Diseño de aplicaciones 2

OBLIGATORIO 1


Facundo Pujol 226033
Serafín Revetria 209143

Facultad ORT Uruguay

GitHub: https://github.com/ORT-DA2/DA2-Pujol-Revetria.git

Descripción del diseño

Instalación

Nuestro sistema es muy fácil de utilizar, lo primero es realizar dos configuraciones:

- El archivo appsettings.json hay que cambiar en la parte "ConnectionStrings": { "BookedUYDB": "Server=.\\"NOMBRE DE TU INSTANCIA"... }, poner el nombre de tu instancia de SQL Server(sin comillas).
- El archivo Key.txt contiene nuestra secret key para generar el JWT (JSON Web Token), este ya tiene una clave pero si se desea se puede modificar.

Después de estas configuraciones hay que ejecutar WebApi.exe (preferentemente como administrador) y la API está abierta en localhost:5000.

Descripción general del trabajo:

El trabajo consiste en crear una Web API que ofrezca operaciones que simulan un sistema de reservas para hospedajes. En si el sistema está basado en "Uruguay Natural", que es una página que ofrece información sobre puntos turísticos de posible interés. La página tiene los puntos turísticos separados por región, y cada uno de ellos está asociado a una categoría en concreto. El cliente podrá realizar consultas a través de llamadas http.

Pero el obligatorio va más allá de la página, ya que se agregan funcionalidades como realizar reservas dentro de un acomodamiento que se encuentra dentro de un punto turístico elegido, hacer un mantenimiento del estado de la reserva, crear administradores que serán los encargados de mantener el estado de una reserva o los administradores pueden marcar un acomodamiento como lleno. Esta funcionalidad genera el nuevo concepto de acomodamiento, reserva, huésped, entre otros. Toda esta información se deberá almacenar dentro de una base de datos creada por nosotros.

Nuestra solución se encarga de resolver todos los puntos detallados anteriormente de manera satisfactoria mediante la creación de una Web API y el código que la respalda.

En resumen creamos un sistema capaz de consultar por puntos turísticos en una región, que cumplan ciertas categorías. Luego puedo consultar por acomodamientos dentro de un punto turístico y realizar reservas en ellos y luego consultar el estado de mi reserva. Desde el punto de vista del administrador, este puede agregar nuevos estados de reserva, agregar más administradores, crear acomodamientos, actualizar la capacidad de un acomodamiento, borrar acomodamientos y crear puntos turísticos.

Bugs reconocidos: A la hora de crear una booking se devuelve el objeto pero se puede apreciar que AccommodationName, AccommodationAdress y AccommodationContact son devueltos en null, luego de testearlo varias veces en Postman y a pesar de haber realizado una minuciosa lectura nuestro código no logramos encontrar el error y debido a que se descubrió sobre el final del tiempo de entrega, no se logró solucionar. De todas formas, si se crea la booking con los datos correspondientes de Accommodation y se almacena en la base de datos de manera correspondiente, por lo que consideramos que es un error menor que no afecta gravemente. En el video grabado para postman se puede apreciar perfectamente que en el minuto 5:30 se hace el add y se ve el

bug correspondiente, pero en el 6:23 cuando se pide la booking por Id, se puede observar que esos valores no vienen en null, confirmando que se almacenan de manera correspondiente.

