

Universidad ORT Uruguay

Facultad De Ingeniería

Obligatorio - Diseño de Aplicaciones 1

**Entregado como requisito para la obtención del Obligatorio 2 de
Diseño de Aplicaciones 1**

Federico Gutiérrez - 262684

Franco Ramos - 230508

Docente - Tutor: Gastón Mosques

Docentes Práctico: Ignacio Azaretto – Martin Radovitzky

Grupo: M5D

2024

https://github.com/IngSoft-DA1-2023-2/262684_230508

Declaración de autoría

Nosotros, Federico Gutiérrez y Franco Ramos, 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 Obligatorio 2 de Diseño de Aplicaciones 1;
- 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 como por ejemplo la inteligencia artificial “ChatGPT” y el asistente virtual de inteligencia artificial “copilot” entre otros;
- 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;

Federico Gutiérrez

10/06/2024

Franco Ramos

10/06/2024

Índice

Declaración de autoría.....	2
Descripción general del trabajo y del sistema.....	4
Trabajo realizado.....	4
Aplicación desarrollada.....	4
Bugs y funcionalidades no implementadas.....	5
Requerimientos Funcionales.....	5
Gestión de disponibilidad de reservas.....	5
Pagos.....	5
Exportar reporte de reservas.....	5
Descripción y justificación del diseño.....	9
Diagramas de Paquetes.....	9
DataAccess.....	9
UserInterface.....	11
Dependencias entre paquetes.....	11
Dependencia entre Controllers-DataAccess.....	11
Dependencia entre UserInterface-DataAccess.....	13
Dependencia entre DataAccess-Model.....	13
Diagramas de Clases.....	15
Diagramas de Interacción.....	17
Diagrama de secuencia Creación de una promoción.....	17
Diagrama de secuencia Exporte de un reporte de reservas.....	18
Diagrama de secuencia Usuario realiza un Booking.....	19
Modelo de Tablas (Imagen en el anexo).....	21
Mecanismos generales y decisiones de diseño.....	22
Interacción Interfaz-Dominio.....	22
Almacenamiento de Datos.....	22
Manejo de Errores y Excepciones.....	22
Utilización de GRASP y SOLID.....	22
Criterios seguidos para asignar responsabilidades.....	23
Análisis de dependencias.....	23
Cobertura de pruebas unitarias (Imagen en el anexo).....	25
Datos precargados en la base de datos (En carpeta en el repo).....	25
Anexo.....	28

Descripción general del trabajo y del sistema

Trabajo realizado

El proyecto consistió en desarrollar una plataforma web para la iniciativa emergente DepoQuick, la cual tiene como propósito revolucionar el mercado de alquileres de depósitos. A esta plataforma web se la conoce como “MVP” (Producto Mínimo Viable). En la creación de la misma se utilizaron tecnologías como .NET Core 6, Blazor Server y Entity Framework Core para poder establecer una experiencia eficiente y atractiva para el usuario. En esta entrega se insistió en poder mantener la persistencia de los datos de la aplicación, manteniendo la lógica de la primera entrega y agregando nuevos requerimientos funcionales.

Por lo tanto, para la comprensión del proyecto se presenta al cliente:

- Documentación completa con el análisis de la aplicación y sus nuevas funcionalidades implementadas. Cabe destacar que cumple con todos los requerimientos funcionales solicitados.
- Implementación de conexión e interacción con una base de datos SQL Server Express 2019 para la persistencia de datos.
- Diagramas de paquetes completos para representar la estructura de la aplicación y de sus componentes, con sus dependencias. Logrando así poder entender la lógica de los mismos.
- Diagramas de clases UML para representar las estructuras de las clases y sus conexiones con la aplicación.
- Diagramas de secuencia para demostrar cómo el usuario interactúa con la aplicación utilizando sus funcionalidades.

Aplicación desarrollada

La aplicación final permite realizar las siguientes funcionalidades:

Admin

- **Registro** de usuarios
- **Creación, Eliminación y Modificación** de promociones
- **Creación y Eliminación** de depósitos
- **Aprobar, Rechazar o Ver estado** de reservas de usuarios
- **Exportar** un reporte de las reservas de los usuarios

Usuario

- **Crear y Pagar** reservas de depósitos
- **Ver** sus reservas de depósitos activos y no activos

Bugs y funcionalidades no implementadas

En esta sección se reportan los errores o bugs de la aplicación, los cuales pueden haber surgido debido a falta de tiempo o una implementación inadecuada del código que no se pudo solucionar a tiempo. Sin embargo, en esta entrega no hemos encontrado errores que sean de nuestro conocimiento. Hemos validado todo lo más posible para asegurar la calidad del producto final. Solamente en la UserInterface al catchear las excepciones usamos el método genérico (Exception e), pero deberíamos haber utilizado las excepciones personalizadas que hicimos, pero por falta de tiempo no lo concretamos. De igual forma funcionan correctamente.

Requerimientos Funcionales

Gestión de disponibilidad de reservas

Para este requerimiento, se realizó un cambio en la estructura del depósito, por lo tanto, un cambio en el objeto StorageUnit. Este ahora almacena un nombre que le ingresa el administrador. Asimismo, el puede ingresar la fecha en la cual el depósito estará disponible para reservas. Tiene su ventana para gestionarlo y añadimos también que se puedan eliminar. Por más que no se requería nos pareció lo indicado si puede crearlas de igual modo puede eliminarlas. Por otro lado, el usuario para buscar dichos depósitos deberá ingresar la fecha en la cual desearía realizar una reserva y se le mostrarán todas las opciones. Si intenta crear una reserva de un depósito ya reservado se informará del problema.

Pagos

Dicho requerimiento incluye la iniciativa para que los usuarios puedan hacer pagos de sus reservas dentro de la aplicación. Una vez reservado el depósito con el monto indicado, en la lista de reservas del usuario le aparecerá su depósito en estado “reservado” hasta que el administrador lo confirme. Antes de que el administrador pueda confirmar o rechazar la reserva el usuario deberá abonar el monto indicado. Luego de ser aprobado por el administrador, el estado se actualizará a “capturado”. De ser rechazada la reserva incluimos que aparezca al usuario que el estado queda como “rechazado”. Cabe destacar que el administrador también puede ver los estados de las reservas de los usuarios.

Exportar reporte de reservas

Como último requerimiento se solicitó que el administrador sea capaz de exportar un reporte si lo desea de las reservas de los usuarios en un formato TXT (separado por tabuladores \t) y CSV (separado por comas).

Los datos a exportar de las reservas serán los siguientes:

- Datos del depósito
- Datos de la reserva
- Estado del pago

Instalación

Para la correcta instalación de la Aplicación se deben de instalar ciertos ejecutables descritos a continuación. La explicación será enfocada al Sistema Operativo Windows.

Para realizar la aplicación utilizamos JetBrains Rider 2023 con la **versión 6.0** de .NET

Descarga con el link: <https://www.jetbrains.com/es-es/rider/download/#section=windows>

Descargar Rider

Windows macOS Linux

Rider incluye una clave de licencia para un **período de evaluación gratis de 30 días**.

Descargar

.exe (Windows) ▼

Para levantar el contenedor de la aplicación y poder conectarse a la base de datos se debe instalar Docker. Docker es una plataforma de software que permite crear, probar y desplegar aplicaciones rápidamente. Utiliza la tecnología de contenedores, que proporciona un entorno aislado y consistente para que las aplicaciones se ejecuten en cualquier sistema, independientemente de las diferencias en las configuraciones del sistema operativo subyacente.

Se puede instalar con el siguiente link: <https://www.docker.com/products/docker-desktop/>

Docker Desktop

The #1 containerization software for developers and teams

Your command center for innovative container development

Get Started

Download for Windows



Luego en el proyecto se debe ingresar el siguiente comando: **docker-compose up**

Por otro lado, para crear la base de datos utilizamos Azure SQL. Azure SQL Database es una solución de base de datos en la nube ofrecida por Microsoft Azure, que proporciona escalabilidad, alta disponibilidad y seguridad para los datos. DBeaver, una herramienta de administración y desarrollo de bases de datos de código abierto, facilita la interacción con Azure SQL Database mediante una interfaz gráfica amigable.

SQL Server en Azure: <https://www.microsoft.com/es-es/sql-server/sql-server-downloads>

Dbeaver: <https://dbeaver.io/download/>

Download

DBeaver Community 24.1

Released on June 3rd 2024 ([Milestones](#)).

It is free and open source ([license](#)).

Also you can get it from the [GitHub mirror](#).

[System requirements](#).

Windows

- [Windows \(installer\)](#)
- [Windows \(zip\)](#)
- [Chocolatey](#) (`choco install dbeaver`)
- [Install from Microsoft Store](#)

Por otro lado, se debe asegurar la instalación de los siguientes paquetes en la solución de la aplicación:

- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.Design
- Microsoft.EntityFrameworkCore.InMemory (Librería para Test Unitarios, unicamente instalar en DataAccess)
- Microsoft.EntityFrameworkCore.SqlServer

Estos últimos mencionados se utilizan tanto en DataAccess como en UserInterface.

Para crear la base de datos y realizar las migraciones de las tablas de forma automática con EF se tiene que ingresar el siguiente comando:

```
dotnet ef migrations --project DataAccess/DataAccess.csproj --startup-project UserInterface/UserInterface.csproj
```

Strings de conexión a modificar

Los string de conexión a modificar se encuentran en el archivo **appsettings.json** y en el archivo **appsettings.Development.json** en el paquete **UserInterface**, actualmente existe un string de conexión, uno para Windows (ApplicationDBLocalConnection).

String de Conexión a modificar para sistemas Windows:

```
"Server=127.0.0.1;Database=DepoQuick;User Id=sa;Password=Passw1rd;TrustServerCertificate=True"
```

Server = 127.0.0.1 (localhost)

Database = DepoQuick (nombre de la base de datos)

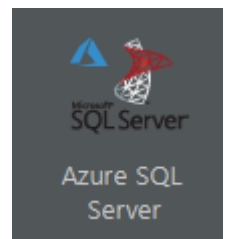
User Id = sa (id del usuario para conexión a la base de datos)

Password = Passw1rd (contraseña del usuario para conexión a la base de datos)

TrustServerCertificate = True (este parámetro es para especificar que la aplicación debe confiar en el servidor de la base de datos sin realizar chequeos/validaciones exhaustivas)

Aclaración para visualizar Base de Datos en DBeaver

Se debe seleccionar la opción de Azure SQL server que es la cual utilizamos en nuestro caso

The screenshot shows the 'Conectar a base de datos' (Connect to database) dialog box in DBeaver. The 'SQL Server Connection Settings' tab is active, showing 'Azure SQL Server ajustes de conexión'. The 'General' sub-tab is selected. Under 'Server', 'Connect by' is set to 'Host' (selected) with 'URL' set to 'jdbc:sqlserver://serverName=127.0.0.1;databaseName=DepoQuick;encrypt=true'. 'Host' is '127.0.0.1' and 'Database/Schema' is 'DepoQuick'. Under 'Authentication', 'Authentication' is set to 'SQL Server Authentication'. 'Nombre de usuario' is 'sa' and 'Contraseña' is masked with dots. 'Save password' is checked. Under 'Settings', 'Show All Databases', 'Show All Schemas', and 'Trust Server Certificate' are all checked. At the bottom, there are buttons for 'Probar conexión ...', 'Anterior', 'Siguiente', 'Finalizar', and 'Cancelar'. A hint at the bottom says 'You can use variables in connection parameters.' and there are links for 'Connection details (name, type, ...)', 'Driver Settings', and 'Licencia del driver'.

Ubicación de los Backups

El Backup, del modelo de la Base de Datos con datos precargados se encuentran en el repositorio vinculado a este documento en la carpeta "Backups", dentro de esta carpeta se encuentran los scripts correspondientes para la creación del modelo de la Base de Datos.

Administrador inicial de la Aplicación

El administrador inicial de la aplicación que decidimos crear para el uso de los docentes es el siguiente:

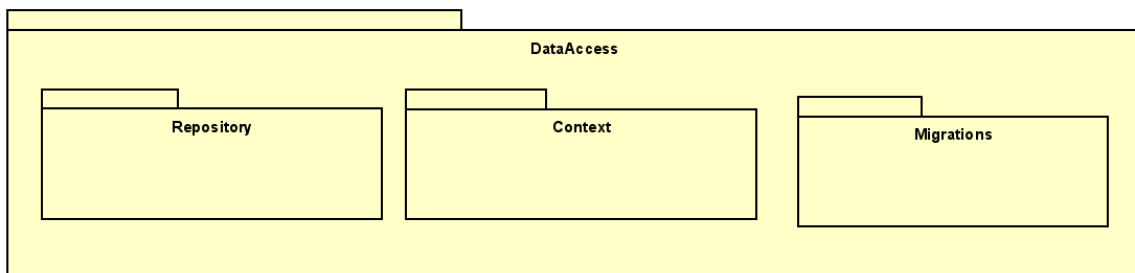
Email: **p@gmail.com**

Contraseña: **Passwd123#**

Descripción y justificación del diseño

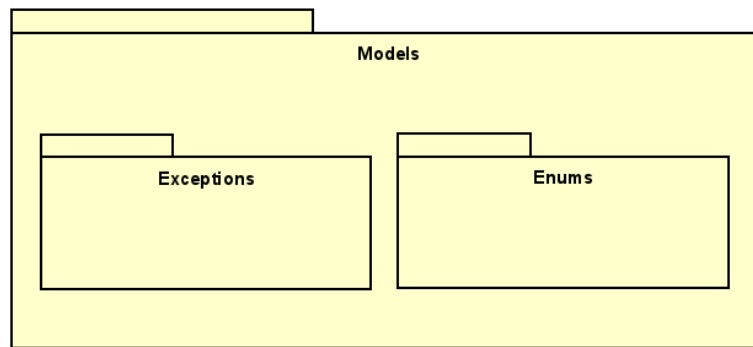
Diagramas de Paquetes

DataAccess



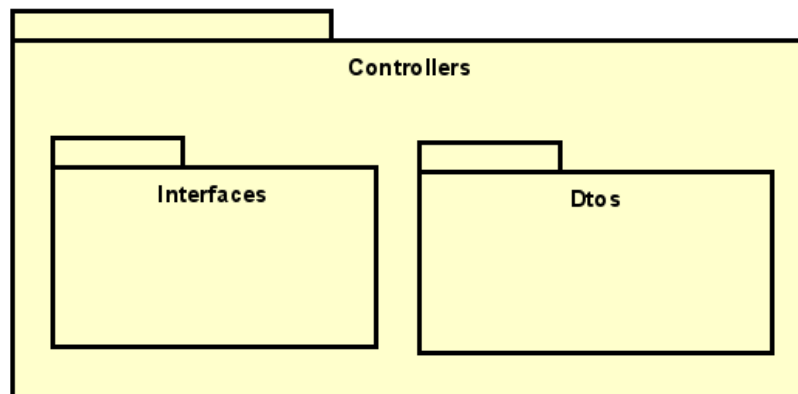
El paquete DataAccess consiste de persistencia y almacenamiento de datos. Es fundamental para la gestión y operación de los datos en la aplicación. Su principal responsabilidad es garantizar que los datos se almacenen de manera segura y que las operaciones CRUD (crear, leer, actualizar, eliminar) se realicen de manera eficiente. Se crean los contextos ApplicationDbContext y InMemoryAppContextFactory. El ApplicationDbContext es esencial para la creación y manejo del modelo de la base de datos. El InMemoryAppContextFactory se utiliza principalmente para pruebas. Este contexto permite crear una base de datos en memoria, lo que facilita la realización de pruebas unitarias y de integración sin la necesidad de interactuar con una base de datos física. Por otro lado, las migraciones son una herramienta esencial para gestionar los cambios en la estructura de la base de datos a lo largo del ciclo de vida de la aplicación. Cada vez que se introduce un cambio en el modelo de datos, como la adición de nuevas tablas, columnas o relaciones, se crea una migración que describe estos cambios. Finalmente los repositorios son los encargados de hacer las operaciones CRUD de los objetos, por lo tanto, el paquete DataAccess depende del paquete Models ya que los modelos definen la estructura de los datos que se persisten en la base de datos.

