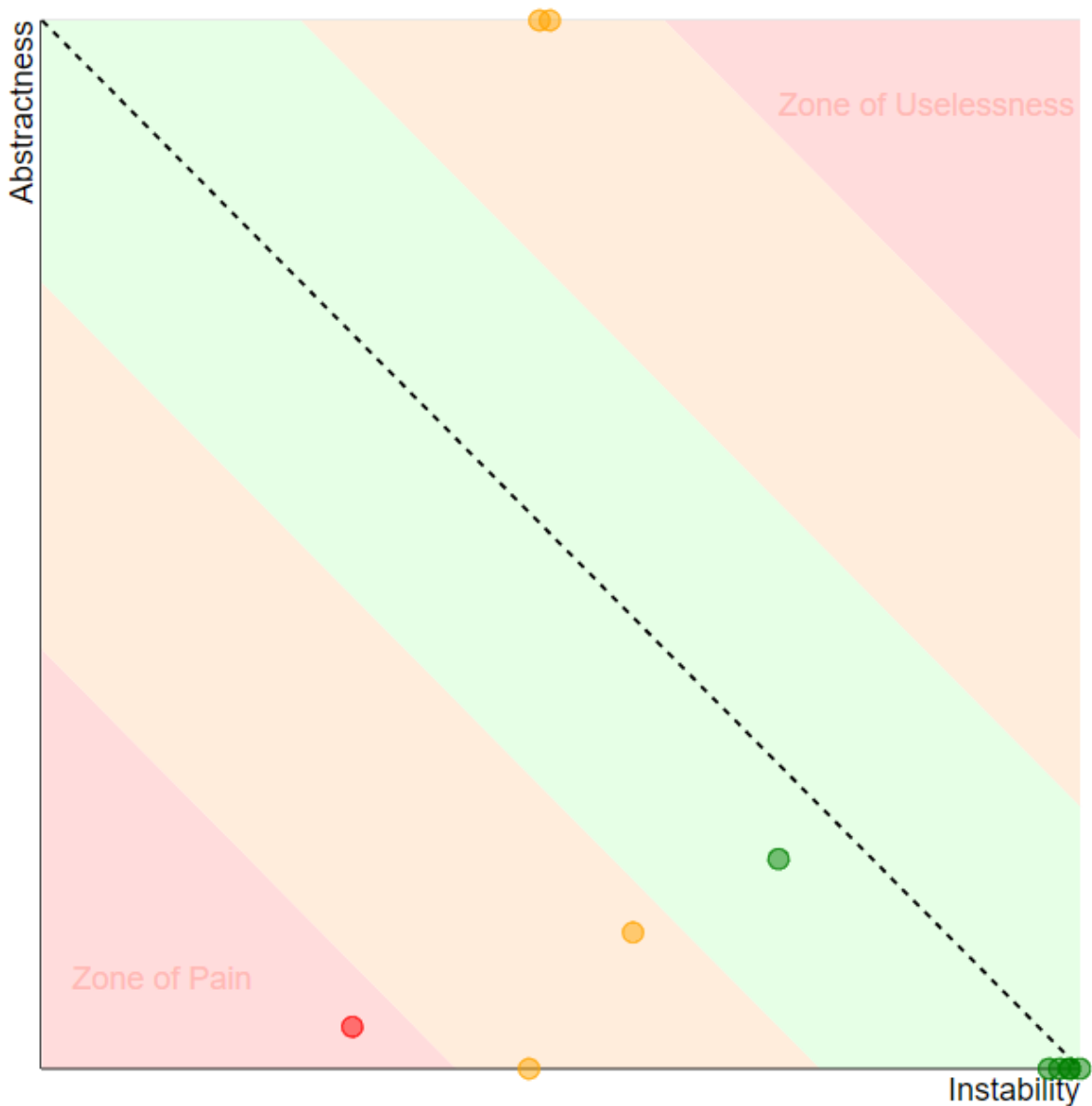
Consideramos que cumplimos el principio de **dependencias acíclicas (ADP)** ya que la estructura de dependencia de paquetes debe formar un grafo dirigido y acíclico.

Para entrar en más detalle respecto a los principios de diseño a nivel de acoplamiento y para profundizar en métricas, a continuación, presentamos un resumen de las métricas de diseño de nuestro proyecto.

| Assembly | I | A | D | D' |
|---------------------------------|------|---|------|------|
| ArenaGestor.Business | 0,99 | | 0 | 0,01 |
| ArenaGestor.DataAccess | 0,99 | | 0 | 0,01 |
| ArenaGestor.DataAccessFactory | 0,98 | | 0 | 0,02 |
| ArenaGestor.BusinessFactory | 0,97 | | 0 | 0,03 |
| ArenaGestor.Extensions | 0,71 | | 0,2 | 0,06 |
| ArenaGestor.API | 0,57 | | 0,13 | 0,21 |
| ArenaGestor.APIContracts | 0,57 | | 0,13 | 0,21 |
| ArenaGestor.DataAccessInterface | 0,48 | | 1 | 0,34 |
| ArenaGestor.BusinessInterface | 0,49 | | 1 | 0,35 |
| ArenaGestor.BusinessHelpers | 0,47 | | 0 | 0,37 |
| ArenaGestor.Domain | 0,30 | | 0,04 | 0,47 |

- **I** = Grado de inestabilidad asociado a un paquete
- **A** = Abstracción del paquete
- **D** = Distancia a la secuencia principal
 - Valores cercanos a 0 (zona de dolor) indican que el paquete es concreto y estable.
 - Valores cercanos a 1 (zona de poca utilidad) indican que el paquete es abstracto e inestable.

Los datos anteriores nos dan como resultado la siguiente gráfica de abstracción y estabilidad, donde los colores corresponden a la ubicación en el mismo.



Principio de dependencias estables (SDP)

A partir de la información recabada, donde las dependencias entre paquetes deben ir en sentido a la estabilidad y por ende un paquete solamente debe depender de otros que sean más estables que él.