Diagrama de descomposición

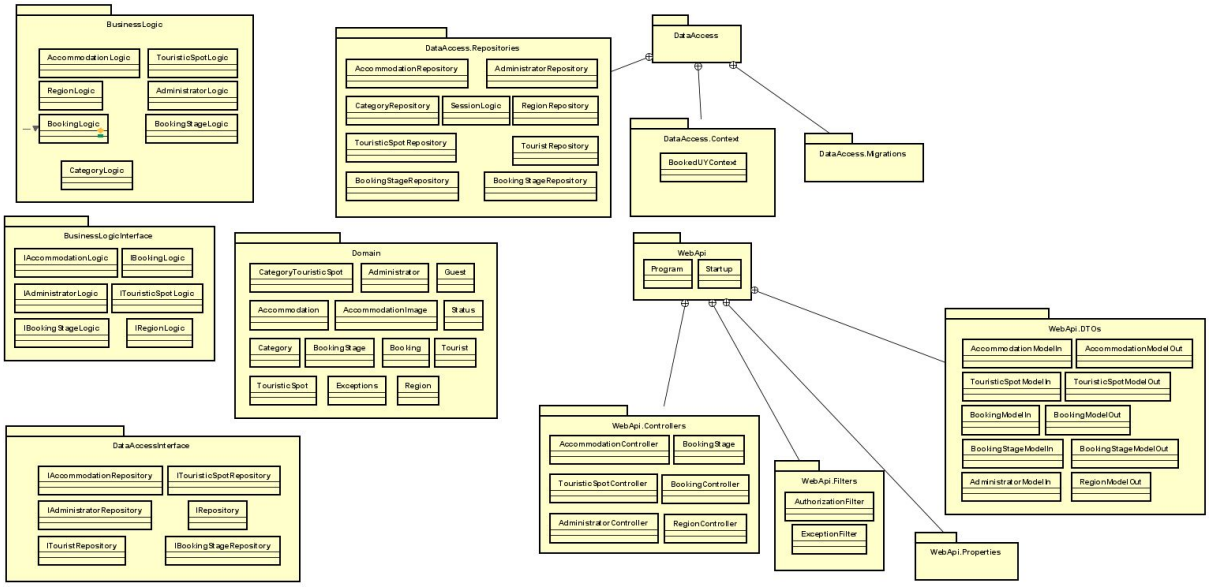
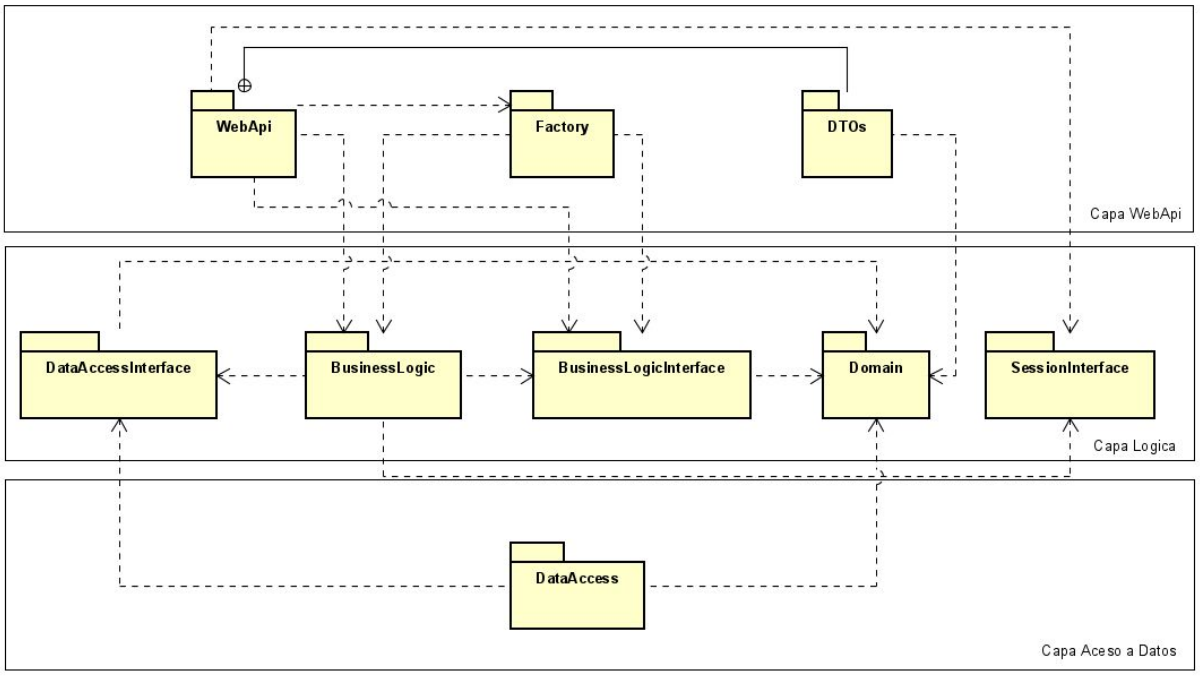
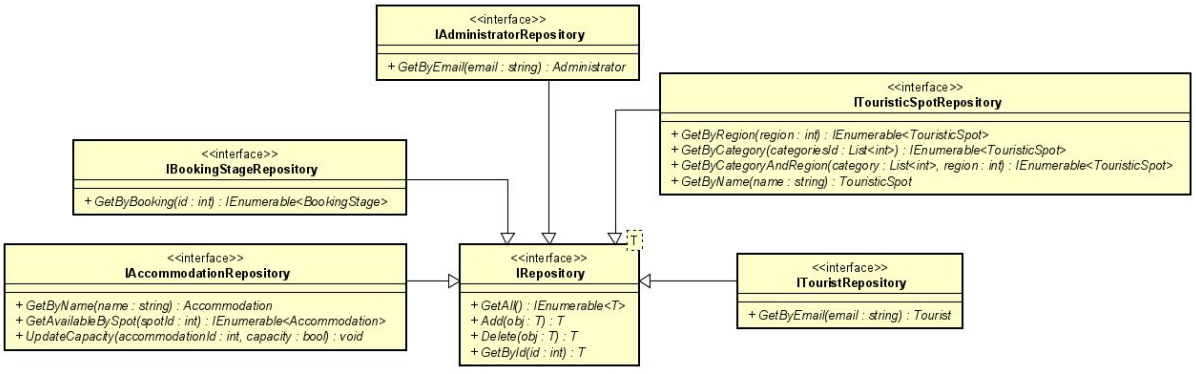