Model



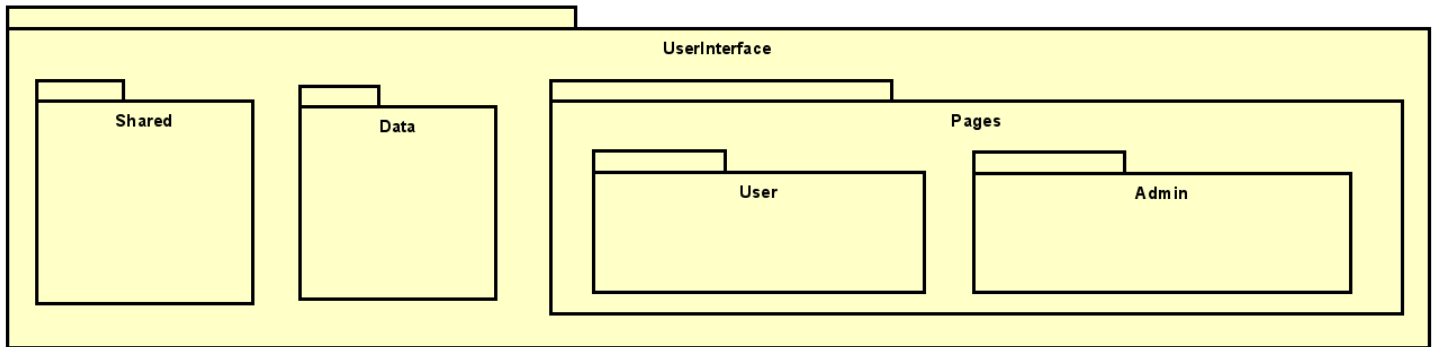
Este paquete es el Dominio de la aplicación. En el mismo se definen todas las clases que se utilizan dentro de la aplicación con sus atributos y métodos característicos. También contiene sub-paquetes como las excepciones que emplea la aplicación y los enums que son útiles para representar un conjunto fijo de constantes. Cabe destacar que este paquete no depende de ningún otro de la solución. Proporciona una base sólida sobre la cual se construyen y operan las demás partes de la aplicación.

Controllers



Este paquete actúa como el intermediario esencial entre el Dominio y la Interfaz de usuario de la aplicación, utilizando para ello los Data Transfer Objects (DTOs). En su diseño, se han creado nuevas interfaces y controladores que facilitan esta interacción. Entre estos componentes, el Controlador destaca como la clase más importante de toda la aplicación. Su principal responsabilidad es obtener los objetos DTO desde la interfaz de usuario cuando un usuario o administrador lleva a cabo una función específica. Una vez que el Controlador recibe el DTO, lo envía a la lógica correspondiente para que se realice el mapeo del DTO y se ejecute la lógica asociada. Este proceso es fundamental para garantizar que los datos se manejen de manera coherente y eficiente entre las diferentes capas de la aplicación. Cabe destacar que el Controlador se creó en base a los principios GRASP (General Responsibility Assignment Software Patterns) y las interfaces se crearon en base a los principios SOLID que son un conjunto de cinco principios. Estos principios son la Responsabilidad Única, Abierto/Cerrado, Sustitución de Liskov, Segregación de Interfaces y Inversión de Dependencia.

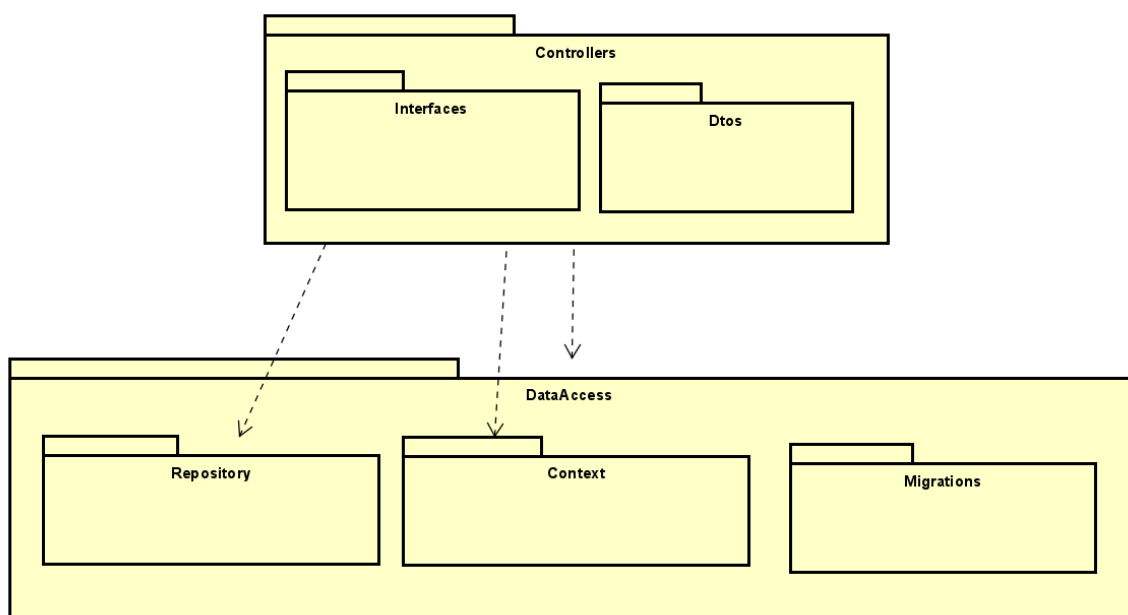
UserInterface



La principal responsabilidad de este paquete es almacenar toda la información de la página web y asegurar que las interfaces y sus funcionalidades se muestren al usuario de manera adecuada. Su objetivo es guiar al usuario de la mejor forma posible, evitando cualquier inconveniente en el uso de la aplicación. Este paquete contiene varios subpaquetes que organizan y gestionan diferentes aspectos de la interfaz de usuario. Uno de los subpaquetes más importantes es **Pages**, que se encarga de gestionar todas las páginas de la aplicación. Cada página representa una sección o funcionalidad específica de la aplicación, y **Pages** asegura que cada una se presente correctamente y funcione como se espera. Otro subpaquete esencial es **Shared**, que se ocupa de la navegabilidad y los estilos de las interfaces. Este subpaquete gestiona elementos comunes de la interfaz de usuario que son reutilizados en varias páginas, como menús de navegación, pie de página, encabezados y estilos visuales.

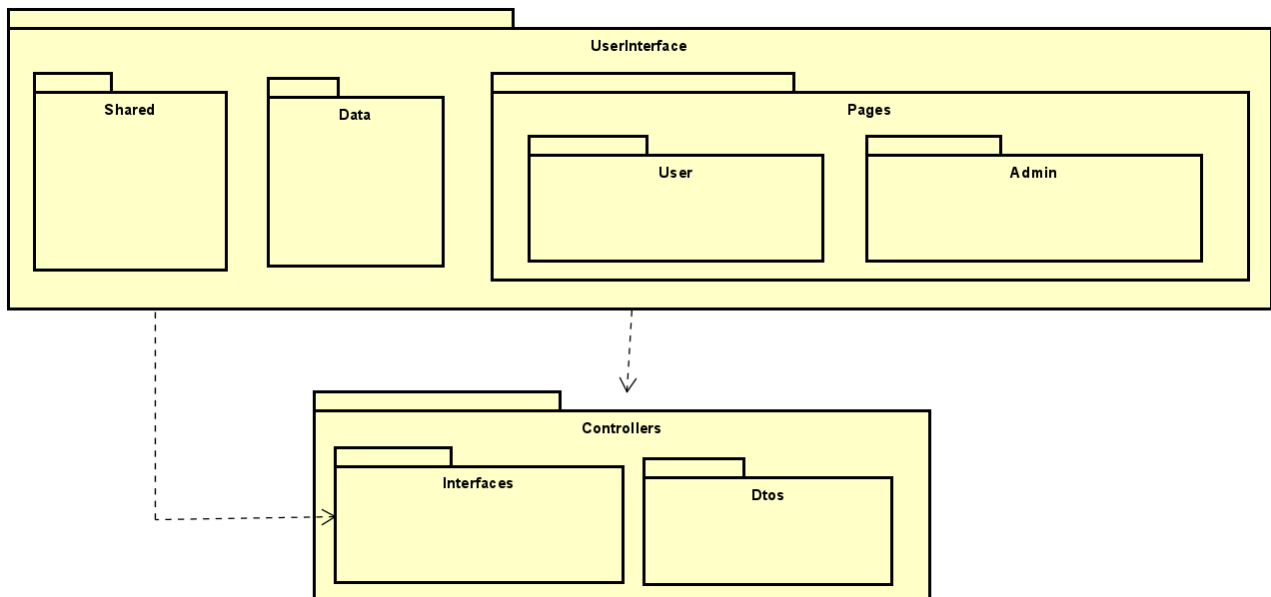
Dependencias entre paquetes

Dependencia entre Controllers-DataAccess



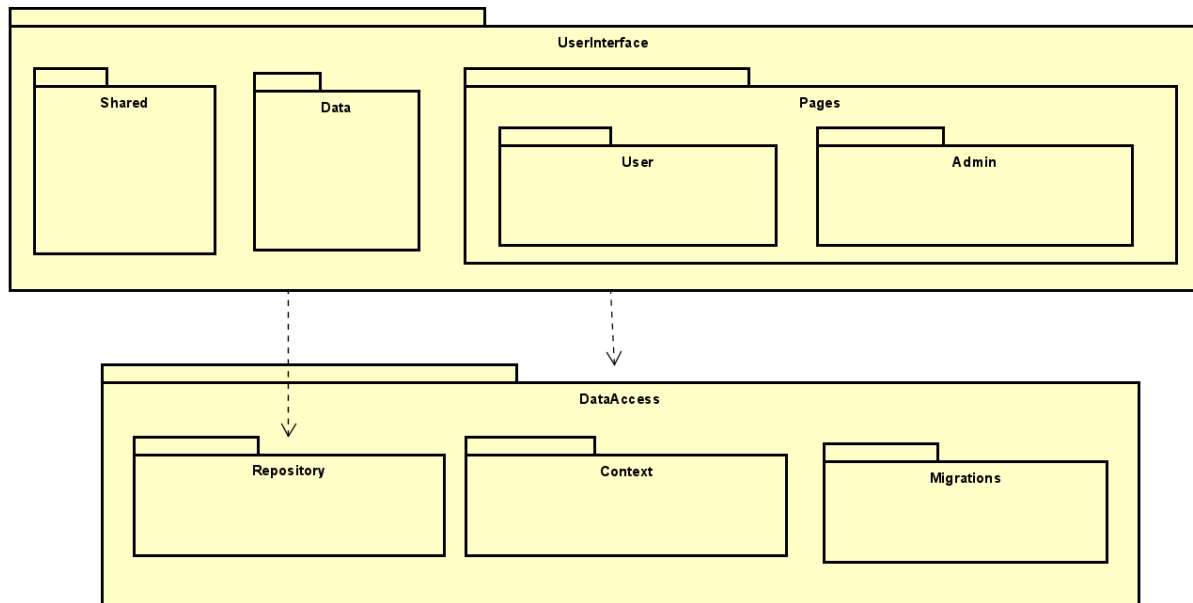
El paquete Controllers contiene la lógica de negocio, encargándose de coordinar y ejecutar las reglas y procesos fundamentales de la aplicación. Por otro lado, el paquete de DataAccess maneja el acceso a los datos, asegurándose de que la información se persista y se recupere de la base de datos de manera adecuada. Controllers depende de DataAccess para obtener los datos necesarios sin tener que preocuparse por los detalles específicos de persistencia. Esta separación de responsabilidades permite que los Controllers se concentren en la lógica de negocio, mientras que DataAccess se ocupa de las operaciones de bajo nivel relacionadas con el almacenamiento y recuperación de datos. El Context en esta estructura representa una sesión con la base de datos y se utiliza para interactuar con las entidades del dominio. Los Repositorios, que actúan como intermediarios entre los Controllers y el Context

Dependencia entre UserInterface-Controllers



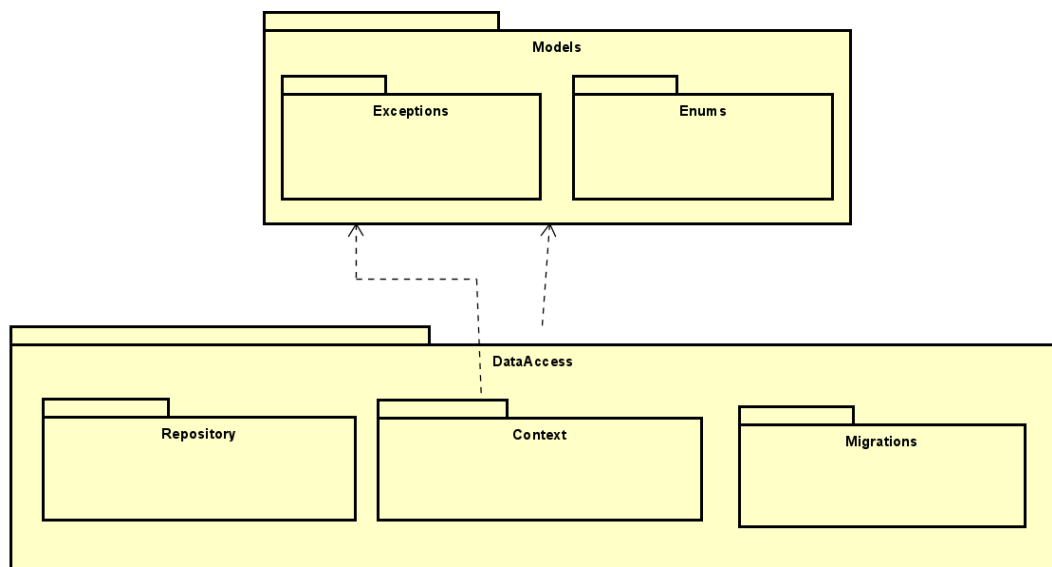
La dependencia entre la capa de UserInterface y el paquete Controllers es fundamental para el correcto funcionamiento de una aplicación. La UserInterface tiene la responsabilidad de mostrar la información al usuario y capturar sus interacciones. Por su parte, los Controllers manejan la lógica de la aplicación y la lógica de negocio. Cuando el usuario realiza una acción en la interfaz, como hacer clic en un botón o llenar un formulario, la UserInterface invoca métodos en los Controllers para procesar esa acción. Los Controllers reciben la solicitud y aplican la lógica de negocio necesaria, como validar datos, realizar cálculos o interactuar con otros servicios. Una vez procesada la acción, los Controllers actualizan el modelo de la aplicación y pueden modificar el estado de los datos. Luego, los Controllers devuelven la respuesta a la UserInterface, que puede consistir en una actualización de la vista con nueva información, la presentación de un mensaje de confirmación o error, o la navegación a una nueva pantalla. Esta interacción garantiza que la aplicación responda de manera coherente y eficiente a las acciones del usuario, proporcionando una experiencia de uso fluida y lógica.

Dependencia entre UserInterface-DataAccess



La dependencia del paquete **UserInterface** con el paquete **DataAccess** se establece a través de la inyección de dependencias de los repositorios definidos mediante el método **AddScoped**. Esta configuración es crucial para que la **UserInterface** pueda interactuar con la capa de datos de manera eficiente y sin acoplamiento directo.

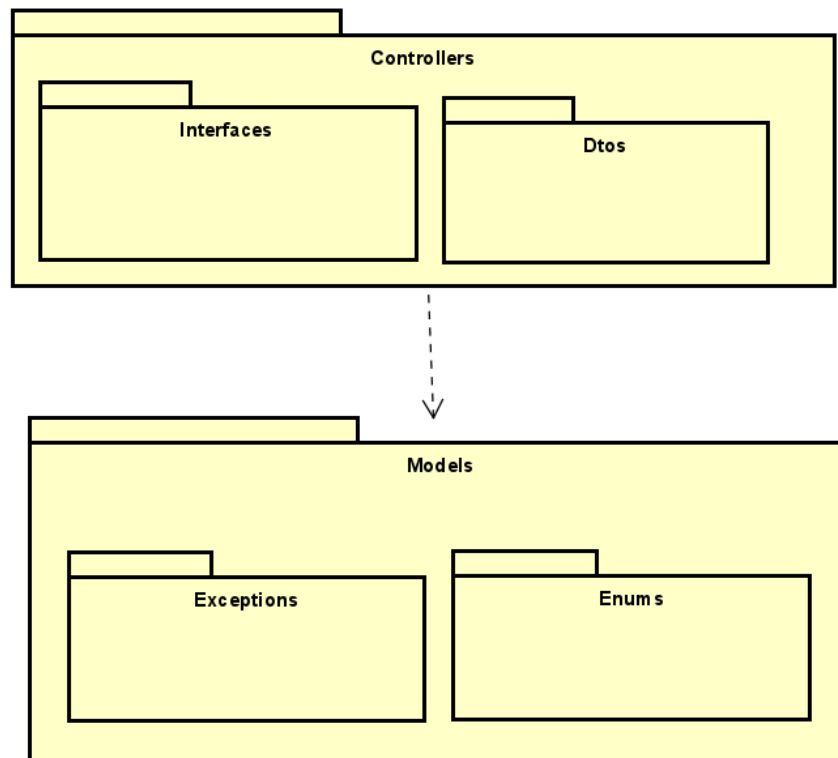
Dependencia entre DataAccess-Model



La dependencia del paquete **DataAccess** con el paquete **Models** es crucial para la correcta implementación y funcionamiento de la aplicación, especialmente en lo que respecta a la definición de contextos y la realización de operaciones CRUD. En el paquete **DataAccess**, los contextos se configuran utilizando **DbSet** para mapear cada objeto del dominio a sus correspondientes tablas en la base de datos. Este mapeo permite una representación clara

y directa de las clases del dominio en el almacenamiento persistente, facilitando así la gestión de los datos. Además, el paquete DataAccess depende del paquete Models para implementar los repositorios. Los repositorios utilizan estos modelos para llevar a cabo las operaciones CRUD sobre los objetos de la aplicación. Por ejemplo, un repositorio para una entidad Promotion proporcionará métodos para crear, leer, actualizar y borrar instancias de Promotion en la base de datos.