Para determinar si cumplimos, se detalla las dependencias entre paquetes:

- Valor cercano a 0 - máxima estabilidad
- Valor cercano a 1 - máxima inestabilidad

Obtenemos la siguiente información:

| Assembly | I | Depends on | I2 |
|---------------------------------|------|---------------------------------|------|
| ArenaGestor.Domain | 0,30 | ArenaGestor.BusinessHelpers | 0,47 |
| ArenaGestor.API | 0,57 | ArenaGestor.BusinessFactory | 0,97 |
| | | ArenaGestor.DataAccessFactory | 0,98 |
| | | ArenaGestor.APIContracts | 0,57 |
| | | ArenaGestor.Domain | 0,30 |
| | | ArenaGestor.Extensions | 0,71 |
| | | ArenaGestor.BusinessInterface | 0,49 |
| ArenaGestor.BusinessInterface | 0,49 | ArenaGestor.Domain | 0,30 |
| | | ArenaGestor.Extensions | 0,71 |
| ArenaGestor.Extensions | 0,71 | - | - |
| ArenaGestor.APIContracts | 0,57 | - | - |
| ArenaGestor.BusinessHelpers | 0,47 | - | - |
| ArenaGestor.DataAccessInterface | 0,48 | ArenaGestor.Domain | 0,30 |
| ArenaGestor.Business | 0,99 | ArenaGestor.DataAccessInterface | 0,48 |
| | | ArenaGestor.BusinessInterface | 0,49 |
| | | ArenaGestor.Domain | 0,30 |
| | | ArenaGestor.BusinessHelpers | 0,47 |
| | | ArenaGestor.Extensions | 0,71 |
| ArenaGestor.BusinessFactory | 0,97 | ArenaGestor.BusinessInterface | 0,49 |
| | | ArenaGestor.Business | 0,99 |
| ArenaGestor.DataAccess | 0,99 | ArenaGestor.Domain | 0,30 |
| | | ArenaGestor.DataAccessInterface | 0,48 |
| ArenaGestor.DataAccessFactory | 0,98 | ArenaGestor.DataAccessInterface | 0,48 |
| | | ArenaGestor.DataAccess | 0,99 |

Los paquetes que no dependen de nadie y es correcta su métrica

- ArenaGestor.Extensions
- ArenaGestor.APIContracts
- ArenaGestor.BusinessHelpers

Los paquetes que dependen de paquetes más estables por lo cual es correcta su métrica

- ArenaGestor.DataAccessInterface
- ArenaGestor.Business
- ArenaGestor.BusinessFactory
- ArenaGestor.DataAccess
- ArenaGestor.DataAccessFactory

Casos intermedios,

- ArenaGestor.BusinessInterface
 - Depende de ArenaGestor.Extensions, consideremos que fue una mala decisión de diseño.

Paquetes con SDP incorrecto

- ArenaGestor.Domain
 - Depende del paquete ArenaGestor.BusinessHelpers, este paquete tiene como funcionalidad hacer validaciones sencillas y que se repiten en varias partes del código. El dominio lo utiliza, ya que consideramos que cada clase debe saber si es válida (principio de responsabilidad), es por esto que el dominio utiliza este paquete para validar ids, email, strings requeridos, etc.
- ArenaGestor.API
 - En este caso, depende de dos paquetes más inestables, estos son ArenaGestor.BusinessFactory y ArenaGestor.DataAccessFactory los cuales se encargan de hacer la inyección de dependencias por ende es en program donde son invocados.

Principio de abstracciones estables (SAP)

Un paquete debe ser tan abstracto como su estabilidad por ende los paquetes más estables (cerca de 0) deben tender a ser más abstractos (cerca de 1) y paquetes inestables deben ser concretos.

En este sentido, para contemplar de manera gráfica ubicamos que el paquete ArenaGestor.Domain donde están las clases del dominio que representan la realidad es el paquete que no cumple este criterio. El dominio es bastante estable en comparación con el resto pero es muy concreto dando como resultado que viole este principio y ubicándose en la zona de dolor lo cual implica que este paquete no es extensible y si cambian impactan en otros paquetes.

El resto de los paquetes en la secuencia principal de R.C.Martin cumplen con el principio.

Anexo

Datos de prueba

Se ha entregado la aplicación con una base de datos con datos de prueba, de manera de poder comenzar las pruebas sin tener que definir una cantidad de datos iniciales. A continuación, se especifican dichos datos de prueba.

Datos globales de la aplicación

Estado de tickets

| TicketStatusId | Status |
|----------------|-----------|
| 1 | Comprado |
| 2 | Utilizado |

Roles de artistas

| RoleArtistId | Name |
|--------------|-------------|
| 1 | Cantante |
| 2 | Baterista |
| 3 | Bajista |
| 4 | Guitarrista |
| 5 | Coro |

Roles de usuario

| RoleId | Name |
|--------|---------------|
| 3 | Acomodador |
| 1 | Administrador |
| 5 | Artista |
| 4 | Espectador |
| 2 | Vendedor |

Países:

| CountryId | Name |
|-----------|-----------|
| 1 | Uruguay |
| 2 | Argentina |

Géneros:

| GenderId | Name |
|----------|-------------|
| 1 | Rock |
| 2 | Pop |
| 3 | Electronica |
| 4 | Cumbia |
| 5 | Jazz |
| 6 | Grunge |
| 7 | Rap |
| 8 | Reggae |
| 9 | Blues |
| 10 | Punk |

Usuarios y sus roles

Usuarios:

Para iniciar sesión, deberán ingresar los correspondientes email y password.

| UserId | Name | Surname | Email | Password |
|--------|---------------|---------|----------------------------------|----------|
| 1 | Francisco | Rosello | francisco.rosello.fran@gmail.com | fran |
| 2 | Maximiliano | Pascale | maximiliano.pascale@gmail.com | maxi |
| 3 | Acomodador | Test | acomodador@example.com | test |
| 4 | Vendedor | Test | vendedor@example.com | test |
| 5 | Administrador | Test | administrador@example.com | test |
| 6 | Espectador | Test | espectador@example.com | test |
| 7 | Super | Super | super@example.com | super |
| 8 | Kurt | Cobain | kurt@example.com | test |
| 9 | Freddie | Mercury | fredy@example.com | test |

Roles de usuarios

Con el fin de realizar pruebas que cumplan con la especificación, se han creado diferentes usuarios con diferentes roles.