Diagrama de paquetes



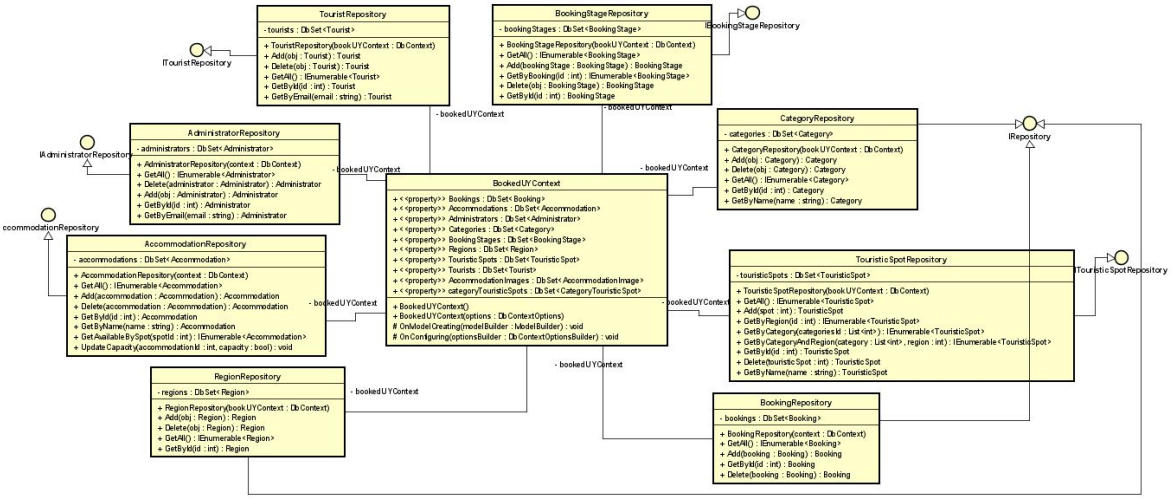
DataAccesInterface

Paquete encargado de definir las interfaces con sus métodos encargados del acceso a los datos dentro de la base de datos.