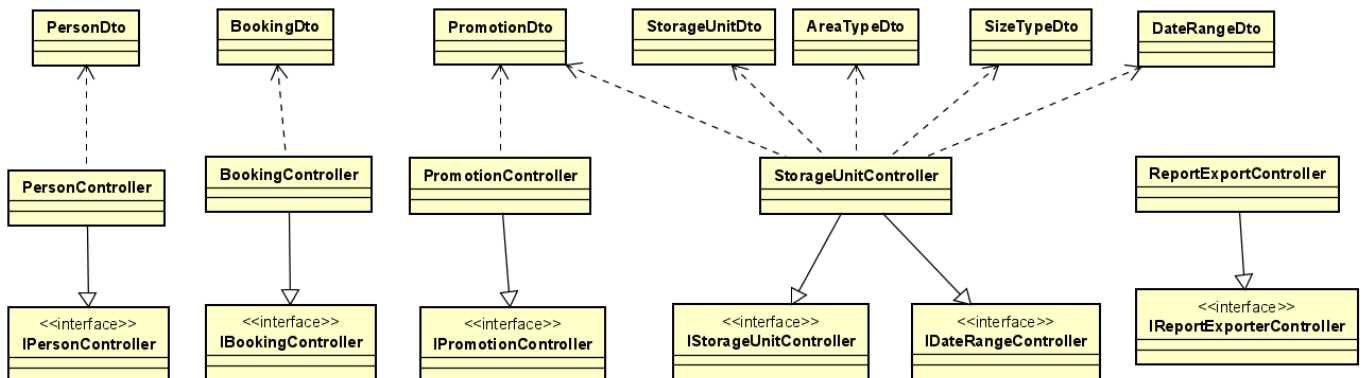
Dependencia Controllers - Model



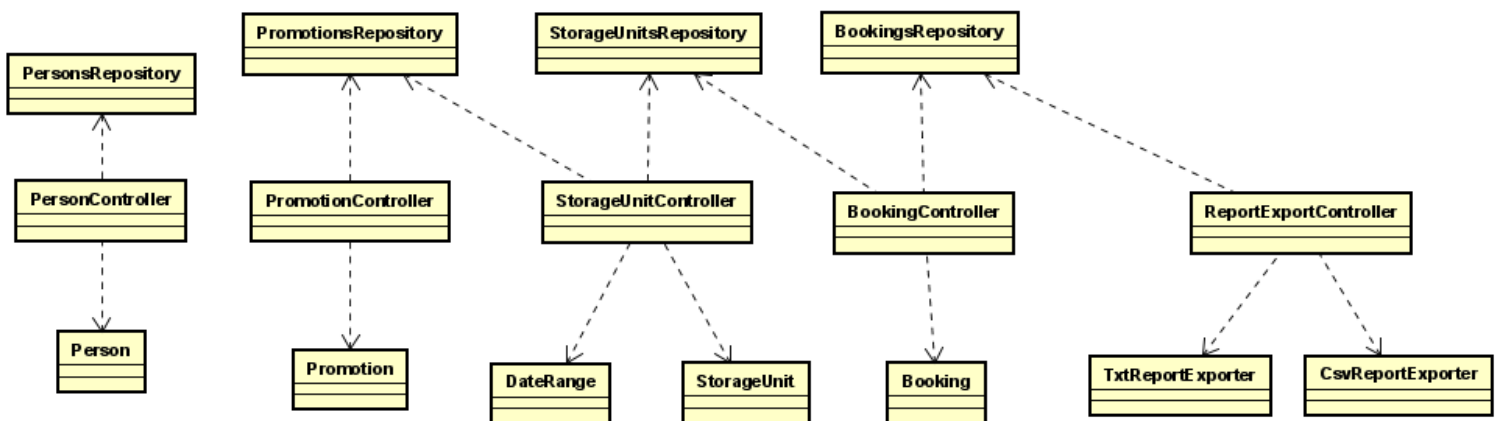
Los modelos de una aplicación no solo representan los datos, sino que también encapsulan la lógica de negocio oculta. Estas clases son esenciales para definir la estructura de los datos y las reglas que rigen su comportamiento en el contexto de la aplicación. Los controladores, por otro lado, gestionan el flujo de la aplicación y actúan como intermediarios entre las solicitudes del usuario y la lógica de negocio implementada en los modelos. Cuando un usuario interactúa con la aplicación, por ejemplo, al enviar un formulario o hacer clic en un botón, es el controlador correspondiente el que recibe y procesa esa entrada. Los controladores utilizan los modelos para recuperar, actualizar y procesar los datos según sea necesario para cumplir con la solicitud del usuario. En resumen, los controladores dependen de los modelos para manejar la lógica de negocio y los datos, y permiten la interacción del usuario con la aplicación.