Super usuarios, se han creado 3 super usuarios, estos tienen todos los roles asignados.

- francisco.rosello.fran@gmail.com
- maximiliano.pascale@gmail.com
- super@example.com

Se ha creado un usuario por rol de sistema:

- Administrador administrador@example.com
- Acomodador acomodador@example.com
- Vendedor vendedor@example.com
- Espectador espectador@example.com

Por último se han creado 2 usuarios de tipo artista, lo cual permite consultar sus conciertos

- kurt@example.com
- fredy@example.com

| UserId | Email | Name | Surname | Role |
|--------|----------------------------------|---------------|---------|---------------|
| 1 | francisco.rosello.fran@gmail.com | Francisco | Rosello | Administrador |
| 1 | francisco.rosello.fran@gmail.com | Francisco | Rosello | Vendedor |
| 1 | francisco.rosello.fran@gmail.com | Francisco | Rosello | Acomodador |
| 1 | francisco.rosello.fran@gmail.com | Francisco | Rosello | Espectador |
| 1 | francisco.rosello.fran@gmail.com | Francisco | Rosello | Artista |
| 2 | maximiliano.pascale@gmail.com | Maximiliano | Pascale | Administrador |
| 2 | maximiliano.pascale@gmail.com | Maximiliano | Pascale | Vendedor |
| 2 | maximiliano.pascale@gmail.com | Maximiliano | Pascale | Acomodador |
| 2 | maximiliano.pascale@gmail.com | Maximiliano | Pascale | Espectador |
| 2 | maximiliano.pascale@gmail.com | Maximiliano | Pascale | Artista |
| 3 | acomodador@example.com | Acomodador | Test | Acomodador |
| 4 | vendedor@example.com | Vendedor | Test | Vendedor |
| 5 | administrador@example.com | Administrador | Test | Administrador |
| 6 | espectador@example.com | Espectador | Test | Espectador |
| 7 | super@example.com | Super | Super | Administrador |
| 7 | super@example.com | Super | Super | Vendedor |
| 7 | super@example.com | Super | Super | Acomodador |
| 7 | super@example.com | Super | Super | Espectador |
| 7 | super@example.com | Super | Super | Artista |
| 8 | kurt@example.com | Kurt | Cobain | Artista |
| 9 | fredy@example.com | Freddie | Mercury | Artista |

Para hacer más comprensible la información, se documenta la consulta realizada.

```
SELECT u.UserId, u.Email, u.Name, u.Surname, r.Name AS Role
FROM RoleUser ru
INNER JOIN [User] u ON ru.UserId = u.UserId
INNER JOIN Role r ON ru.RoleId = R.RoleId
ORDER BY ru.UserId, ru.RoleId;
```

Artistas, solistas y bandas.

Aristas:

Los artistas son personas particulares, esto nos permite:

- Crear bandas compuestas por varios artistas
- Crear solistas compuestos por un único artista
- Podremos usar un mismo artista para varias bandas así como permitir que sea solista a la vez.

| ArtistId | Name | UserId |
|----------|------------------|--------|
| 1 | Paul McCartney | NULL |
| 2 | George Harrison | NULL |
| 3 | Ringo Starr | NULL |
| 4 | John Lennon | NULL |
| 5 | Bob Dylan | NULL |
| 6 | Elvis Presley | NULL |
| 7 | Mick Jagger | NULL |
| 8 | Keith Richards | NULL |
| 9 | Ronnie Wood | NULL |
| 10 | Jimi Hendrix | NULL |
| 11 | Bob Marley | NULL |
| 12 | Jimmy Page | NULL |
| 13 | Robert Plant | NULL |
| 14 | John Paul Jones | NULL |
| 15 | John Bonham | NULL |
| 16 | Stevie Wonder | NULL |
| 17 | Joey Ramone | NULL |
| 18 | Johnny Ramone | NULL |
| 19 | Marky Ramone | NULL |
| 20 | Dee Dee Ramone | NULL |
| 21 | Kurt Cobain | 8 |
| 22 | David Guetta | NULL |
| 23 | Agustin Casanova | NULL |
| 24 | Dave Grohl | NULL |
| 25 | Krist Novoselic | NULL |
| 26 | Freddie Mercury | 9 |
| 27 | Roger Taylor | NULL |
| 28 | John Deacon | NULL |
| 29 | Brian May | NULL |
| 30 | Angus Young | NULL |
| 31 | Stevie Young | NULL |
| 32 | Brian Johnson | NULL |
| 33 | Cliff Williams | NULL |
| 34 | Phil Rudd | NULL |
| 35 | Julieta Venegas | NULL |
| 36 | Sebastián Yatra | NULL |

Bandas

Como mencionamos anteriormente, una banda está compuesta por artistas.