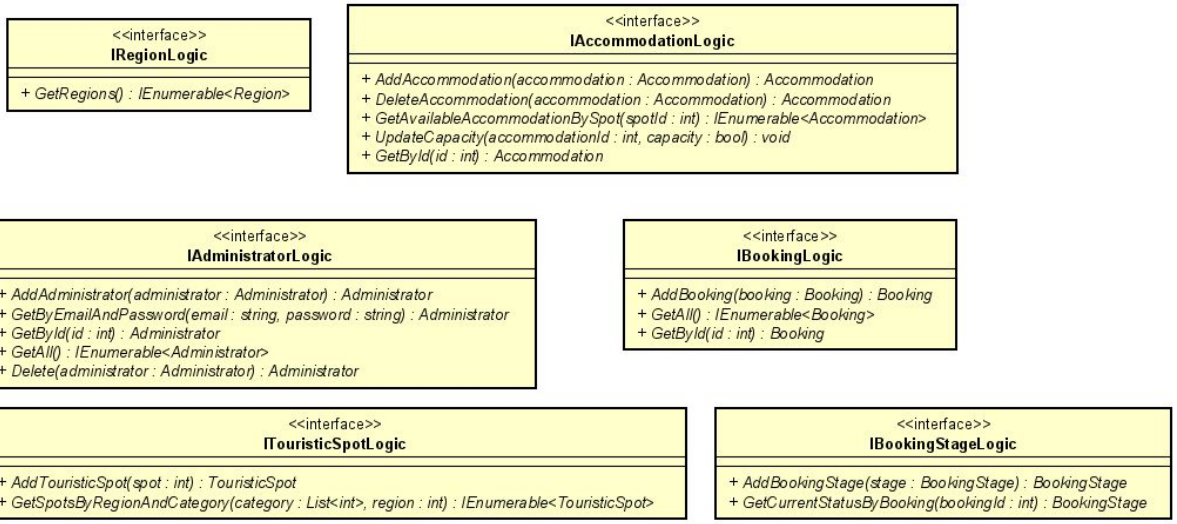
DataAccess

Implementación de las interfaces en DataAccesInterface, contiene el modelo de acceso a datos de cada entidad del sistema.