Diagramas de Clases

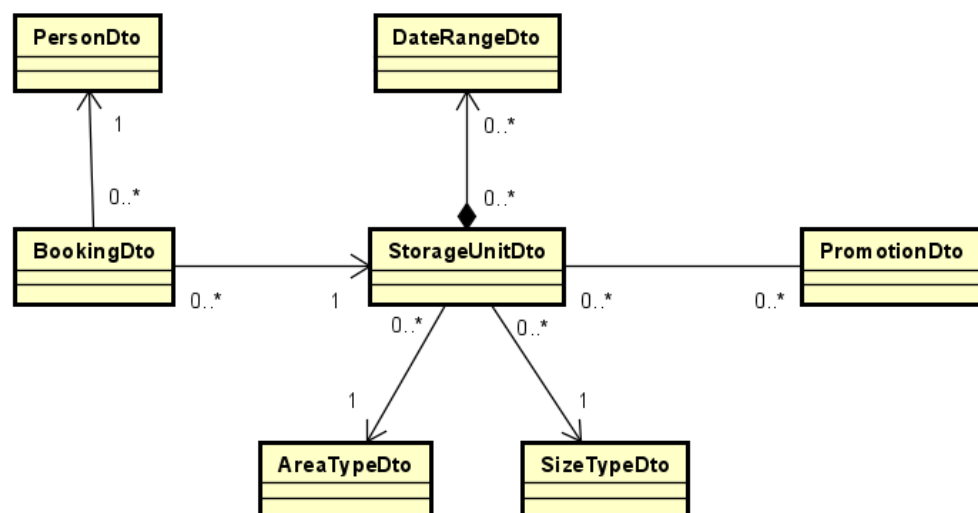
Controllers con sus Interfaces, Dtos



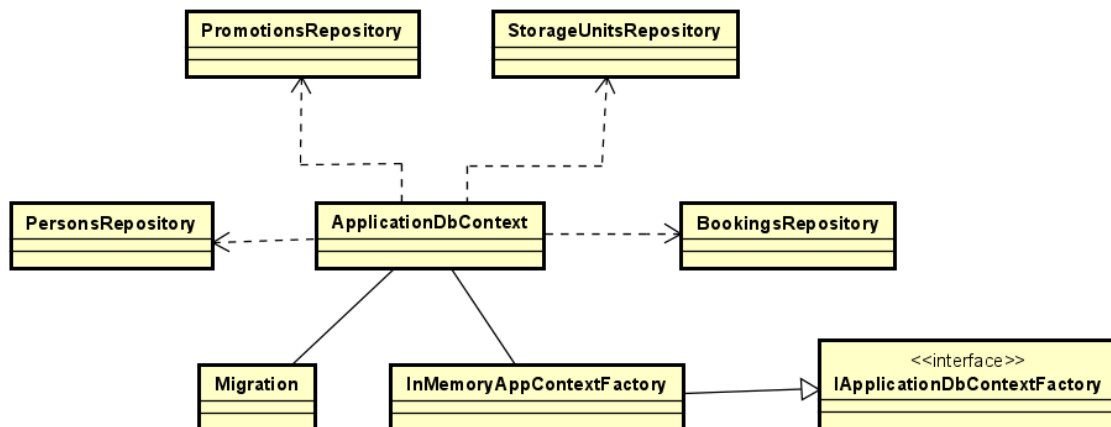
Controllers con Repository y Model



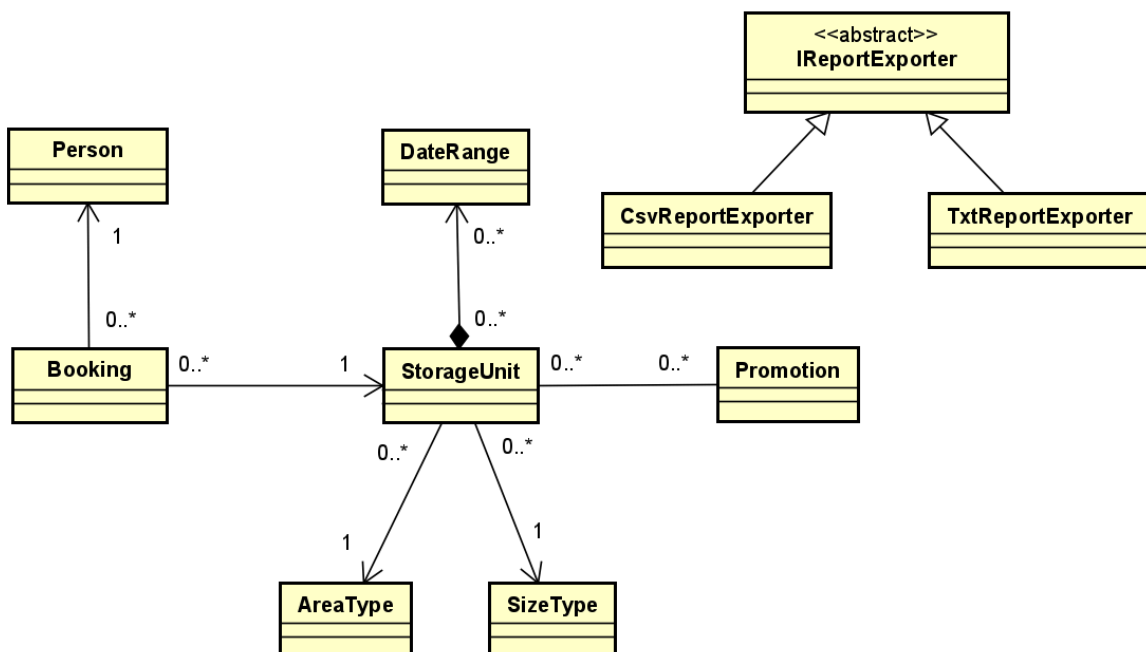
Dtos



DataAccess



Models



Diagramas de Interacción

Diagrama de secuencia Creación de una promoción

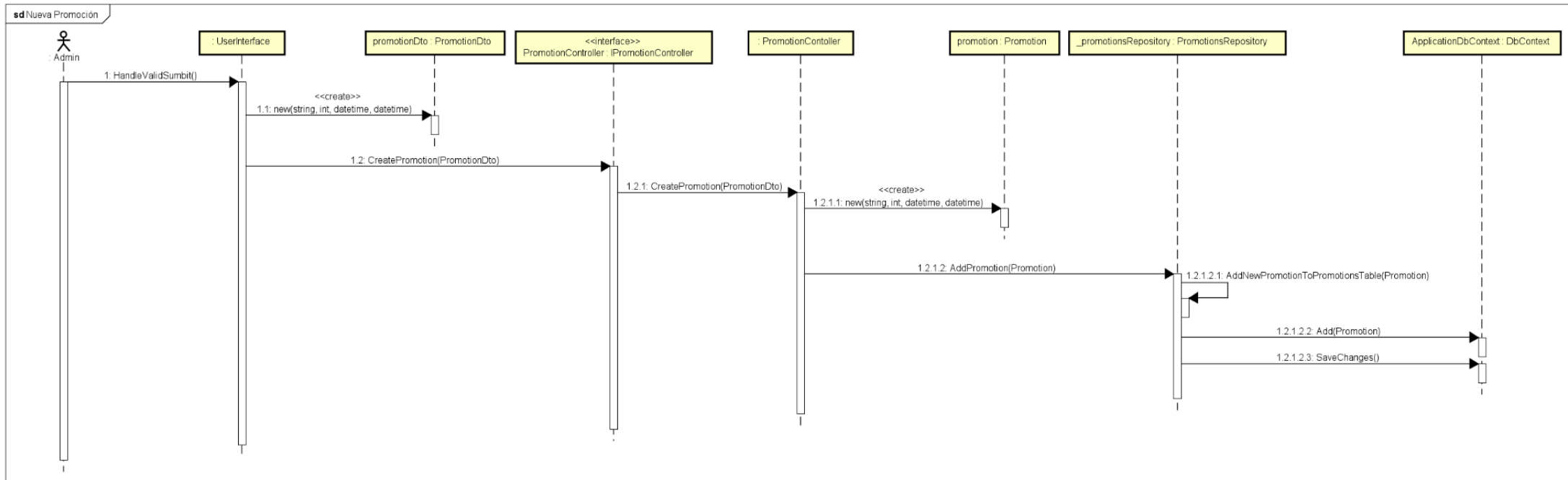


Diagrama de secuencia Exporte de un reporte de reservas

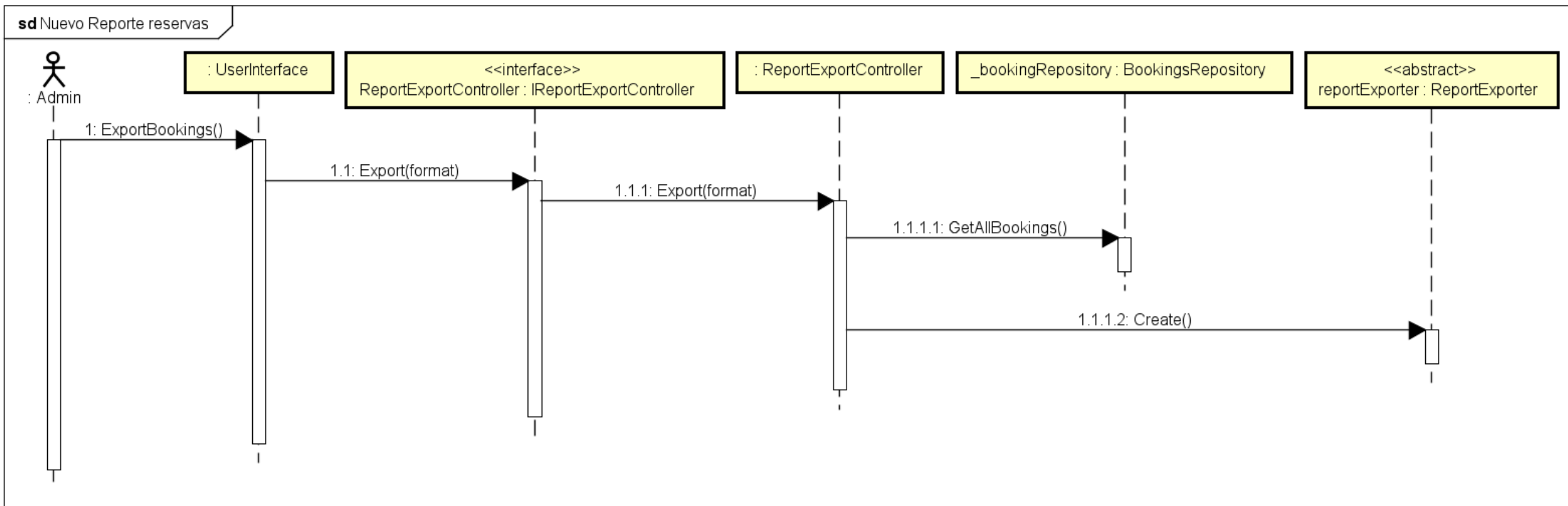
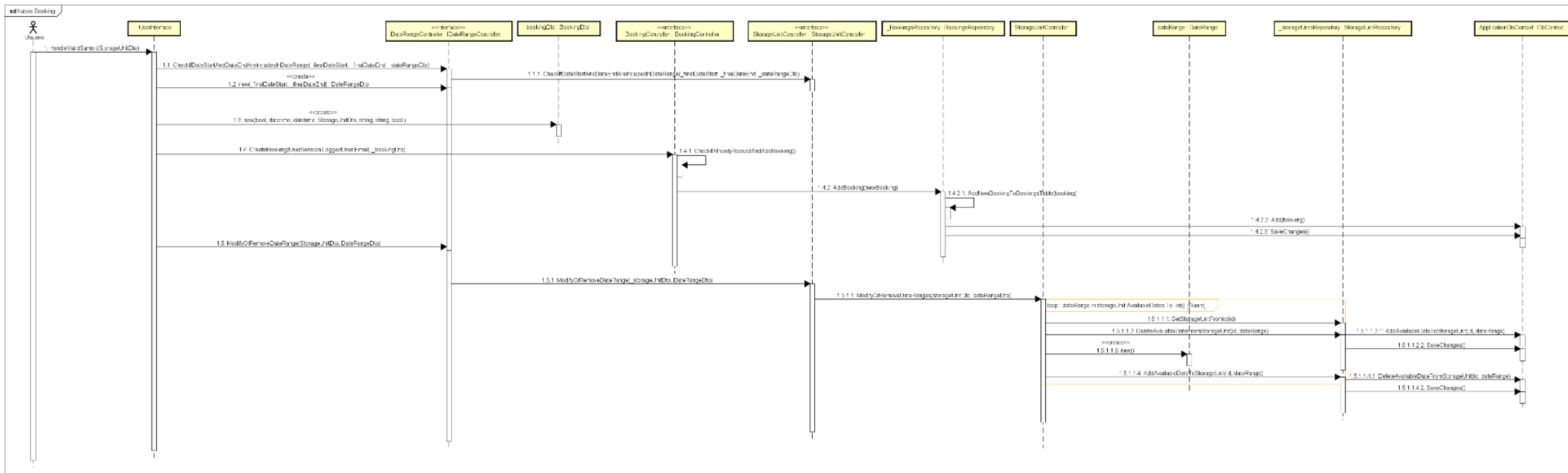
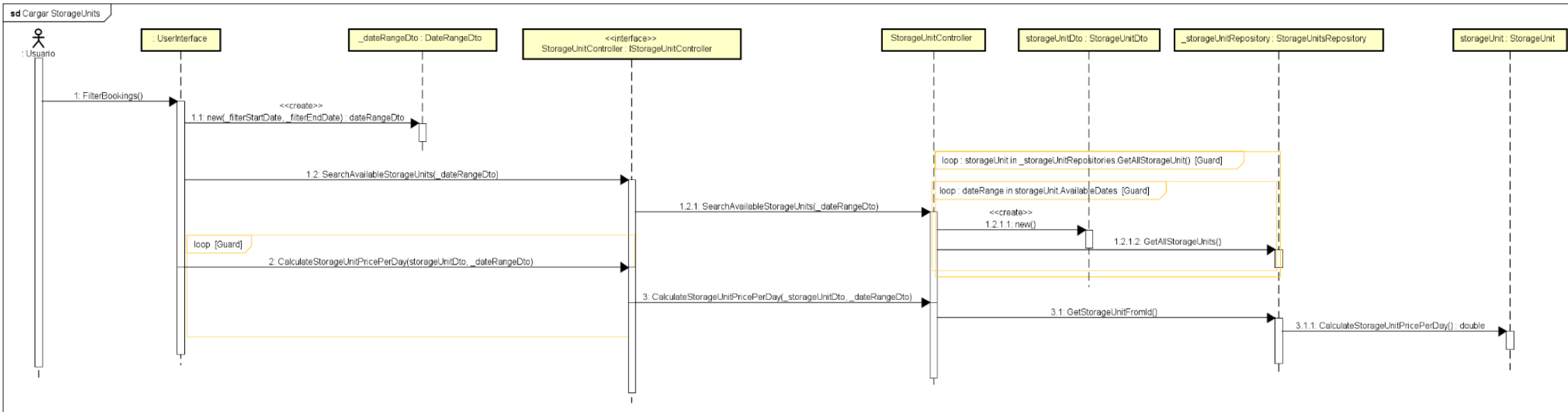


Diagrama de secuencia Usuario realiza un Booking



Los esquemas se encuentran en la carpeta de Diagrama de clases, dentro de la misma en la carpeta “Diagramas de Interacción” ya que en la imagen no se puede apreciar del todo bien.

También realizamos el método FilterBookings() que es el encargado de cargar los datos de los StorageUnit para que el Usuario pueda reservar a el deseado.



Modelo de Tablas (Imagen en el anexo)

En esta sección vamos a explicar cómo se generaron nuestras tablas de la base de datos utilizando Entity Framework. En este proyecto se aplicó la metodología Code-First la cual implica hacer el código de modelo de las clases del dominio (en nuestro caso usando C#) y EF Core se encarga de generar la base de datos a partir de mencionadas clases. Lo interesante de esto es que EF se encarga tanto de crear la base de datos, tanto como realizar las migraciones. Cabe destacar que todo se realiza de forma automática. En nuestro caso solamente tuvimos que asignar ciertos id que sean la primary key de algunas clases para poder migrarlas de forma correcta a la base de datos y además tuvimos que configurar a mano la relación “Many to many” de la promoción con el depósito. La configuración de esta última es un ejemplo de Fluent Api la cual permite configurar el comportamiento y las relaciones de las entidades en el método OnModelCreating dentro del DbContext.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Person>() // EntityTypeBuilder<Person>
        .HasKey(p:Person => p.Email);
    modelBuilder.Entity<Promotion>() // EntityTypeBuilder<Promotion>
        .HasKey(p:Promotion => p.Id);
    modelBuilder.Entity<DateRange>() // EntityTypeBuilder<DateRange>
        .HasKey(d:DateRange => d.Id);
    modelBuilder.Entity<Booking>() // EntityTypeBuilder<Booking>
        .HasKey(b:Booking => b.Id);