| MusicalProtagonistId | BandName | ArtistName | GenderName | RoleName |
|----------------------|--------------------|-----------------|------------|-------------|
| 1 | The Beatles | Paul McCartney | Rock | Bajista |
| 1 | The Beatles | George Harrison | Rock | Guitarrista |
| 1 | The Beatles | Ringo Starr | Rock | Baterista |
| 1 | The Beatles | John Lennon | Rock | Cantante |
| 2 | The Rolling Stones | Mick Jagger | Rock | Cantante |
| 2 | The Rolling Stones | Keith Richards | Rock | Bajista |
| 2 | The Rolling Stones | Ronnie Wood | Rock | Guitarrista |
| 7 | Led Zeppelin | Jimmy Page | Rock | Guitarrista |
| 7 | Led Zeppelin | Robert Plant | Rock | Cantante |
| 7 | Led Zeppelin | John Paul Jones | Rock | Bajista |
| 7 | Led Zeppelin | John Bonham | Rock | Baterista |
| 9 | Ramones | Joey Ramone | Punk | Cantante |
| 9 | Ramones | Johnny Ramone | Punk | Guitarrista |
| 9 | Ramones | Marky Ramone | Punk | Baterista |
| 9 | Ramones | Dee Dee Ramone | Punk | Bajista |
| 10 | Nirvana | Kurt Cobain | Grunge | Cantante |
| 10 | Nirvana | Dave Grohl | Grunge | Bajista |
| 10 | Nirvana | Krist Novoselic | Grunge | Baterista |
| 13 | Queen | Freddie Mercury | Rock | Cantante |
| 13 | Queen | Roger Taylor | Rock | Baterista |
| 13 | Queen | John Deacon | Rock | Bajista |
| 13 | Queen | Brian May | Rock | Guitarrista |
| 14 | AC/DC | Angus Young | Rock | Guitarrista |
| 14 | AC/DC | Stevie Young | Rock | Guitarrista |
| 14 | AC/DC | Brian Johnson | Rock | Cantante |
| 14 | AC/DC | Cliff Williams | Rock | Bajista |
| 14 | AC/DC | Phil Rudd | Rock | Baterista |

Para hacer más comprensible la información, se documenta la consulta realizada.

```

SELECT b.MusicalProtagonistId, m.Name BandName, a.Name ArtistName,
g.Name GenderName, r.Name RoleName
FROM Band b
INNER JOIN ArtistBand ba ON ba.MusicalProtagonistId = b.MusicalProtagonistId
INNER JOIN Artist a ON a.ArtistId = ba.ArtistId
INNER JOIN MusicalProtagonist m ON ba.MusicalProtagonistId = m.MusicalProtagonistId
INNER JOIN RoleArtist r ON ba.RoleArtistId = r.RoleArtistId
INNER JOIN Gender g ON g.GenderId = m.GenderId
ORDER BY m.MusicalProtagonistId;

```

Solistas

Como mencionamos anteriormente, un solista está compuesto por un artista.

| MusicalProtagonistId | ArtistName | GenderName | RoleName |
|----------------------|------------------|-------------|-------------|
| 3 | Bob Dylan | Rock | Cantante |
| 4 | Elvis Presley | Rock | Cantante |
| 5 | Jimi Hendrix | Rock | Guitarrista |
| 6 | Bob Marley | Reggae | Cantante |
| 8 | Stevie Wonder | Jazz | Cantante |
| 11 | David Guetta | Electronica | Cantante |
| 12 | Agustin Casanova | Cumbia | Cantante |
| 15 | Julieta Venegas | Pop | Cantante |
| 16 | Sebastián Yatra | Cumbia | Cantante |

Para hacer más comprensible la información, se documenta la consulta realizada.

```
SELECT s.MusicalProtagonistId, m.Name ArtistName, g.Name
GenderName, r.Name RoleName
FROM Soloist s
INNER JOIN MusicalProtagonist m ON s.MusicalProtagonistId = m.MusicalProtagonistId
INNER JOIN RoleArtist r ON r.RoleArtistId = s.RoleArtistId
INNER JOIN Gender g ON g.GenderId = m.GenderId
ORDER BY m.MusicalProtagonistId;
```

Conciertos

Los conciertos son un punto importante en la aplicación, por lo cual, hemos creado una variedad de estos.

| ConcertId | TourName | Date | TicketCount | Price | LocationId |
|-----------|---------------------------|-----------------------------|-------------|-------|------------|
| 1 | Nevermind | 2022-08-15 03:00:00.0000000 | 50000 | 2000 | 1 |
| 2 | The World Tour | 2022-06-20 03:00:00.0000000 | 60000 | 10000 | 2 |
| 3 | The Beatles' 1966 US tour | 2022-06-15 01:16:01.1630000 | 50000 | 4000 | 3 |
| 4 | América Latina Olé Tour | 2023-06-01 03:00:00.0000000 | 20000 | 1000 | 4 |
| 5 | Solistas Tour | 2022-07-10 03:00:00.0000000 | 20000 | 500 | 5 |
| 6 | Cumbia Tour | 2022-06-20 03:00:00.0000000 | 8000 | 1500 | 6 |

Como requerimiento, un concierto puede ser realizado por varios solistas y bandas, para esto, en cada concierto hemos creado varias casuísticas, como, conciertos solo por solistas, solo por bandas, solistas y bandas. A continuación los casos creados.

| ConcertId | Name | TourName |
|-----------|--------------------|---------------------------|
| 1 | Nirvana | Nevermind |
| 2 | The Beatles | The World Tour |
| 2 | The Rolling Stones | The World Tour |
| 2 | Bob Dylan | The World Tour |
| 2 | Elvis Presley | The World Tour |
| 2 | Jimi Hendrix | The World Tour |
| 2 | Bob Marley | The World Tour |
| 2 | Led Zeppelin | The World Tour |
| 2 | Stevie Wonder | The World Tour |
| 2 | Ramones | The World Tour |
| 2 | Nirvana | The World Tour |
| 2 | Queen | The World Tour |
| 2 | AC/DC | The World Tour |
| 3 | The Beatles | The Beatles' 1966 US tour |
| 4 | The Rolling Stones | América Latina Olé Tour |
| 5 | Bob Dylan | Solistas Tour |
| 5 | Elvis Presley | Solistas Tour |
| 5 | Jimi Hendrix | Solistas Tour |
| 5 | Bob Marley | Solistas Tour |
| 5 | Stevie Wonder | Solistas Tour |
| 5 | David Guetta | Solistas Tour |
| 6 | Agustin Casanova | Cumbia Tour |
| 6 | Sebastián Yatra | Cumbia Tour |

Para hacer más comprensible la información, se documenta la consulta realizada.

```

SELECT c.ConcertId, m.Name, c.TourName
FROM [ArenaGestorDB].[dbo].[ConcertProtagonist] cp
INNER JOIN [ArenaGestorDB].[dbo].[Concert] c ON cp.ConcertId =
c.ConcertId
INNER JOIN MusicalProtagonist m ON m.MusicalProtagonistId =
cp.MusicalProtagonistId
ORDER BY c.ConcertId