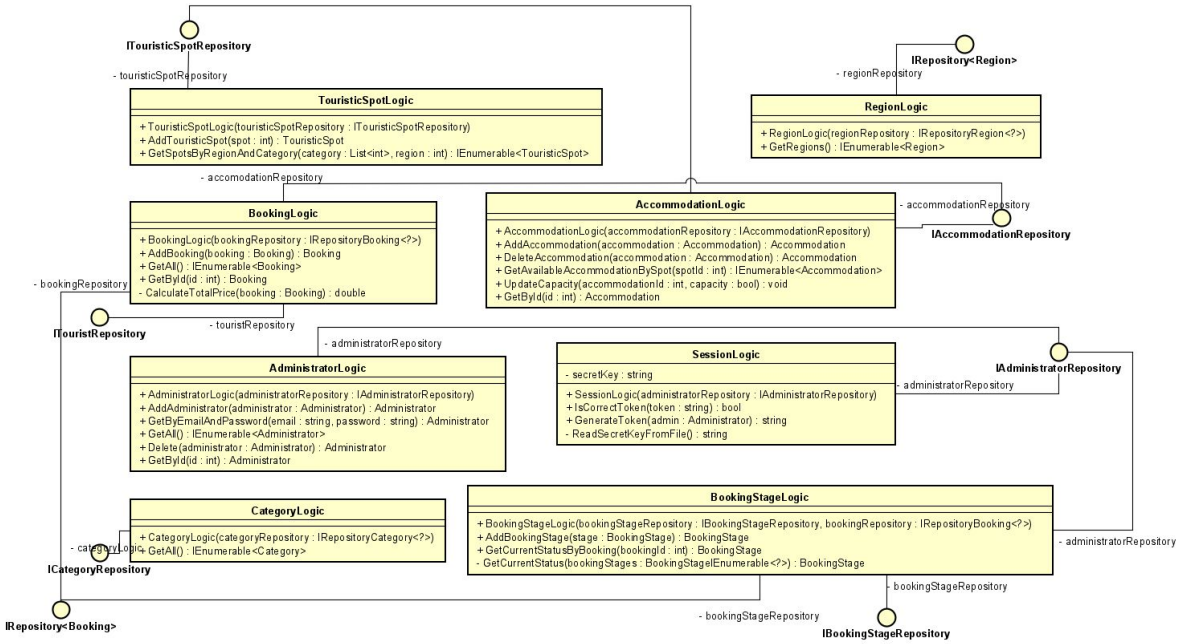
BusinessLogicInterface

Interfaces con sus respectivos métodos que se ocupan de la lógica de negocio, es decir, métodos encargados de procesar las consultas de la WebApi y de consultar los repositorios.