    modelBuilder.Entity<Promotion>() // EntityTypeBuilder<Promotion>
        .HasMany(navigationExpression: p:Promotion => p.StorageUnits) // CollectionNavigationBuilder<Promotion,StorageUnit>
        .WithMany(navigationExpression: s:StorageUnit => s.Promotions) // CollectionCollectionBuilder<StorageUnit,Promotion>
        .UsingEntity<Dictionary<string, object>>()
        {
            joinEntityName: "StorageUnitPromotion",
            configureRight: j:EntityTypeBuilder<Dictionary<...,>> => j
            {
                .HasOne<StorageUnit>()
                .WithMany()
                .HasForeignKey("StorageUnitId")
                .OnDelete(DeleteBehavior.Cascade),
            configureLeft: j:EntityTypeBuilder<Dictionary<...,>> => j
            {
                .HasOne<Promotion>()
                .WithMany()
                .HasForeignKey("PromotionId")
                .OnDelete(DeleteBehavior.Cascade),
            configureJoinEntityType: j:EntityTypeBuilder<Dictionary<...,>> =>
            {
                {
                    j.HasKey(params propertyNames: "StorageUnitId", "PromotionId");
                }
            }
        }
    };
}
```

Como se puede apreciar en la imagen adjunta en el ModelBuilder creamos una tabla intermedia entre promoción y un depósito utilizando el UsingEntity. Esta tabla se guarda como clave primaria compuesta las claves primarias de la promoción y del depósito (siendo estas claves foráneas de la tabla). Por lo tanto, con esta configuración al acceder a una instancia de StorageUnit, tendremos una lista de promociones asociadas a ese StorageUnit,

y viceversa, al acceder a una instancia de Promotion, obtendremos una lista de StorageUnit asociadas a esa promoción.

Mecanismos generales y decisiones de diseño

Interacción Interfaz-Dominio

La interfaz de usuario interactúa con el Dominio a través de los DTOs. Al realizar una funcionalidad de la aplicación ya sea por un usuario o administrador se crea un objeto DTO de la misma incluyendo los datos ingresados. Posteriormente se interactúa con la interfaz del controlador que le corresponde para que se lleven a cabo las operaciones de lógica de negocio necesarias en el objeto. Luego se comunica con el repositorio indicado para ese objeto y finalmente se agrega/elimina/modifica en la Base de Datos.

Almacenamiento de Datos

En esta entrega se hizo hincapié en mantener la persistencia de los datos cuando un usuario o administrador interactúa con la interfaz de usuario. Para lograr esto se trabajó con Entity Framework haciendo instancia de su herramienta fundamental “DbContext”. Con esta herramienta los repositorios interactúan con el contexto y se migran nuestros objetos con sus datos ingresando de forma automática a la base de datos utilizando los “DbSet”. Esto nos ahorra a nosotros como programadores no tener que crear la base de datos desde cero. Para la administración de nuestra base de datos utilizamos la aplicación recomendada por el cliente “DBeaver”.

Manejo de Errores y Excepciones

Para gestionar adecuadamente el manejo de errores y excepciones para todas las clases del dominio, se crearon las respectivas excepciones para los supuestos ingresos de datos erróneos de un usuario promedio. En la interfaz de usuario se ingresaron las sentencias try-catch para excepciones específicas. Al capturar las excepciones, la aplicación puede mostrar mensajes de error significativos y amigables para el usuario, evitando así interrumpir la fluidez de la navegación y mejorando la experiencia del usuario en la aplicación. Este enfoque asegura que los usuarios reciban retroalimentación inmediata y clara sobre cualquier problema con los datos ingresados, permitiéndoles corregir los errores sin tener que abandonar el flujo actual de la aplicación.

Utilización de GRASP y SOLID

Por parte del uso GRASP tenemos implementado la metodología **Single Responsibility Principle (SRP)**. Este principio establece que una clase debería tener solo una razón para cambiar. Al implementar el patrón del controlador, estamos separando las responsabilidades de manejo de la interfaz de usuario de las responsabilidades de procesamiento de datos. Esto significa que cada controlador tiene una única responsabilidad: manejar las solicitudes de entrada. Por otro lado, para el uso de SOLID tenemos implementado la metodología **Interface Segregation Principle (ISP)**. Este principio dice que ningún cliente debería depender de métodos que no utiliza. Al crear interfaces que utiliza el controlador para comunicarse con otros componentes del sistema, estamos siguiendo este principio. Esto

permite que los componentes del sistema dependen solo de las interfaces que necesitan y no de funcionalidades adicionales.

Criterios seguidos para asignar responsabilidades

En esta entrega, se han asignado responsabilidades específicas tanto a los controladores como a los repositorios, con el objetivo de estructurar y organizar de manera eficiente el flujo de trabajo y la interacción con la base de datos. Los controladores desempeñan un papel fundamental al actuar como intermediarios entre la interfaz de usuario y la lógica de negocio. Se encargan de procesar las solicitudes provenientes de la interfaz y de invocar la lógica de negocio correspondiente para manejar esas solicitudes de manera adecuada. La lógica de negocio, por su parte, es responsable de implementar las reglas y procesos que gobiernan el funcionamiento del sistema. Esta capa interactúa directamente con los repositorios para llevar a cabo operaciones relacionadas con la persistencia de datos. Al delegar estas responsabilidades a la lógica de negocio, se asegura que las reglas y validaciones del dominio sean consistentes y estén centralizadas en un solo lugar. Los repositorios juegan un rol crucial en la gestión de datos. Su principal función es comunicarse con el contexto de la base de datos para realizar las operaciones de acceso y manipulación de datos. Esto incluye la inserción, actualización, eliminación y consulta de registros en las tablas de la base de datos. Al utilizar repositorios, se abstraen los detalles de acceso a datos, permitiendo que la lógica de negocio y los controladores se centren en sus responsabilidades sin preocuparse por la implementación específica de la persistencia de datos. En conjunto, esta arquitectura facilita la separación de preocupaciones, mejora la mantenibilidad del código y permite una evolución más sencilla del sistema. Los controladores gestionan las interacciones de la interfaz, la lógica de negocio asegura el cumplimiento de las reglas del dominio, y los repositorios se ocupan de las tareas de persistencia, todo esto trabajando de manera coordinada para garantizar un flujo de datos coherente y eficiente en toda la aplicación.

Análisis de dependencias

Para la funcionalidad de "Exportar reporte de reservas", se ofrece el análisis siguiente:

Clases involucradas:

- **Booking** : Representa la reserva del usuario. Contiene como atributos: estado de aprobación, fechaInicio, fechaFin, depósito, estado de la reserva, estado del pago, mensaje de rechazo y email del usuario que la realiza.
- **StorageUnit**: Depósito asociado a la reserva que se realiza. Contiene como atributos: área, tamaño, climatización, lista de promociones (puede no tener), lista de rangos de fecha (al menos el de la reserva).
- **Promotion**: Si el depósito involucrado en la reserva tiene promoción esto se refleja en el costo total-final de la reserva. Contiene como atributos: etiqueta, porcentaje del descuento, fechaInicio, fechaFin y lista de depósitos a los cuales pertenece.
- **ReportExporter** (y sus implementaciones **CsvReportExporter**, **TxtReportExporter**): Representa el reporte a ser generado, con métodos para agregar datos de reservas y formatear el contenido.