```

Tickets

El último punto son los tickets

| TicketId | Status | Email | Amount | TourName |
|--------------------------------------|----------|------------------------|--------|---------------------------|
| 57F1B4CD-D610-4A1C-EF42-08DA4E703754 | Comprado | sinregistrar@user.com | 100 | The Beatles' 1966 US tour |
| CD3ECF9F-1BAC-4003-EF43-08DA4E703754 | Comprado | espectador@example.com | 100 | The Beatles' 1966 US tour |
| 558F9AEB-2DF1-4180-EF44-08DA4E703754 | Comprado | espectador@example.com | 25000 | Nevermind |
| AEF531B6-0DDF-4B78-EF45-08DA4E703754 | Comprado | espectador@example.com | 25000 | Nevermind |

Se han vendido tickets de tal forma que:

- Venta a usuario no registrado
- Venta a usuario registrado
- Venta Sold Out de concierto Nevermind

Para hacer más comprensible la información, se documenta la consulta realizada.

```
SELECT t.TicketId, ts.Status, t.Email, t.Amount, c.TourName
FROM [ArenaGestorDB].[dbo].[Ticket] t
INNER JOIN Concert c ON c.ConcertId = t.ConcertId
INNER JOIN TicketStatus ts ON ts.TicketStatusId =
t.TicketStatusId
```

Documentación de la API

Adjunto a este documento se encuentra la carpeta “Documentación API” el cual tiene el archivo “Documentation.html” el cual es una página web donde se especifica cada uno de los endpoints agregados o modificados, todos los códigos de errores que devuelve cada uno, headers necesarios y estructura del body de ser necesario.

Para generar este documento se utilizó la herramienta “Swagger hub” la cual permite generar toda la documentación de forma automática mediante un archivo Open API. Dicho archivo es generado por Visual Studio. El mismo contiene información incompleta de todos los endpoints y mediante Swagger hub se agrega la información faltante como los posibles códigos de error, descripción de cada endpoint y cuales requieren un token de autenticación.

En particular en esta entrega se borraron los endpoints:

- /concert/upcomingConcerts
- /concert/dateRangeConcert

Los mismos fueron reemplazados por el endpoint /concert/ donde ahora en el header se pueden especificar parámetros como el Upcoming y fechas de inicio y fin para filtrar cumpliendo la misma finalidad que los endpoints anteriores.

Se corrigen por no cumplir con REST, por utilizar un verbo

- /Tickets/Buy -> /Tickets/Shopping
- /Tickets/Scan -> /Tickets/ PUT
- /Tickets/Sell -> /Tickets/Sale

Se agregan los siguientes endpoints por los nuevos requerimientos:

- Concerts/GetConcertsByArtist
- Countrys/GetCountrys
- Roles/GetUserRoles
- Roles/GetArtistRoles

Informe de cobertura

| Jerarquía | Cubiertos (% de bloques) | No cubiertos (% de bloques) | No cubiertos (bloques) | Cubiertos (bloques) |
|--|--------------------------|-----------------------------|------------------------|---------------------|
| ▲ [icon] frose_NIGHT1_2022-06-15.21_24_14.coverage | 99,02 % | 0,98 % | 26 | 2627 |
| ▲ [icon] arenagestor.api.dll | 98,73 % | 1,27 % | 5 | 389 |
| ▲ { } ArenaGestor.API.Controllers | 98,42 % | 1,58 % | 5 | 311 |
| ▶ [icon] ArtistsController | 100,00 % | 0,00 % | 0 | 29 |
| ▶ [icon] BandsController | 100,00 % | 0,00 % | 0 | 35 |
| ▶ [icon] ConcertsController | 100,00 % | 0,00 % | 0 | 40 |
| ▶ [icon] CountrysController | 100,00 % | 0,00 % | 0 | 7 |
| ▶ [icon] GendersController | 100,00 % | 0,00 % | 0 | 29 |
| ▶ [icon] ImportExportController | 100,00 % | 0,00 % | 0 | 20 |
| ▶ [icon] RolesController | 100,00 % | 0,00 % | 0 | 12 |
| ▶ [icon] SecurityController | 73,68 % | 26,32 % | 5 | 14 |
| ▶ [icon] SoloistsController | 100,00 % | 0,00 % | 0 | 35 |
| ▶ [icon] TicketsController | 100,00 % | 0,00 % | 0 | 35 |
| ▶ [icon] UsersController | 100,00 % | 0,00 % | 0 | 55 |
| ▲ { } ArenaGestor.API.Filters | 100,00 % | 0,00 % | 0 | 78 |
| ▶ [icon] AuthorizationFilter | 100,00 % | 0,00 % | 0 | 47 |
| ▶ [icon] ExceptionFilter | 100,00 % | 0,00 % | 0 | 31 |
| ▲ [icon] arenagestor.business.dll | 99,11 % | 0,89 % | 10 | 1115 |
| ▲ { } ArenaGestor.Business | 99,04 % | 0,96 % | 10 | 1030 |
| ▶ [icon] ArtistsService | 100,00 % | 0,00 % | 0 | 124 |
| ▶ [icon] BandsService | 100,00 % | 0,00 % | 0 | 107 |
| ▶ [icon] ConcertsService | 94,97 % | 5,03 % | 10 | 189 |
| ▶ [icon] CountrysService | 100,00 % | 0,00 % | 0 | 13 |
| ▶ [icon] GendersService | 100,00 % | 0,00 % | 0 | 73 |
| ▶ [icon] ImportExportService | 100,00 % | 0,00 % | 0 | 31 |
| ▶ [icon] MusicalProtagonistService | 100,00 % | 0,00 % | 0 | 21 |
| ▶ [icon] RolesService | 100,00 % | 0,00 % | 0 | 8 |
| ▶ [icon] SecurityService | 100,00 % | 0,00 % | 0 | 67 |
| ▶ [icon] SoloistsService | 100,00 % | 0,00 % | 0 | 99 |
| ▶ [icon] TicketService | 100,00 % | 0,00 % | 0 | 125 |
| ▶ [icon] UsersService | 100,00 % | 0,00 % | 0 | 173 |
| ▲ { } ArenaGestor.Business.Helpers | 100,00 % | 0,00 % | 0 | 85 |
| ▶ [icon] ReflectionHelpers | 100,00 % | 0,00 % | 0 | 85 |
| ▲ [icon] arenagestor.businesshelpers.dll | 96,43 % | 3,57 % | 1 | 27 |
| ▲ { } ArenaGestor.BusinessHelpers | 96,43 % | 3,57 % | 1 | 27 |
| ▶ [icon] CommonValidations | 94,74 % | 5,26 % | 1 | 18 |
| ▶ [icon] StringGenerator | 100,00 % | 0,00 % | 0 | 9 |
| ▲ [icon] arenagestor.dataaccess.dll | 99,10 % | 0,90 % | 10 | 1096 |
| ▲ { } ArenaGestor.DataAccess.Managements | 99,10 % | 0,90 % | 10 | 1096 |
| ▶ [icon] ArtistsManagement | 93,98 % | 6,02 % | 8 | 125 |
| ▶ [icon] BandsManagement | 99,59 % | 0,41 % | 1 | 243 |
| ▶ [icon] ConcertsManagement | 99,63 % | 0,37 % | 1 | 269 |
| ▶ [icon] CountrysManagement | 100,00 % | 0,00 % | 0 | 20 |
| ▶ [icon] GendersManagement | 100,00 % | 0,00 % | 0 | 36 |
| ▶ [icon] MusicalProtagonistManagement | 100,00 % | 0,00 % | 0 | 30 |
| ▶ [icon] RolesManagement | 100,00 % | 0,00 % | 0 | 10 |
| ▶ [icon] SessionManagement | 100,00 % | 0,00 % | 0 | 31 |
| ▶ [icon] SoloistsManagement | 100,00 % | 0,00 % | 0 | 158 |
| ▶ [icon] TicketManagement | 100,00 % | 0,00 % | 0 | 68 |
| ▶ [icon] TicketStatusManagement | 100,00 % | 0,00 % | 0 | 21 |
| ▶ [icon] UsersManagement | 100,00 % | 0,00 % | 0 | 85 |