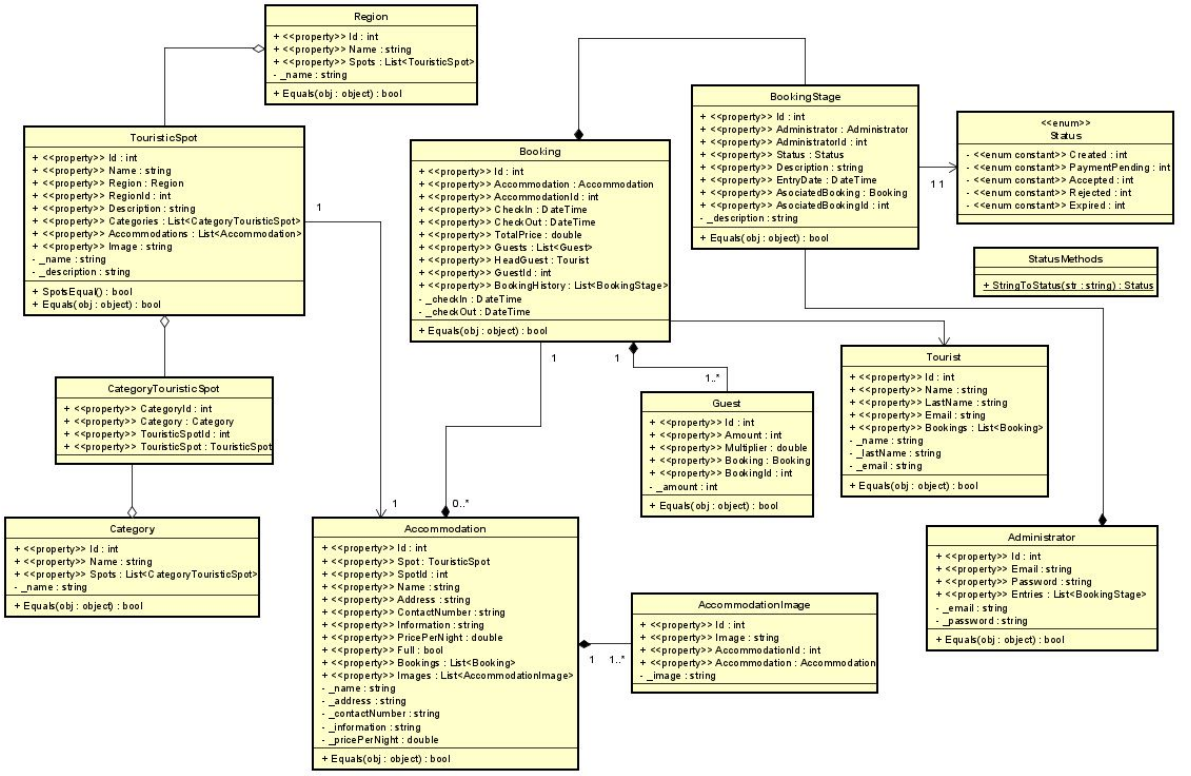
BusinessLogic

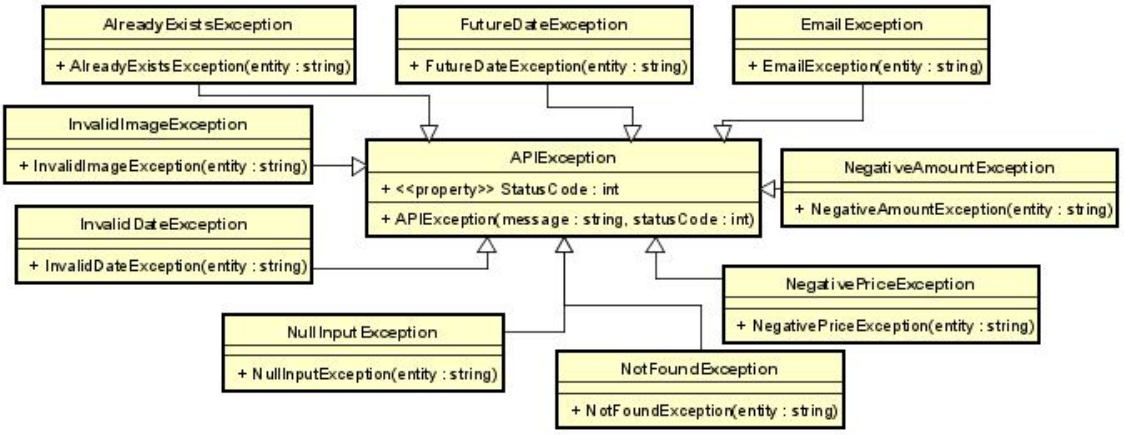
Implementación de las interfaces de BusinessLogicInterface con la implementación de métodos necesaria para nuestro sistema BookedUY.



Domain

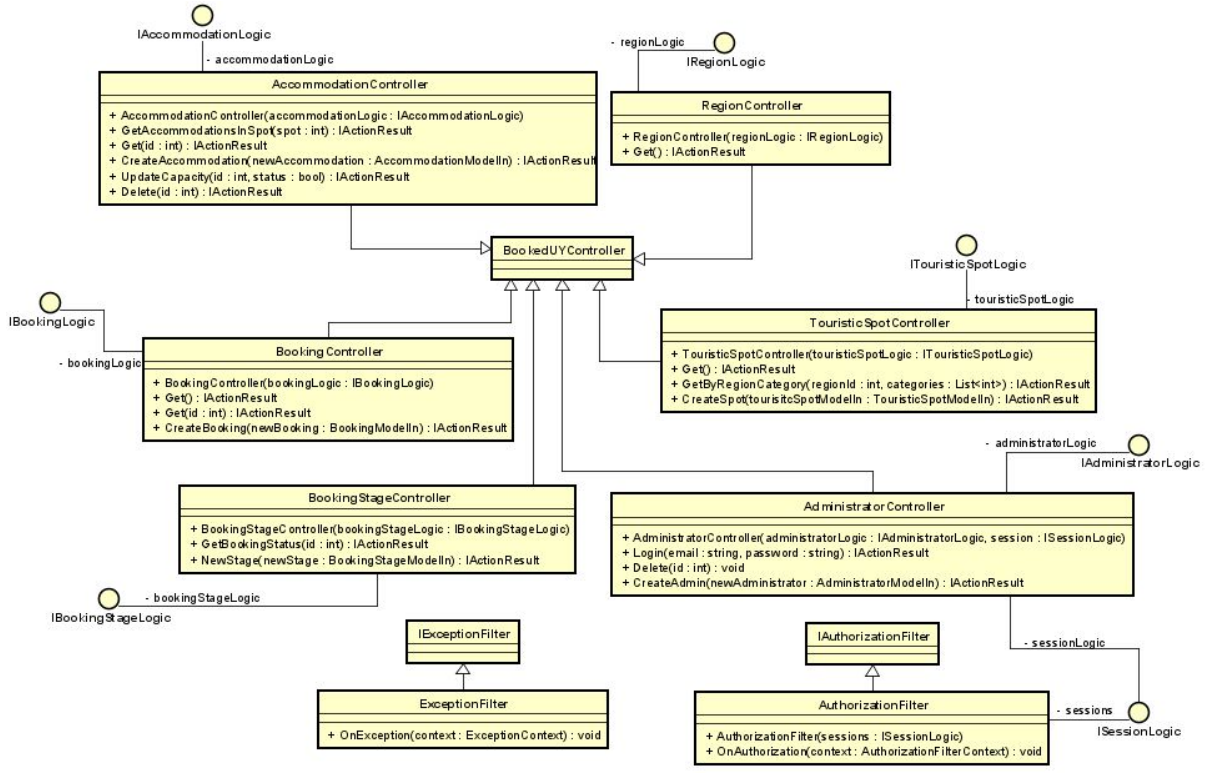
Paquete que contiene todas las entidades de negocio de nuestro sistema con sus respectivas relaciones entre ellas.