- ReportExportController: Clase responsable de la lógica de exportación del reporte en diferentes formatos (Csv o Txt).
- BookingsRepository: Clase responsable de agregar/eliminar/modificar/obtener reservas de la base de datos.

Dependencia:

ReportExportController depende de:

- BookingsRepository para obtener las reservas.
- ReportExporter para exportar las reservas en el formato adecuado.

ReportExporter (abstracto) y sus implementaciones (CsvReportExporter, TxtReportExporter) dependen de:

- La lista de Booking que reciben como parámetro en el método Export y en el método GetData.

Booking depende de:

- StorageUnit para identificar la unidad de almacenamiento asociada.
- Promotion para aplicar descuentos.

Cohesión:

- ReportExportController: Alta cohesión, ya que se enfoca únicamente en manejar la exportación de reportes.
- BookingsRepository: Alta cohesión, responsable de las operaciones de persistencia y recuperación de Booking.
- ReportExporter: Alta cohesión, responsable de exportar datos de reservas en diferentes formatos.
- Booking: Alta cohesión, representa una reserva con todos sus atributos relacionados.
- StorageUnit: Alta cohesión, representa una unidad de almacenamiento con sus atributos.
- Promotion: Alta cohesión, representa una promoción con sus atributos y asociaciones.

Acoplamiento:

- ReportExportController y BookingsRepository: El acoplamiento podría reducirse utilizando una interfaz para BookingsRepository.
- ReportExportController y ReportExporter: Acoplamiento necesario, pero está gestionado adecuadamente.
- Booking y StorageUnit: Dependencia necesaria para representar la relación entre reservas y unidades de almacenamiento.
- StorageUnit y Promotion: Dependencia necesaria para representar las promociones asociadas a las unidades de almacenamiento.

Cobertura de pruebas unitarias (Imagen en el anexo)

A lo largo de este proyecto se incentivó el uso de la metodología TDD (Test-Driven Development), la cual se basa en realizar tests previos a la codificación final. En esta entrega, se solicitaba agregar nuevos requerimientos funcionales, lo que implicó realizar algunos refactores en los tests ya creados previamente. En cuanto a la lógica del proyecto, se eliminó la parte de memoria y se adaptaron los tests existentes a la nueva estructura del sistema con base de datos. Para lograr una adecuada adaptación, se crearon nuevas interfaces y repositorios de la base de datos, lo que permitió una mejor gestión y organización del código. Asimismo, se desarrollaron nuevos tests tanto para los repositorios como para los controladores. Estos tests aseguraron que las nuevas funcionalidades cumplieran con los requisitos establecidos y que el sistema mantenía su integridad y funcionalidad. La metodología TDD no solo nos permitió desarrollar un código más robusto y confiable, sino que también facilitó la identificación y corrección de errores en etapas tempranas del desarrollo. Esto contribuyó a una mayor calidad del software y a una mejor mantenibilidad del mismo. Además, al refactorizar los tests existentes y crear nuevos, nos aseguramos de que todas las partes del sistema estaban correctamente integradas y funcionaban como se esperaba. La integración de TDD en este proyecto ha demostrado ser una estrategia efectiva para garantizar la calidad del software y la satisfacción de los requerimientos funcionales. La constante verificación mediante tests nos ha permitido construir un sistema sólido, adaptable y con un alto nivel de confiabilidad.

Datos precargados en la base de datos (En carpeta en el repo)

Usuarios:

Nombre y Apellido: p p
Correo: p@gmail.com
Contraseña: Passwd123#
isAdmin: Yes

Nombre y Apellido: Federico Gutierrez
Correo: fede@gmail.com
Contraseña: Fede2023#
isAdmin: No

Nombre y Apellido: Franco Ramos
Correo: francoramos@gmail.com
Contraseña: FrancoRamos2023#
isAdmin: No

Nombre y Apellido: Gaston Mousques
Correo: gaston@gmail.com
Contraseña: Gaston2023#

isAdmin: No

Nombre y Apellido: Juan Perez
Correo: juanperez@gmail.com
Contraseña: Juan2023#
isAdmin: No

Nombre y Apellido: Maria Garcia
Correo: mariagarcia@gmail.com
Contraseña: Maria2023#
isAdmin: No

Promociones:

Label - StartDate - EndDate - Discount

Winter Discount - 09/06/2024 - 29/06/2024 - 30
Summer Discount - 01/01/2024 - 01/15/2024 - 20
Spring Discount - 10/09/2024 - 30/09/2024 - 35
Weekend Discount 07/06/2024 - 09/06/2024 - 40
Bonus Discount 01/03/2024 - 10/03/2024 - 25
Day Discount - 01/03/2024 - 02/03/2024 - 15
Christmas Discount - 24/12/2024 - 25/12/2024 - 45
Anniversary Discount - 15/11/2024 - 30/11/2024 - 50
Autumn Discount - 01/10/2024 - 29/10/2024 - 25
Exclusive Discount - 13/05/2024 - 19/05/2024 - 30

Depósitos:

Id - Area - Size - Climatization

13 - E - Small - Yes
15 - B - Large - Yes
27 - D - Large - No
3 - A - Small - No
7 - D - Medium - Yes
9 - C - Medium - No

Depósitos con Promoción:

StorageUnitId - PromotionId

7 - 1
27 - 9

27 - 3
13 - 9
3 - 4

Rangos de fecha:

StorageUnitId - StartDate - EndDate

13 - 09/06/2024 - 29/06/2024
15 - 09/06/2024 - 30/06/2024
27 - 05/10/2024 - 15/10/2024
27 - 07/06/2024 - 06/06/2024
3 - 01/03/2024 - 15/03/2024
7 - 09/06/2024 - 29/06/2024
9 - 01/03/2024 - 30/03/2024
9 - 05/05/2024 - 30/06/2024

Reservas:

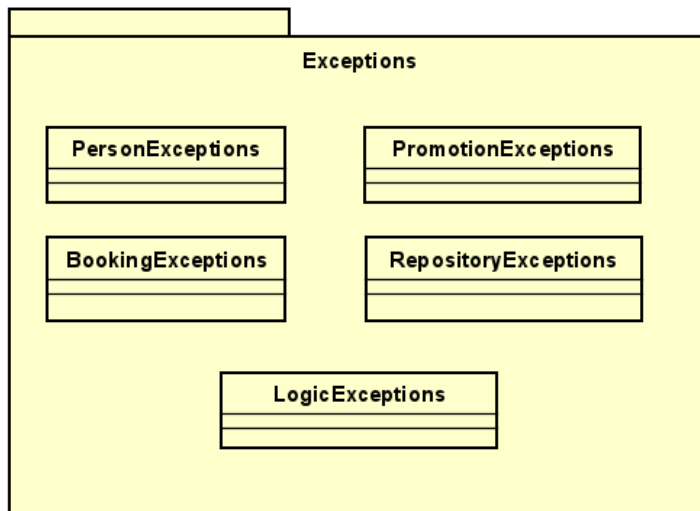
Approved - PersonEmail - DateStart - DateEnd - StorageUnitId - RejectedMessage - Status - Payment

Yes - mariagarcia@gmail.com - 09/06/2024 - 12/06/2024 - 13 - "" - Capturado - Yes
No - mariagarcia@gmail.com - 09/06/2024 - 12/06/2024 - 15 - "" - Reservado - No
No - mariagarcia@gmail.com - 09/06/2024 - 12/06/2024 - 7 - "rechazado por incompletitud" - Rechazado - Yes
No - mariagarcia@gmail.com - 09/06/2024 - 12/06/2024 - 9 - "" - Reservado - No
No - juanperez@gmail.com - 05/10/2024 - 06/10/2024 - 27 - "" - Reservado - Yes
No - gaston@gmail.com - 01/03/2024 - 06/03/2024 - 3 - "" - Reservado - No
No - gaston@gmail.com - 01/03/2024 - 06/03/2024 - 9 - "" - Reservado - Yes
No - juanperez@gmail.com - 13/06/2024 - 16/06/2024 - 13 - "a" - Rechazado - Yes
No - juanperez@gmail.com - 13/06/2024 - 16/06/2024 - 15 - "" - Reservado - No
Yes - juanperez@gmail.com - 13/06/2024 - 16/06/2024 - 7 - "" - Capturado - Yes
No - juanperez@gmail.com - 13/06/2024 - 16/06/2024 - 9 - "" - Reservado - No
No - fede@gmail.com - 05/05/2024 - 10/05/2024 - 9 - "" - Reservado - Yes
No - fede@gmail.com - 07/10/2024 - 09/10/2024 - 27 - "" - Reservado - No
No - fede@gmail.com - 18/06/2024 - 20/06/2024 - 13 - "rechazado" - Rechazado - Yes
No - fede@gmail.com - 18/06/2024 - 20/06/2024 - 15 - "" - Reservado - No
No - fede@gmail.com - 18/06/2024 - 20/06/2024 - 7 - "rejected" - Rechazado - Yes
No - francoramos@gmail.com - 21/06/2024 - 23/06/2024 - 13 - "" - Reservado - Yes
No - francoramos@gmail.com - 21/06/2024 - 23/06/2024 - 15 - "" - Reservado - No
Yes - francoramos@gmail.com - 21/06/2024 - 23/06/2024 - 7 - "" - Capturado - Yes
No - francoramos@gmail.com - 21/06/2024 - 23/06/2024 - 9 - "" - Reservado - No

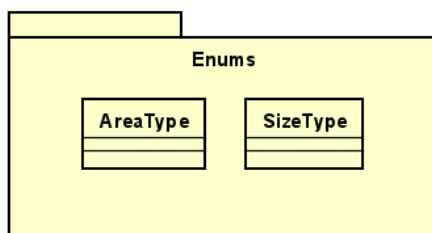
Anexo

Diagrama de Paquetes

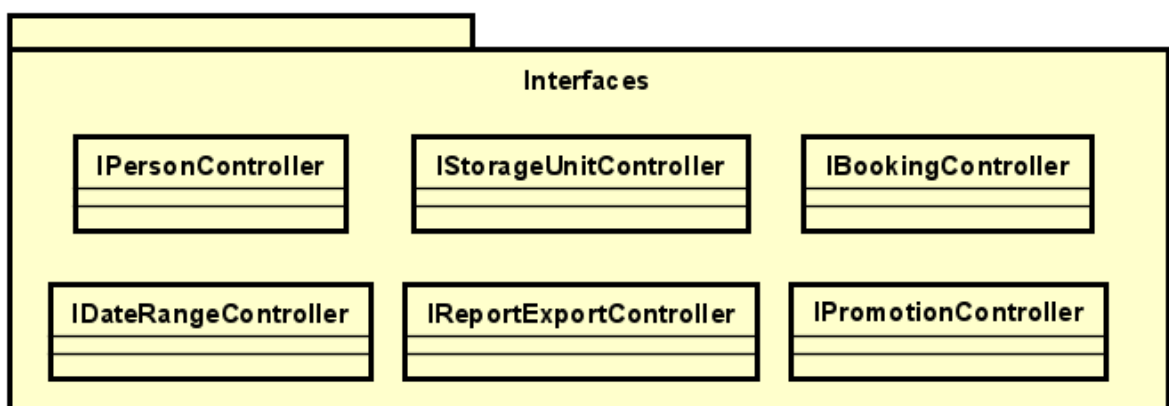
Paquete Exceptions



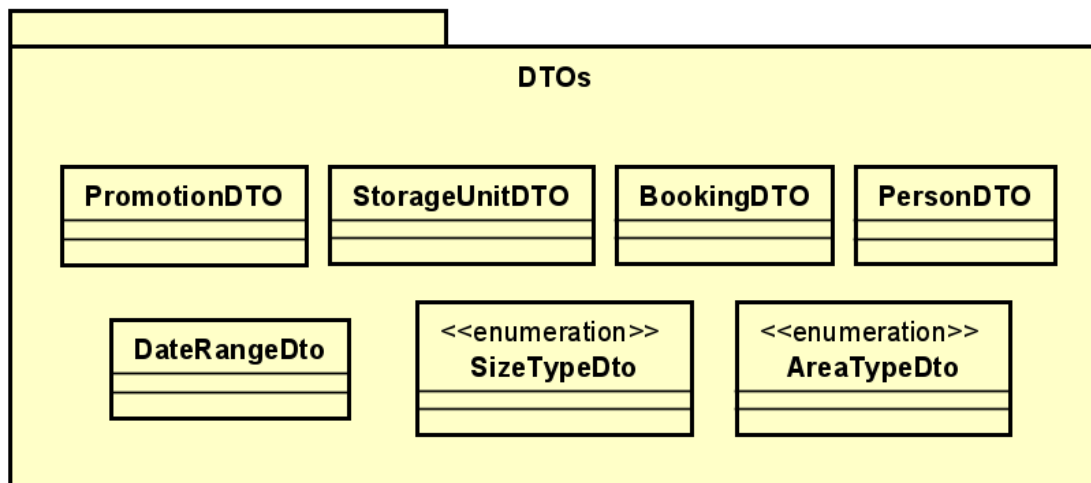
Paquete Enums



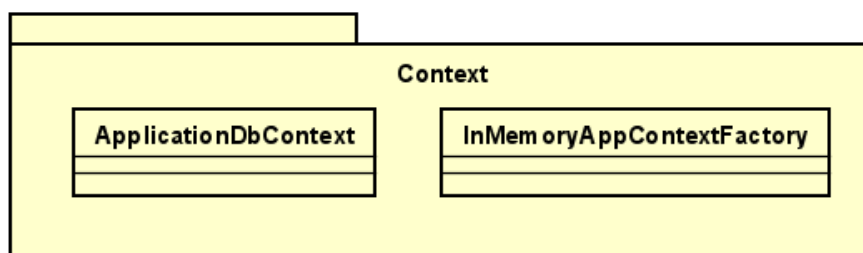
Paquete Interfaces



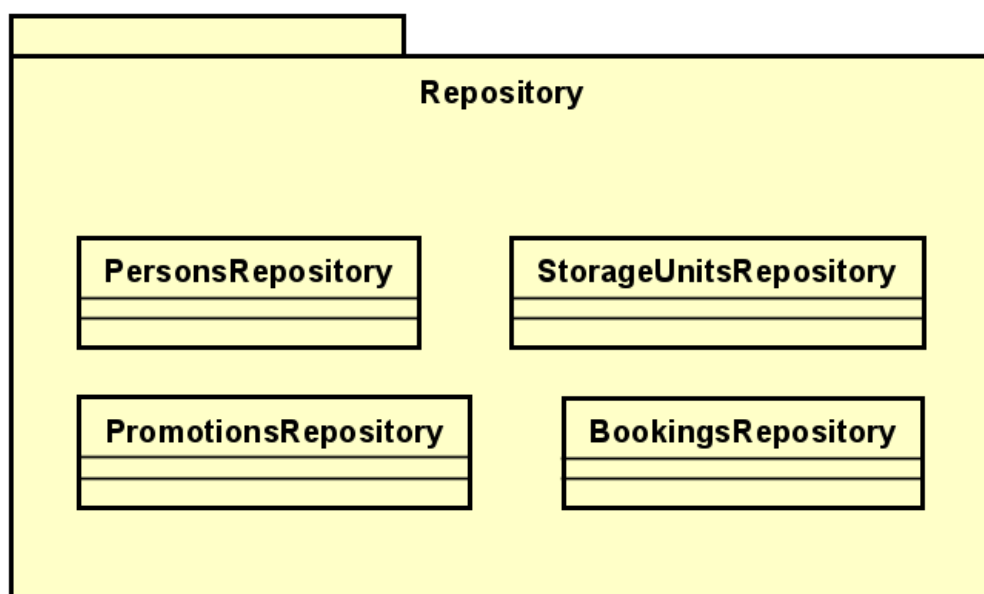
Paquete Dtos



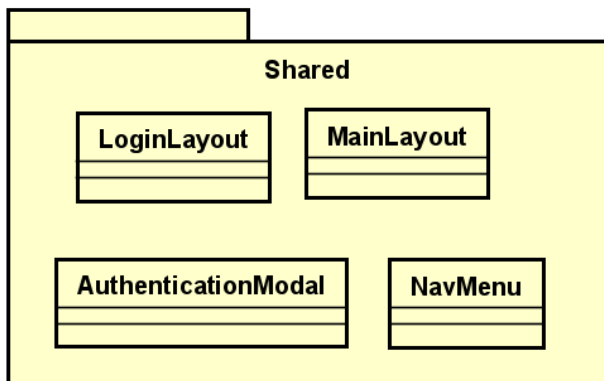
Paquete Context



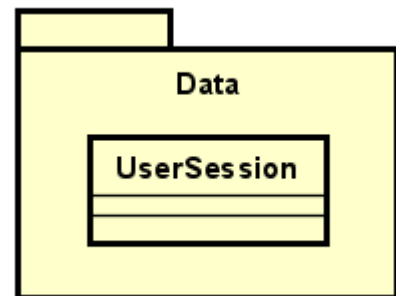
Paquete Repository



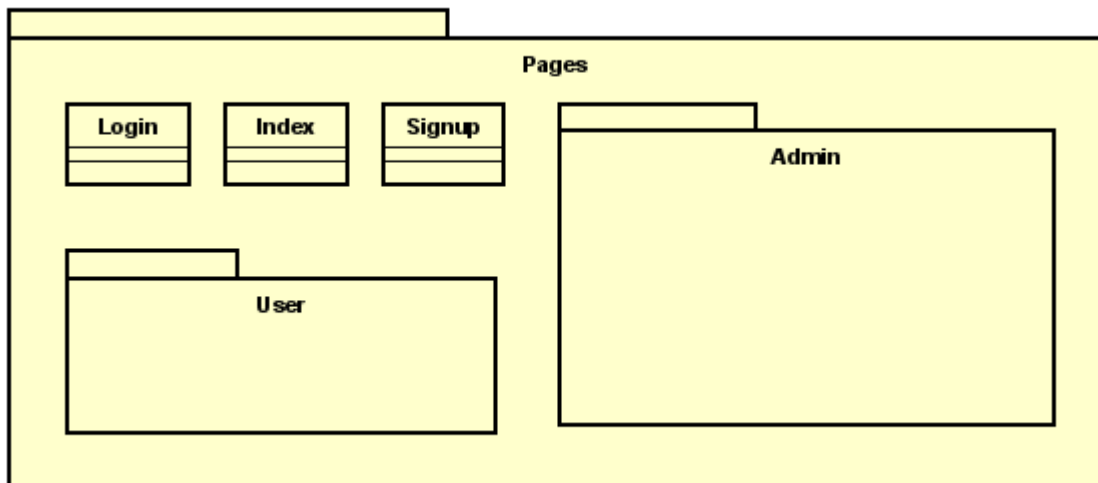
Paquete Shared



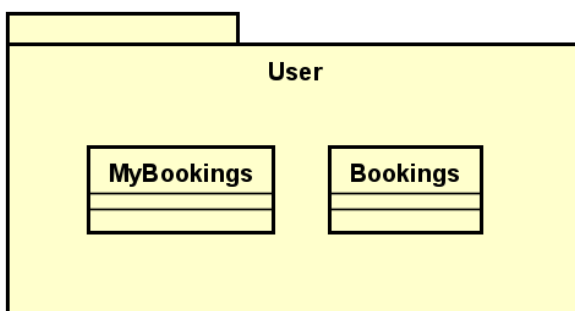
Paquete Data



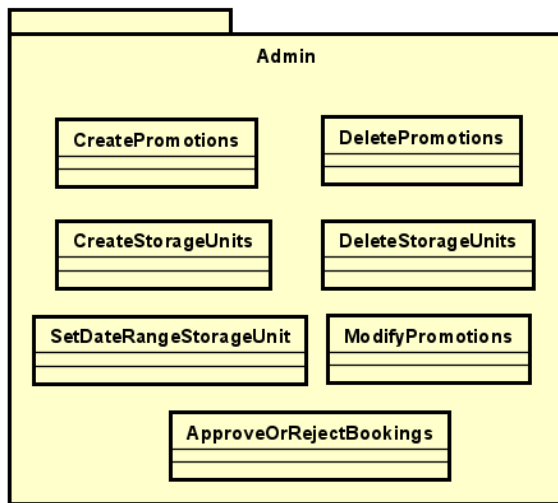
Paquete Pages



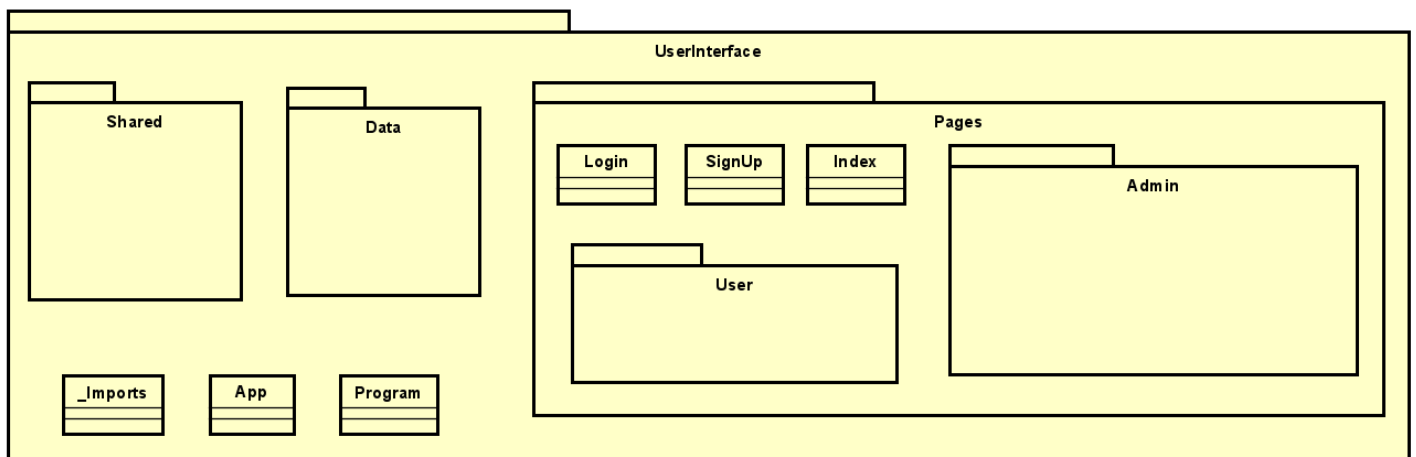
Paquete User



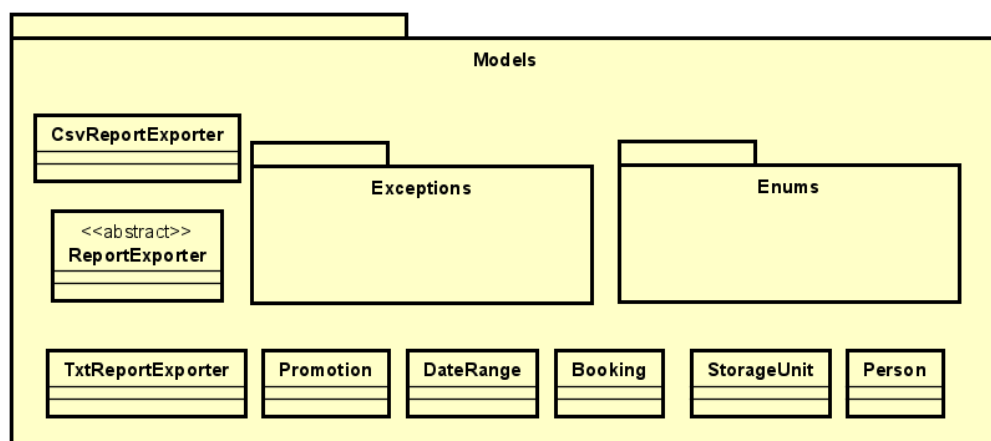
Paquete Admin



Paquete UserInterface



Paquete Model



Paquete Controllers

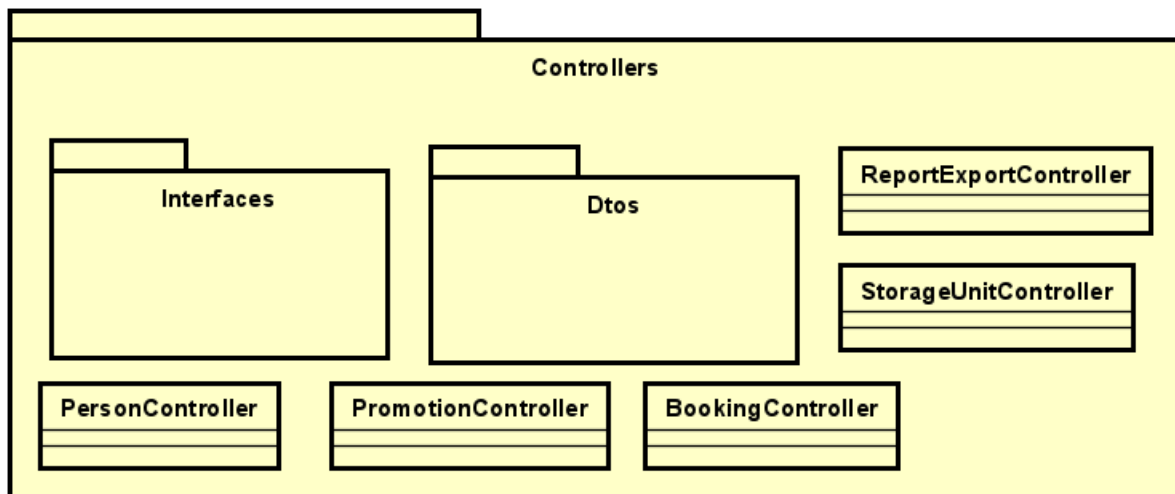
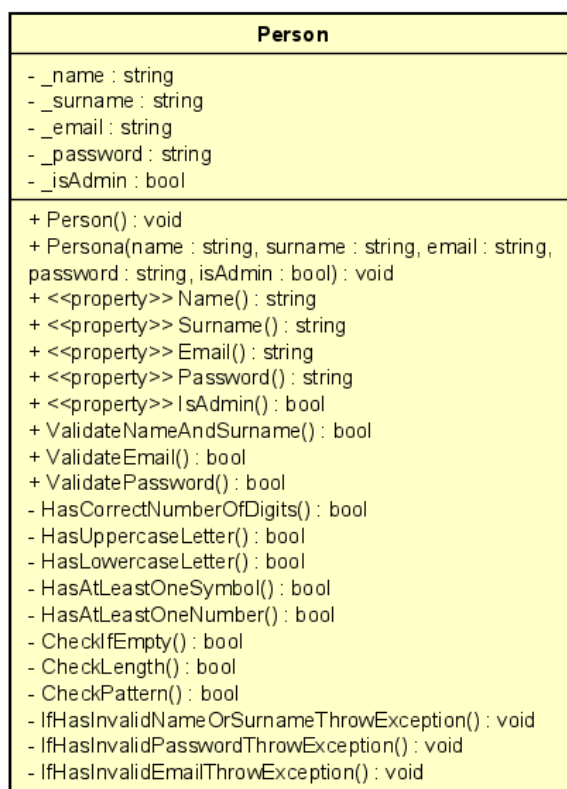


Diagrama de Clases

Clase Person



Clase Promotion

Promotion