En esta entrega las pruebas automáticas cubren el 99.02% de los bloques totales.

Se puede ver que en cada uno de los paquetes la cobertura de pruebas es alta (Mayor a 95%).

Para el paquete BusinessHelpers no existen casos de prueba, estos métodos se prueban ya que son utilizados por varias clases de la Business Logic de la cual hay un 99.04% de cobertura.

Existe un margen de mejora para agregar más tests que cubren el 100% de los bloques. Así como también agregar pruebas funcionales ya que todas las pruebas actuales son unitarias.

Las pruebas se desarrollaron usando Moq para crear mocks de las clases de las cuales dependen varias de las clases, de forma que cada método de la solución se testea de forma aislada y sin preocuparse de si otro método externo funciona o no.

Para ejecutar este análisis se desactivó que se verifique la cobertura sobre clases autogeneradas por el Framework tales como las migraciones. Y otras propias de una WebApi como la clase Startup. También se desactivó el análisis sobre las clases de dominio ya que estas no tienen funcionalidad, son solamente estructuras. Por último se desactivó el análisis en paquetes que solamente contienen interfaces, paquetes de testing y paquetes Factory.

Test

Para el desarrollo de los casos de prueba de la solución se usó la metodología de Test Driven Development la cual se basa en escribir primero los casos de prueba para luego programar las funcionalidades en sí.

Particularmente se usó la estrategia de Outside In la cual se basa en no saber o no tener en cuenta cual es el estado de objetos externos al método que queremos escribir. Esto significa que todas las dependencias externas de un método son mapeadas a un Mock, o sea un objeto falso que está programado para devolver respuestas específicas a las llamadas que pueda hacerle el método a testear. De esta forma la prueba se ejecuta sin necesidad de depender de ninguna otra clase y de forma totalmente aislada centrándonos en la funcionalidad del método a implementar.