<ul style="list-style-type: none"> - Id : int - _label : string - _discount : int - _dateStart : DateTime - _dateEnd : DateTime
<ul style="list-style-type: none"> + Promotion() : void + Promotion(string label : int, int discount : int, dateStart : DateTime, dateEnd : DateTime) : void + <<property>> Label() : string + <<property>> Discount() : int + <<property>> DateStart() : DateTime + <<property>> DateEnd() : DateTime + ValidateLabel() : bool + ValidateDiscount() : bool + ValidateDate() : bool - IfHasInvalidLabelThrowException() : void - IfHasInvalidDiscountThrowException() : void - IfHasInvalidDateThrowException() : void

Clase StorageUnit

StorageUnit
<ul style="list-style-type: none"> - _id : string - <<enumeration>> _area : AreaType - <<enumeration>> _size : SizeType - _climatization : bool - List<Promotion> _promotions : Promotion - List<DateRange> _availableDates : DateRange
<ul style="list-style-type: none"> + StorageUnit() : void + StorageUnit(area : SizeType, size : int, climatization : bool, promotions : List<Promotion>, availableDates : List<DateRange>) : void + <<property>> Id() : string + <<property>> Area() : AreaType + <<property>> Size() : SizeType + <<property>> Climatization() : bool + <<property>> Promotions() : List<Promotion> + <<property>> AvailableDates() : List<DateRange> + CalculateStorageUnitPricePerDay() : double - RuleOf3() : double - ValueOfSizeOfStorageUnit() : double - ValueOfClimatization() : double - HasPromotions() : bool - GetValuePromotions() : double + AddDateRange(dateRange : DateRange) : void + IsInDateRange(date : DateTime) : void

Enum AreaType

<<<enumeration>>> AreaType	
- A : int	
- B : int	
- C : int	
- D : int	
- E : int	

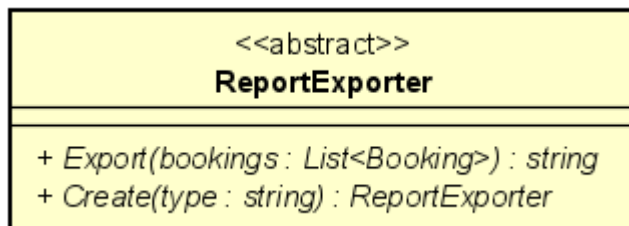
Enum SizeType

<<<enumeration>>> SizeType	
- Small : int	
- Medium : int	
- Large : int	

Clase Booking

Booking	
- Id : int	
- _approved : bool	
- _dateStart : DateTime	
- _dateEnd : DateTime	
- _storageUnit : StorageUnit	
- _rejectedMessage : string	
- _status : string	
- _payment : bool	
- _personEmail : string	
+ Booking() : void + Booking(approved : bool, dateStart : DateTime, dateEnd : DateTime, StorageUnit : StorageUnit, string rejectedBooking : bool, status : string, payment : bool, personEmail : string) : void + <<property>> Approved() : bool + <<property>> DateStart() : DateTime + <<property>> DateEnd() : DateTime + <<property>> StorageUnit() : StorageUnit + <<property>> RejectedMessage() : string + <<property>> Status() : string + <<property>> Payment() : bool + <<property>> personEmail() : string + GetCountOfDays() : int + CalculateBookingTotalPrice() : double - TotalPriceWithDiscountForBookingDays() : double - CheckDiscount(totalPrice : double) : double - RuleOf3(totalPrice : double, discount : double) : double + CheckRejection() : bool + CheckDate() : bool - IfHasInvalidDateThrowException() : void - IfHasInvalidRejectionThrowException() : void	

Clase ReportExporter



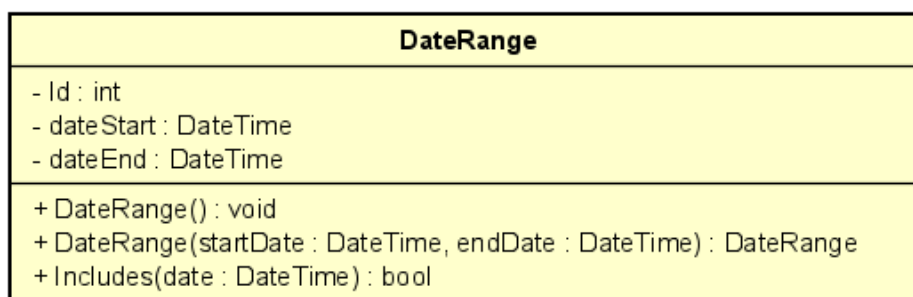
Clase CsvReportExporter



Clase TxtReportExporter



Clase DateRange



Clase PersonDto

PersonDto
+ Name : string + Surname : string + Email : string + Password : string + IsAdmin : bool
+ PersonDto() : PersonDto + PersonDto(name : string, surname : string, email : string, password : string, isAdmin : bool) : PersonDto

Clase PromotionDto

PromotionDto
+ Label : string + Discount : int + dateStart : datetime + dateEnd : datetime
+ PromotionDto() : PromotionDto + PromotionDto(label : string, discount : int, dateStart : datetime, dateEnd : datetime) : PromotionDto

Clase StorageUnitDto

StorageUnitDto
+ Id : string + Area : AreaType + Size : SizeType + Climatization : bool + List<PromotionDto> Promotions : PromotionDto + List<DateRangeDto> AvailableDates : DateRangeDto
+ StorageUnitDto() : StorageUnitDto + StorageUnitDto(id : string, area : AreaType, size : SizeType, climatization : bool, promotions : List<PromotionDto>, availableDates : List<DateRangeDto>) : StorageUnitDto

Clase BookingDto

BookingDto
+ Approved : bool + DateStart : datetime + DateEnd : datetime + StorageUnitDto : StorageUnitDto + RejectedMessage : string + personEmail : string
+ BookingDto() : BookingDto + BookingDto(approved : bool, dateStart : datetime, dateEnd : datetime, storageUnitDto : StorageUnitDto, rejectedMessage : string, personEmail : string) : BookingDto

Clase AreaTypeDto

AreaTypeDto
- Value : int - Name : string
+ AreaTypeDto(areaType : AreaType) : AreaTypeDto

Clase SizeTypeDto

SizeTypeDto
- Value : int - Name : string
+ SizeTypeDto(sizeType : SizeType) : SizeTypeDto

Clase DateRangeDto

DateRangeDto
+ DateStart : DateTime + DateEnd : DateTime
+ DateRangeDto() : void + DateRangeDto(startDate : DateTime, endDate : DateTime) : DateRangeDto

PersonController

PersonController
- _personRepositories : PersonRepository
+ PersonController(context : ApplicationDbContext) : PersonController + CheckIfEmailsRegistered(email : string) : bool - IfEmailsNotRegisteredThrowException(registered : bool) : void + CheckIfPasswordIsCorrect(password : string, verifyPassword : string) : bool - PasswordStringMatch(password : string, verifyPassword : string) : bool + Login(email : string, password : string) : PersonDto - LoginCheckPersonValidations(email : string, password : string) : PersonDto + SignUp(personDto : PersonDto) : void - AddPersonIfItsValid(personDto : PersonDto) : void + GetPersonDtoFromEmail(personEmail : PersonDto) : PersonDto + CheckIfAdminExists() : bool

PromotionController

PromotionController
- _promotionRepositories : PromotionRepository + PromotionController(context : ApplicationDbContext) : PromotionController + CreatePromotion(promotion : PromotionDto) : void + ModifyPromotion(oldLabel : string, promotion : PromotionDto) : void + RemovePromotion(promotion : PromotionDto) : void + GetPromotionsDto() : List<PromotionDto> + GetPromotionDtoFromLabel(label : string) : PromotionDto - IfPromotionDoesNotExistThrowException() : void - IfPromotionExistsThrowException() : void

ReportExportController

ReportExportController
- _bookingsRepositories : BookingsRepository - _reportExporter : ReportExporter + ReportExportController(context : ApplicationDbContext) : ReportExportController + Export(format : string) : string

StorageUnitController

StorageUnitController
- _storageUnitRepositories : StorageUnitRepository - _promotionsRepository : PromotionsRepository + StorageUnitController(context : ApplicationDbContext) : StorageUnitController + CreateStorageUnit(storageUnitDto : StorageUnitDto) : void + RemoveStorageUnit(storageUnitDto : StorageUnitDto) : void + CreateListPromotions(storageUnitDto : StorageUnitDto) : List<Promotion> + CreateListAvailableDates(storageUnitDto : StorageUnitDto) : List<DateRange> + GetStorageUnitsDto() : List<StorageUnitDto> + ChangeToPromotionsDto(promotions : List<Promotion>) : List<PromotionDto> + ChangeToDateRangeDto(availableDates : List<DateRange>) : List<DateRangeDto> + GetStorageUnitDtoFromId(id : string) : StorageUnitDto + DeletePromotionFromAllStorageUnits(promotionDto : PromotionDto) : void + AddAvailableDateRangeToStorageUnit(id : string, dateRangeDto : DateRangeDto) : void + SearchAvailableStorageUnits(dateRangeDto : DateRangeDto) : List<StorageUnitDto> + EliminateDateRangeFromStorageUnit(id : string, dateRangeDto : DateRangeDto) : void + ModifyOrRemoveDateRange(storageUnitDto : StorageUnitDto, dateRangeDto : DateRangeDto) : void + CheckIfDateStartAndDateEndAreIncludeInDateRange(startDate : DateTime, endDate : DateTime, dateRangeDto : DateRangeDto) : void + CalculateStorageUnitPricePerDay(storageUnitDto : StorageUnitDto, dateRangeDto : DateRangeDto) : double + ConvertAreaTypeToAreaTypeDto(areaType : AreaType) : AreaTypeDto + ConvertAreaTypeDtoToAreaType(areaTypeDto : AreaTypeDto) : AreaType + ConvertSizeTypeToSizeTypeDto(sizeType : SizeType) : SizeTypeDto + ConvertSizeTypeDtoToSizeType(sizeTypeDto : SizeTypeDto) : SizeType - IfDateRangeAlreadyExistsThrowException(storageUnit : StorageUnit, newDateRange : DateRangeDto) : void - IfStorageUnitAlreadyExistsThrowException() : void - IfStorageUnitDoesNotExistThrowException() : void - DateRangeExistsSoThrowException() : void - IfDateRangeIsInvalidThrowException(dateRangeDto : DateRangeDto) : void - IfThereIsNoStorageUnitWithThisDateRangeThrowException(availableStorageUnits : List<StorageUnitDto>) : void

BookingController

BookingController
<ul style="list-style-type: none"> - _bookingRepositories : BookingsRepository - _storageUnitsRepository : StorageUnitsRepository
<ul style="list-style-type: none"> + BookingController(context : ApplicationDbContext) : int + CreateBooking(userEmail : string, bookingDto : BookingDto) : void - CheckIfAlreadyBookedAndAddBooking(userEmail : string, bookingDto : BookingDto) : void - IfUserAlreadyBookTheStorageUnitThrowException() : void + CheckIfBookingsApproved(bookingDto : BookingDto) : bool + ChangeToStorageUnit(storageUnitDto : StorageUnitDto) : StorageUnit + CalculateTotalPriceOfBooking(bookingDto : bookingDto) : double + PayBooking(userEmail : string, bookingDto : BookingDto) : void + GetAllBookingsDto() : List<BookingDto> - IfBookingPaymentIsAlreadyTrueThrowException(booking : Booking) : void + ApproveBooking(userEmail : string, bookingDto : BookingDto) : void - IfUserDidNotMakeThePaymentThrowException() : void - IfBookingsAlreadyApprovedThrowException() : void - IfBookingRejectedMessageIsNotEmptyThrowException() : void + SetRejectionMessage(userEmail : string, bookingDto : BookingDto, rejectionMessage : string) : void - IfRejectionMessageIsEmptyThrowException() : void

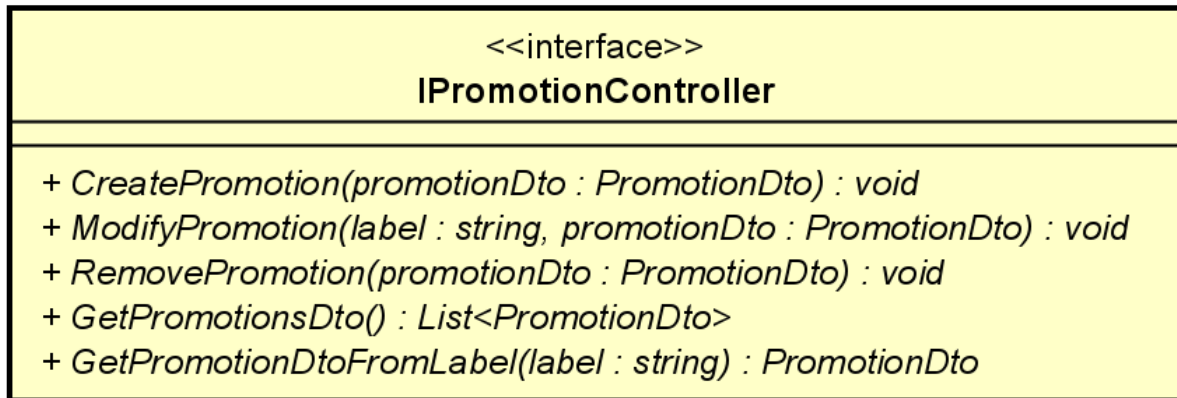
IReportExportController

<<interface>>
IReportExportController
+ <i>Export(format : string) : string</i>

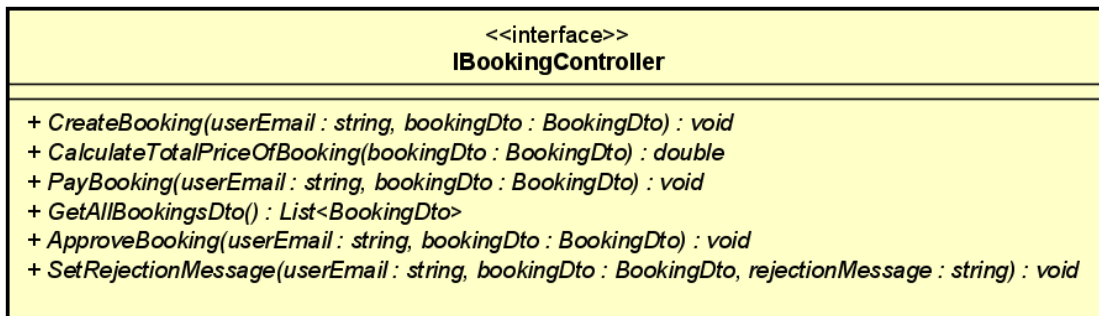
IPersonController

<<interface>>
IPersonController
<ul style="list-style-type: none"> + <i>CheckIfPasswordIsCorrect(password : string, verifyPassword : string) : bool</i> + <i>Login(email : string, password : string) : PersonDto</i> + <i>SignUp(personDto : PersonDto) : void</i> + <i>GetPersonDtoFromEmail(personEmail : string) : PersonDto</i> + <i>CheckIfAdminExists() : bool</i>

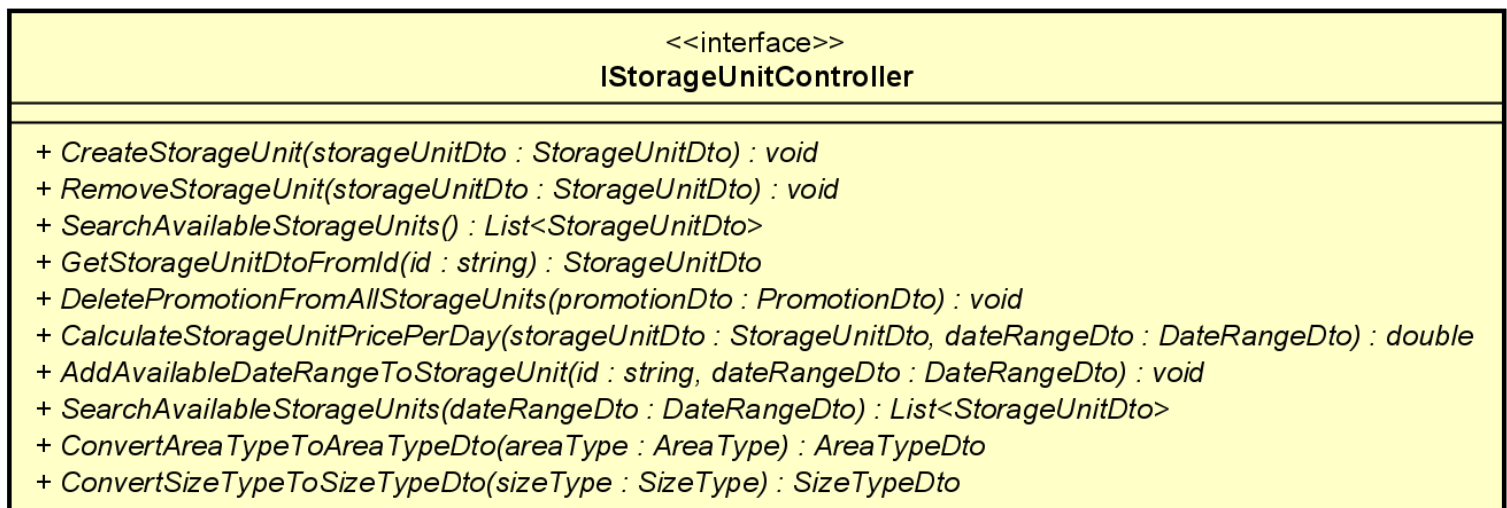
IPromotionController



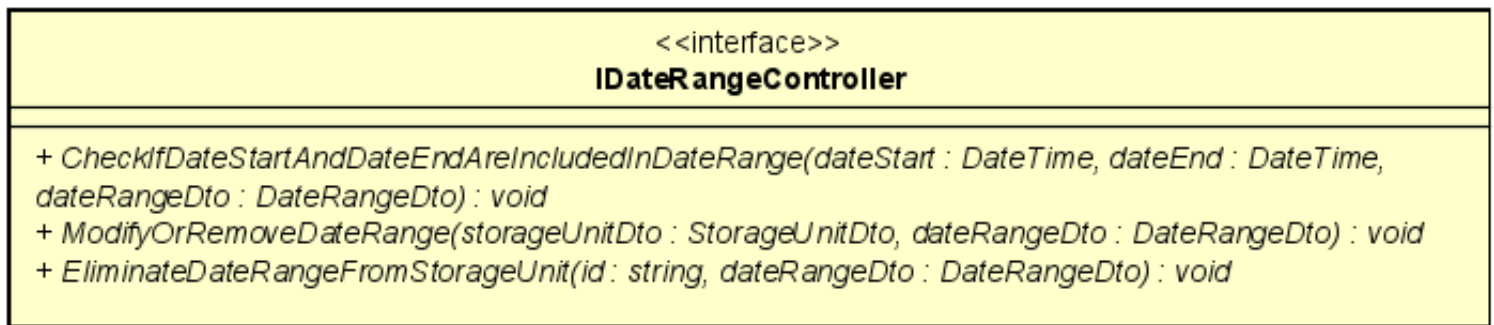
IBookingController



IStorageUnitController



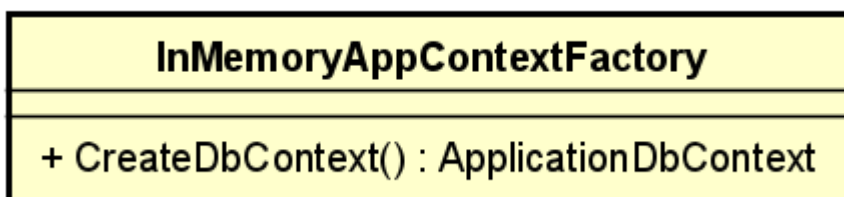
IDateRangeController



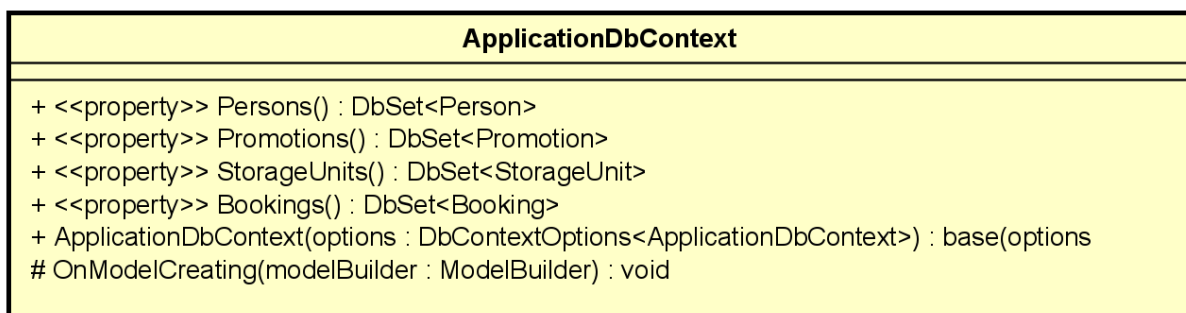
IApplicationDbContextFactory



InMemoryAppContextFactory



ApplicationDbContext



PersonsRepository

PersonsRepository
- _database : ApplicationDbContext
+ PersonsRepository(database : ApplicationDbContext) : PersonsRepository + AddPerson(person : Person) : void - AddNewPersonToPersonsTable(person : Person) : void - PersonAlreadyExistsSoThrowException() : void + PersonAlreadyExists(newPerson : Person) : bool + FindPersonByEmail(email : string) : Person + GetAllPersons() : List<Person>

BookingsRepository

BookingsRepository
- _database : ApplicationDbContext
+ BookingsRepository(database : ApplicationDbContext) : BookingsRepository + AddBooking(booking : Booking) : void - AddNewBookingToBookingsTable(booking : Booking) : void - BookingAlreadyExistsSoThrowException() : void + BookingAlreadyExists(booking : Booking) : bool + FindBookingByStorageUnitIdAndEmail(id : string, email : string) : Booking + GetAllBookings() : List<Booking> + DeleteBooking(booking : Booking) : void + UpdateBooking(booking : Booking) : void

PromotionsRepository

PromotionsRepository
- _database : ApplicationDbContext
+ PromotionsRepository(database : ApplicationDbContext) : PromotionsRepository + AddPromotion(promotion : Promotion) : void - AddNewPromotionToPromotionsTable(promotion : Promotion) : void - PromotionAlreadyExistsSoThrowException() : void + PromotionAlreadyExists(promotion : Promotion) : bool + DeletePromotion(promotion : Promotion) : void + UpdatePromotion(label : string, newPromotion : Promotion) : void + FindPromotionByLabel(label : string) : Promotion + GetAllPromotions() : List<Promotion>

StorageUnitsRepository

StorageUnitsRepository

- _database : ApplicationDbContext

- + StorageUnitsRepository(database : ApplicationDbContext) : StorageUnitsRepository
- + AddStorageUnit(storageUnit : StorageUnit) : void
- AddNewStorageUnitToStorageUnitsTable(storageUnit : StorageUnit) : void
- StorageUnitAlreadyExistsSoThrowException() : void
- + StorageUnitAlreadyExists(storageUnit : StorageUnit) : bool
- + DeleteStorageUnit(storageUnit : StorageUnit) : void
- + GetStorageUnitFromId(id : string) : StorageUnit
- + GetAllStorageUnits() : List<StorageUnit>
- + AddAvailableDateToStorageUnit(storageUnitId : string, dateRange : DateRange) : void
- + DeleteAvailableDateFromStorageUnit(storageUnitId : string, dateRange : DateRange) : void

Modelo de tablas

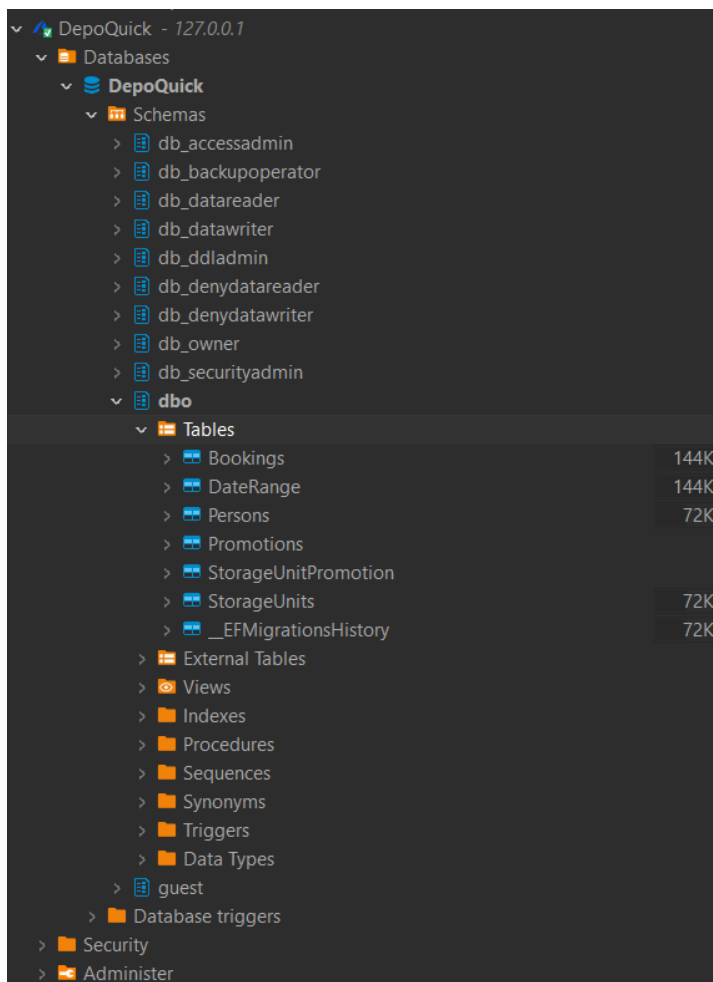


Tabla Person

Structure of the 'Persons' table:

Column	Data Type	Nullable
Email	varchar	Yes
Name	varchar	Yes
Surname	varchar	Yes
IsAdmin	bit	Yes
Password	varchar	Yes

Tabla Bookings

Structure of the 'Bookings' table:

Column	Data Type	Nullable
Id	int	No
Approved	bit	Yes
PersonEmail	varchar	Yes
DateStart	datetime	Yes
DateEnd	datetime	Yes
StorageUnitId	int	Yes
RejectedMessage	varchar	Yes
Status	int	Yes
Payment	decimal	Yes

Tabla Promotion

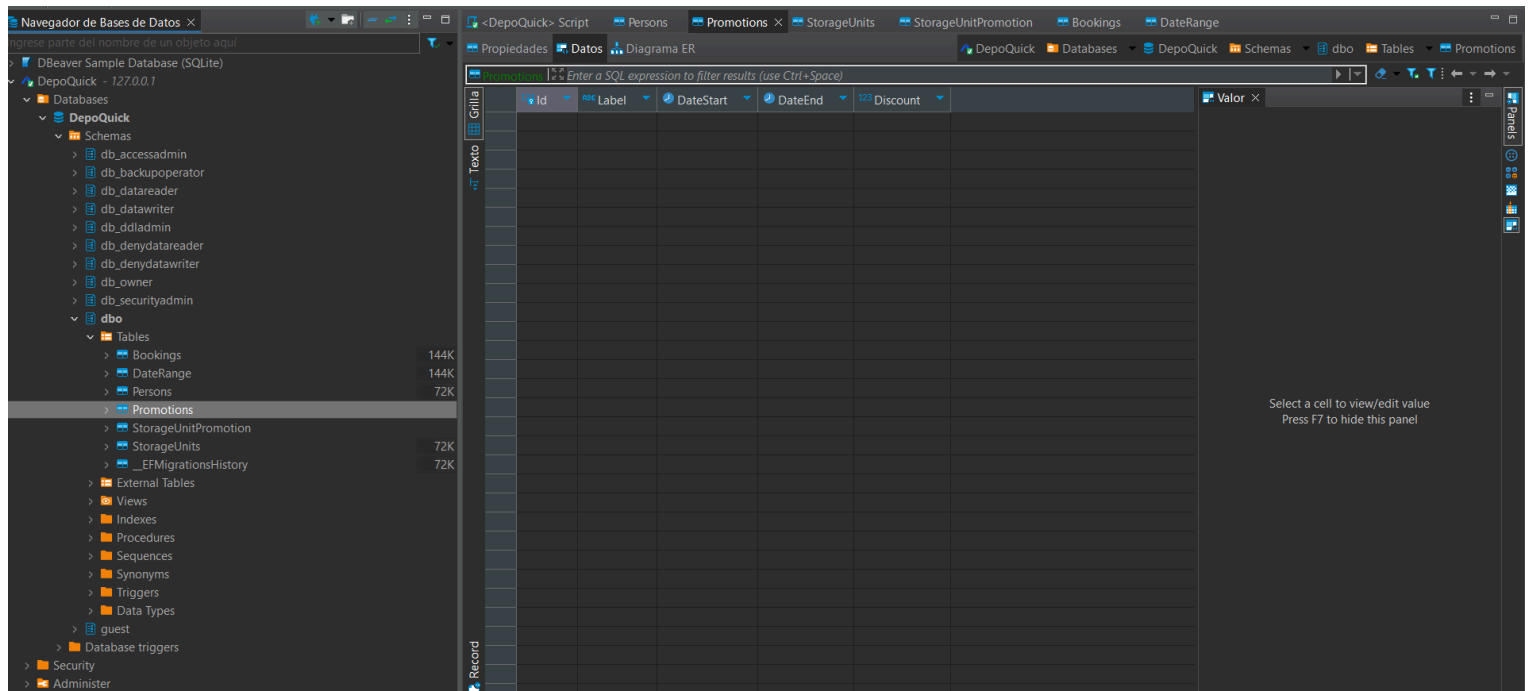


Tabla DateRange

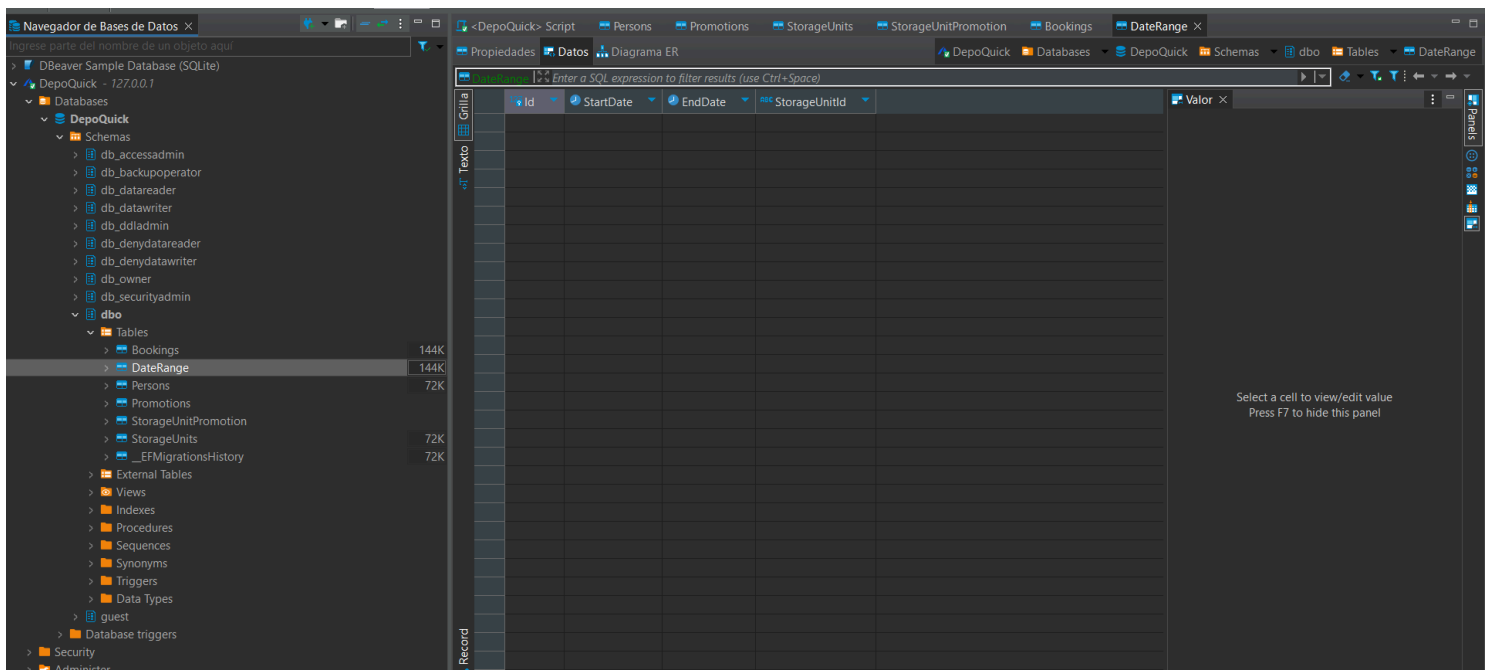
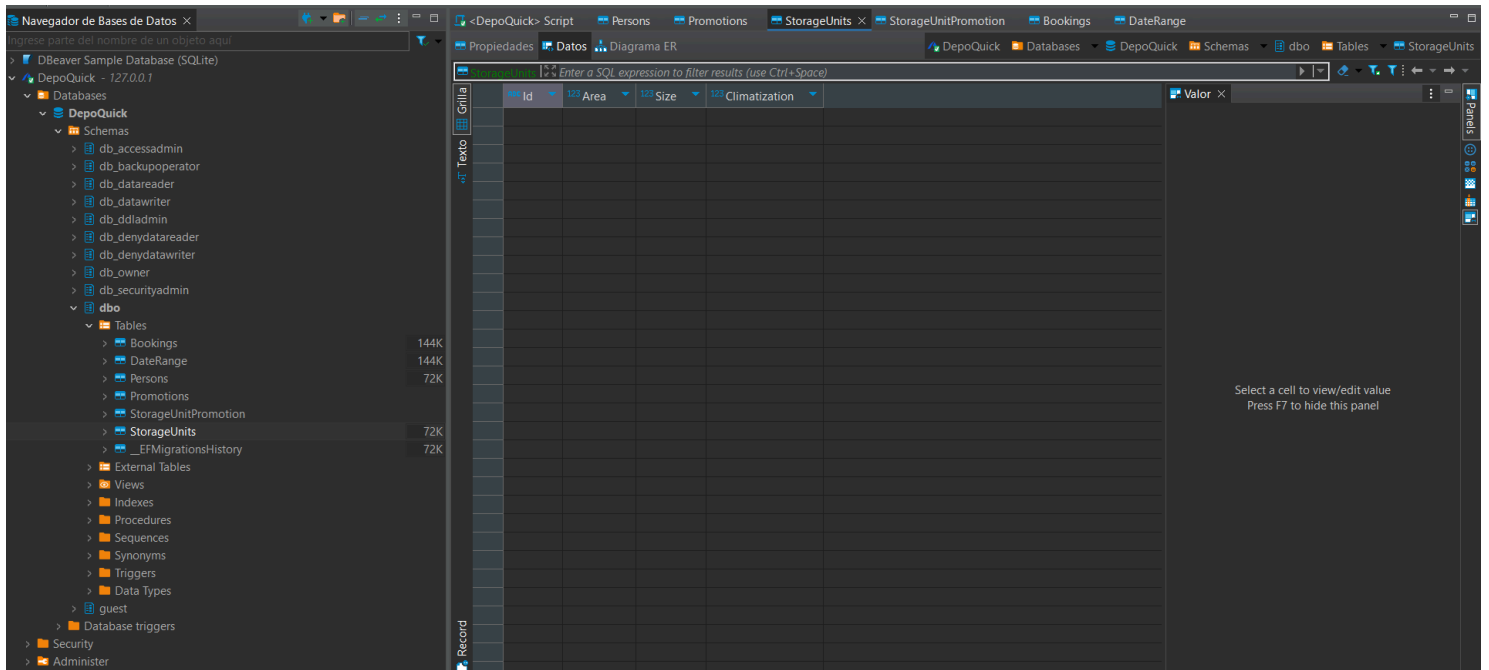


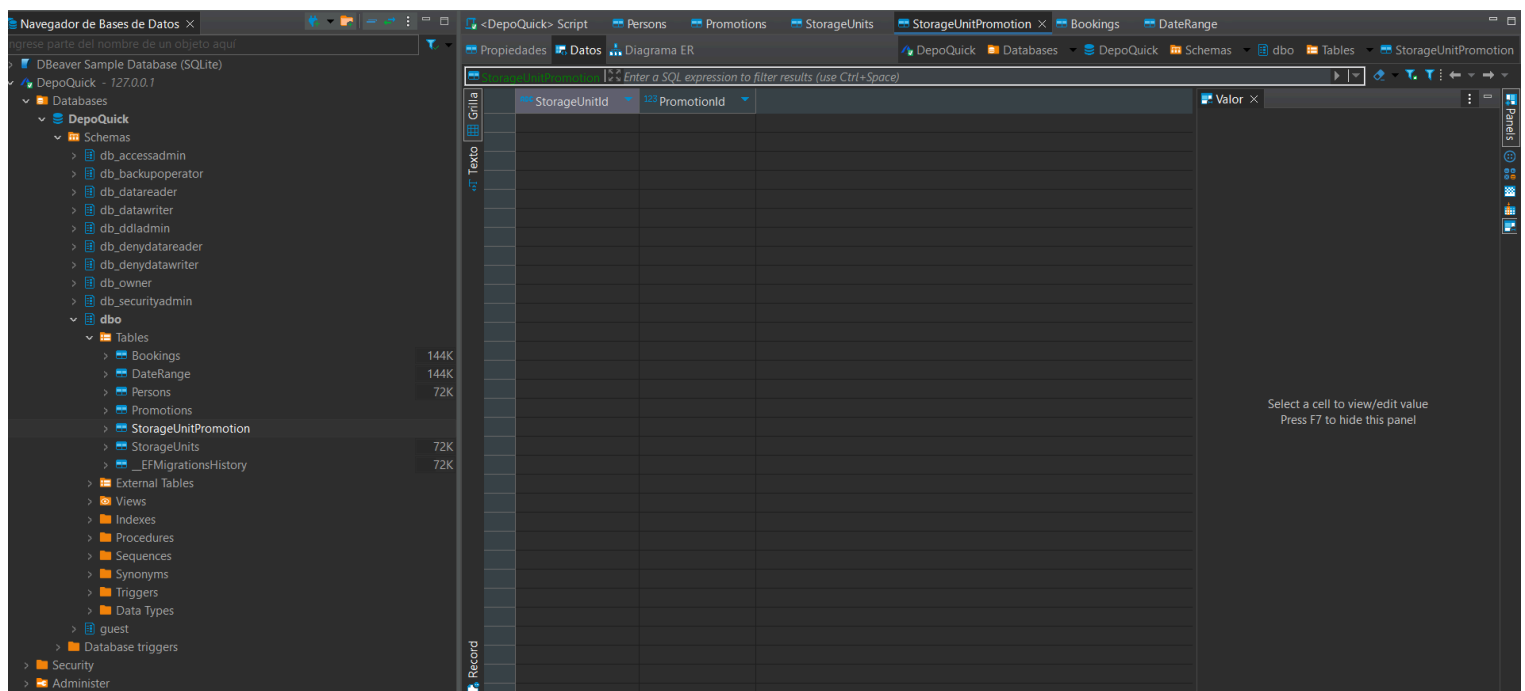
Tabla StorageUnit



The screenshot shows the DBeaver interface with the 'StorageUnit' table selected in the 'dbo' schema. The table has four columns: Id, Area, Size, and Climatization. The data grid is empty, and the right panel shows the 'Valor' column.

Id	Area	Size	Climatization
----	------	------	---------------

Tabla StorageUnitPromotion



The screenshot shows the DBeaver interface with the 'StorageUnitPromotion' table selected in the 'dbo' schema. The table has two columns: StorageUnitId and PromotionId. The data grid is empty, and the right panel shows the 'Valor' column.

StorageUnitId	PromotionId
---------------	-------------

Análisis de Cobertura de Pruebas Unitarias (Rider)

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
✓ Total	96%	79/2195
✓ Model	98%	8/472
✓ Model	98%	8/472
> CsvReportExporter	100%	0/20
> Person	100%	0/127
> ReportExporter	100%	0/6
> StorageUnit	100%	0/100
> TxtReportExporter	100%	0/24
> Exceptions	100%	0/18
> Booking	98%	2/94
> Promotion	94%	4/66
> DateRange	88%	2/17
> ModelTests	97%	10/287

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
✓ Controllers	97%	20/581
✓ Controllers	97%	20/581
> ReportExportController	100%	0/9
> Dtos	100%	0/123
> StorageUnitController	99%	3/238
> PersonController	98%	1/56
> PromotionController	96%	2/46
> BookingController	87%	14/109
✓ DataAccess	96%	7/188
✓ DataAccess	96%	7/188
> Repository	97%	4/156
> Context	91%	3/32
> ControllersTests	94%	28/458

Symbol	Coverage (%) ▾	Uncovered/Total Stmts.
✓ ModelTests	97%	10/287
✓ ModelTests	97%	10/287
> CsvReportExporterTests	100%	0/24
> DateRangeTests	100%	0/17
> TxtReportExporterTests	100%	0/24
> StorageUnitTests	98%	1/51
> BookingTests	97%	2/62
> ReportExporterTests	97%	1/33
> PersonTests	92%	3/38
> PromotionTests	92%	3/38

ControllersTests	94%	28/458
LogicTests	94%	28/458
ReportExportControllerTests	100%	0/17
PromotionControllerTests	96%	2/47
PersonControllerTests	95%	4/81
StorageUnitControllerTests	94%	11/190
BookingControllerTests	91%	11/123

Symbol	Coverage (%)	Uncovered/Total Stmts.
ModelTests	97%	10/287
DataAccessTests	97%	6/209
DataAccessTests	97%	6/209
PromotionRepositoryTests	98%	1/49
BookingRepositoryTests	97%	2/67
PersonRepositoryTests	97%	1/35
StorageUnitsRepositoryTests	97%	2/58

Diagrama UML generado en la base de datos