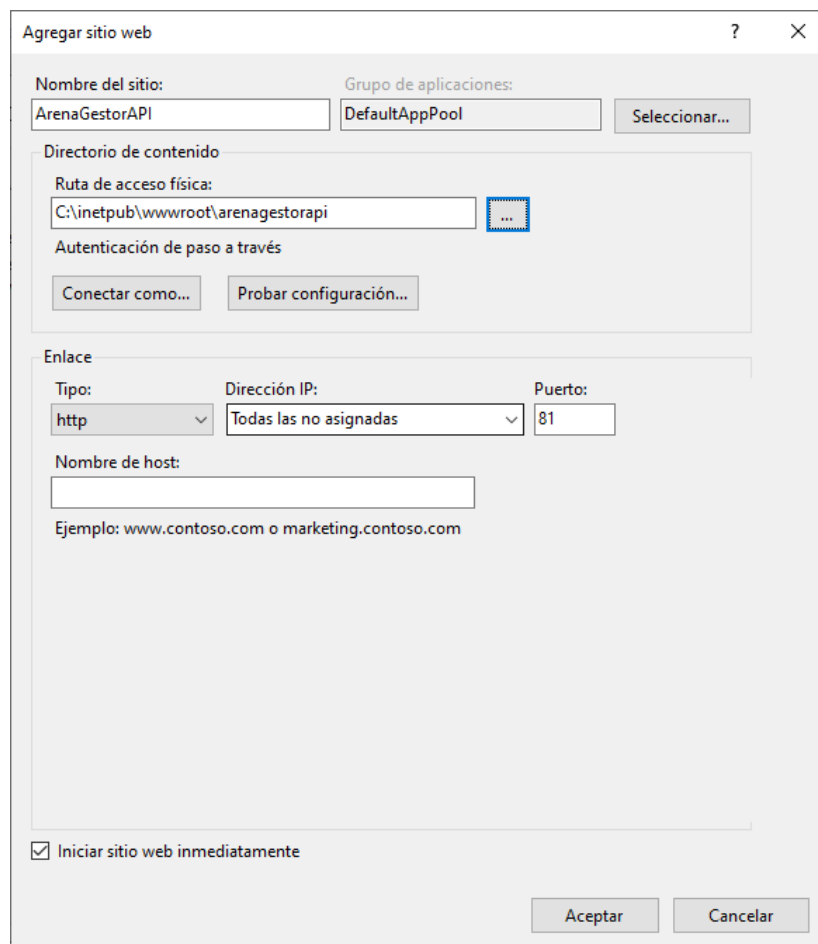
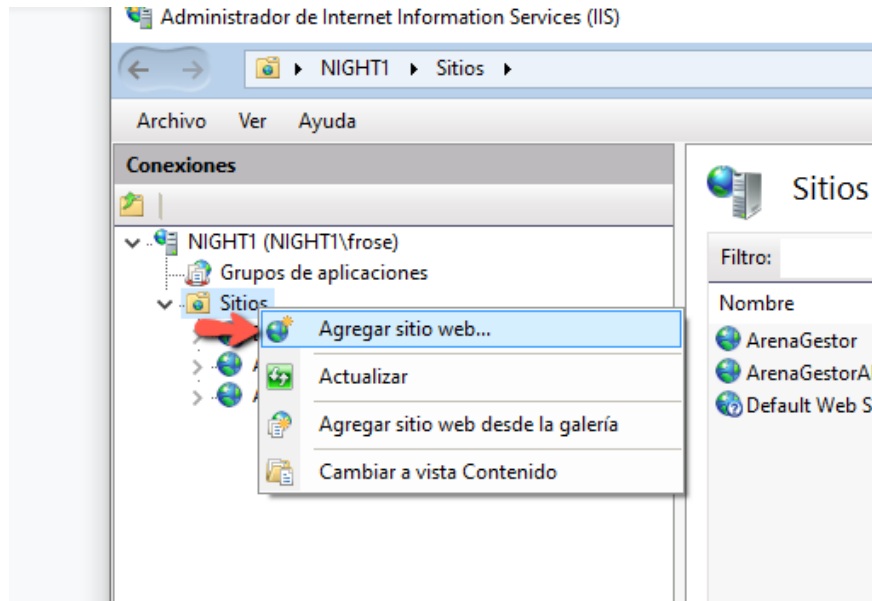
Para testear toda la solución, se creó un proyecto de testing para los paquetes API, Business y Data Access.

No se creó uno para el BusinessHelpers ya que sus métodos son usados en toda la solución y se testean al crear casos de pruebas de los paquetes que usan este.

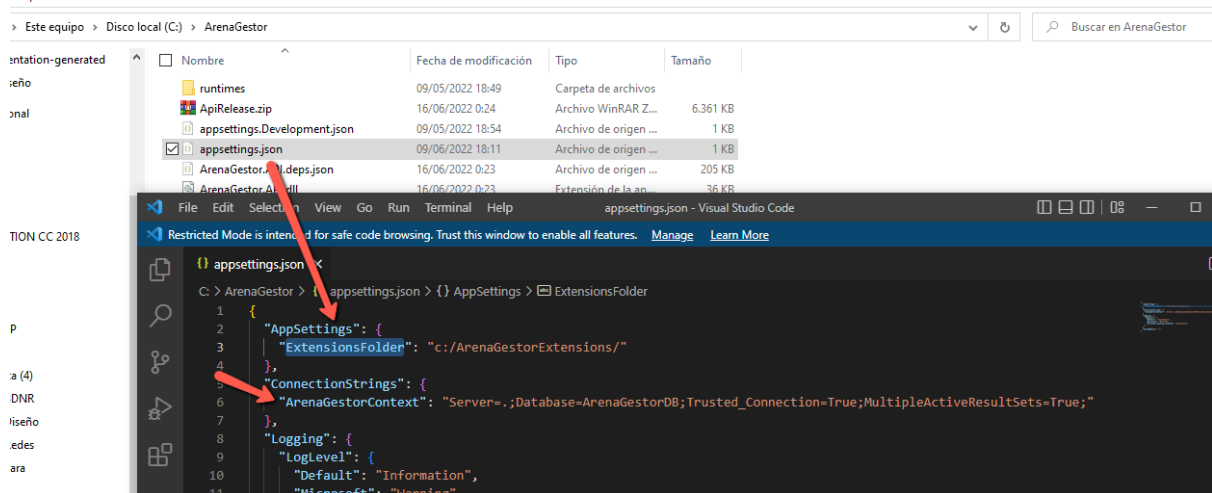
Todo el código implementado en esta solución cumple los lineamientos de Clean Code del capítulo 1 al 10 y el 12.

Manual de instalación

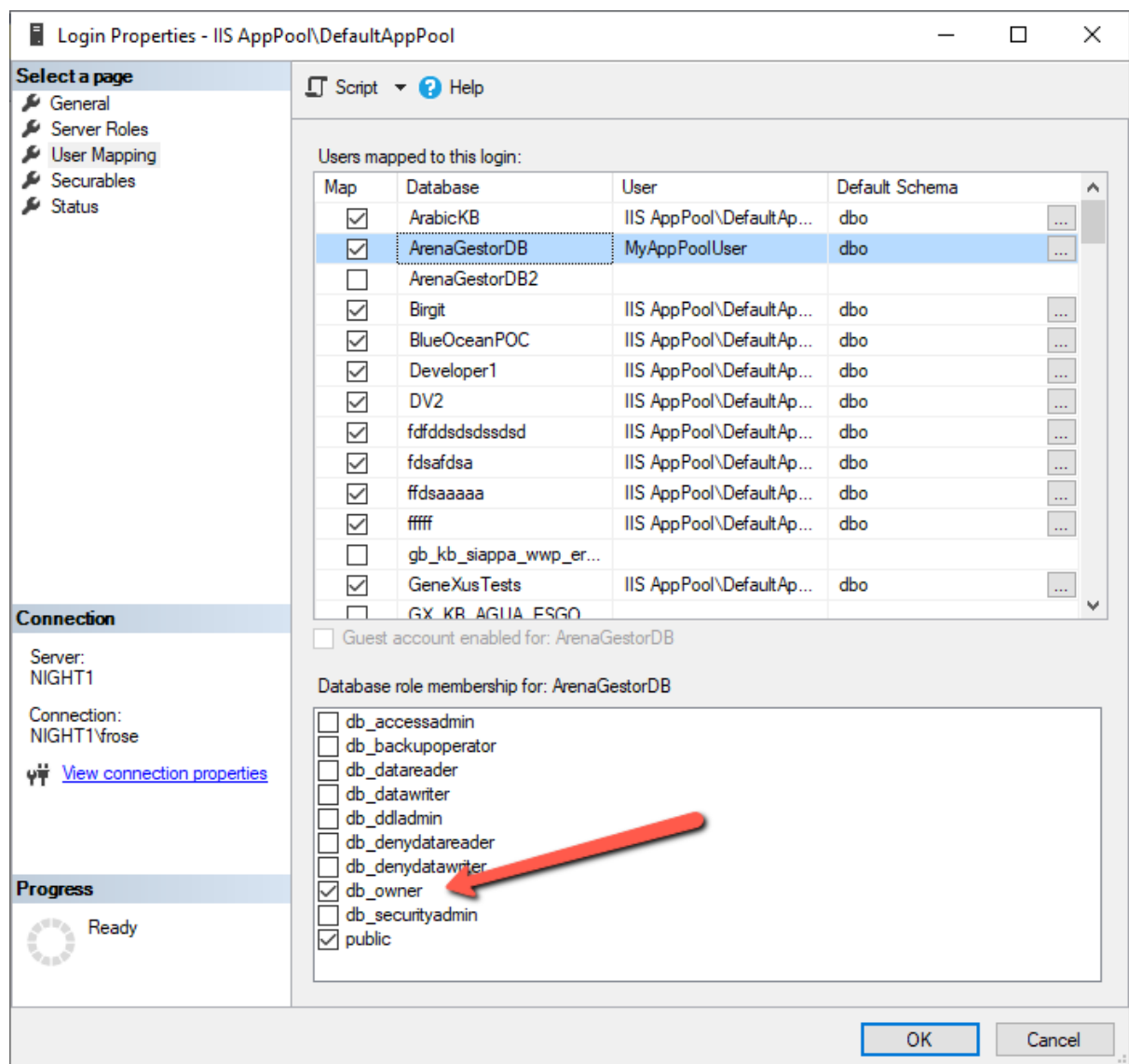
- 1) Importar .bak de la base de datos
- 2) Copiar los archivos de la API en una carpeta de nuestra computadora
- 3) Crear un sitio IIS apuntando a la carpeta anterior.



4) Configurar el Connection string y el ExtensionsFolder del archivo appsettings.json

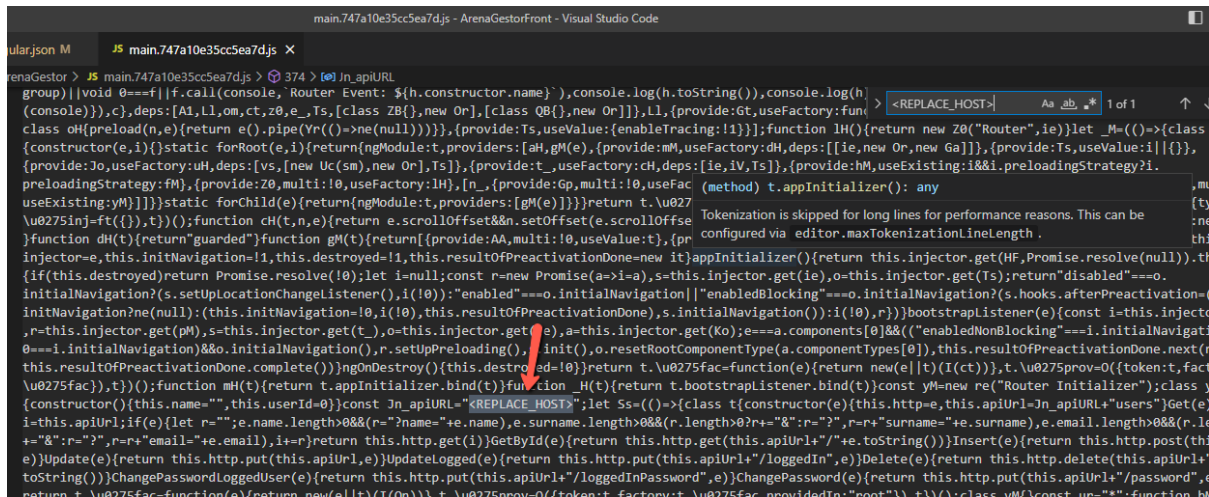


Asignar permisos en la base de datos al usuario de IIS:



Crear otro directorio y otro sitio el frontend.

Modificar el archivo main y cambiarle el tag <REPLACE_HOST> por la dirección de la web api.



Códigos de error

En esta tabla se detallan los posibles códigos de error que puede devolver la API:

| Código | Posibles mensajes |
|--------|---|
| 200 | Resource solicitado, insertado, modificado. Mensaje vacío en caso de un delete |
| 400 | Bad Request. Se devuelve cuando hay algún parámetro de la petición (Tanto en el header como en el body) que no es correcto, por ejemplo no se envió un ítem del body el cual es requerido |
| 401 | Token not specified. Se da cuando el Endpoint necesita estar autenticado y no se recibió el token. |
| 403 | Forbidden. Se da cuando el token especificado es inválido o no se tiene permisos sobre el endpoint |
| 404 | Not found. Endpoint inválido o resource solicitado no encontrado. |
| 500 | Internal server error. Se da cuando hay un error del lado del servidor |