WebApi

Paquete encargado de resolver las diferentes consultas HTTP hechas al sistema. Se encarga de recibir las llamadas pasarle a la lógica de negocio las entidades de entrada o pedirle que este le devuelva una respuesta para una llamada.



Factory

Este paquete se encarga de la inyección de dependencias del sistema y fue creadp para reducir el acoplamiento delegando operaciones a esta clase.

Migrations

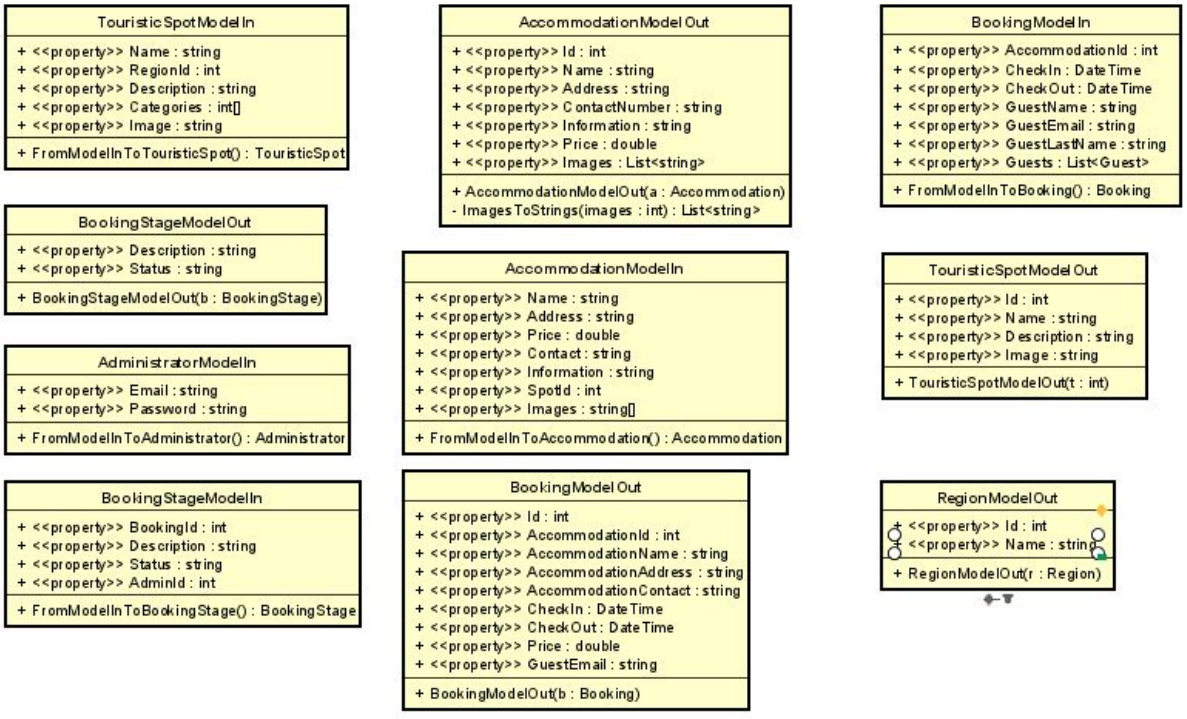
Este paquete está creado para la generación de migraciones de la base de Datos. Para definir correctamente quien es responsable de este proceso.

SessionInterface

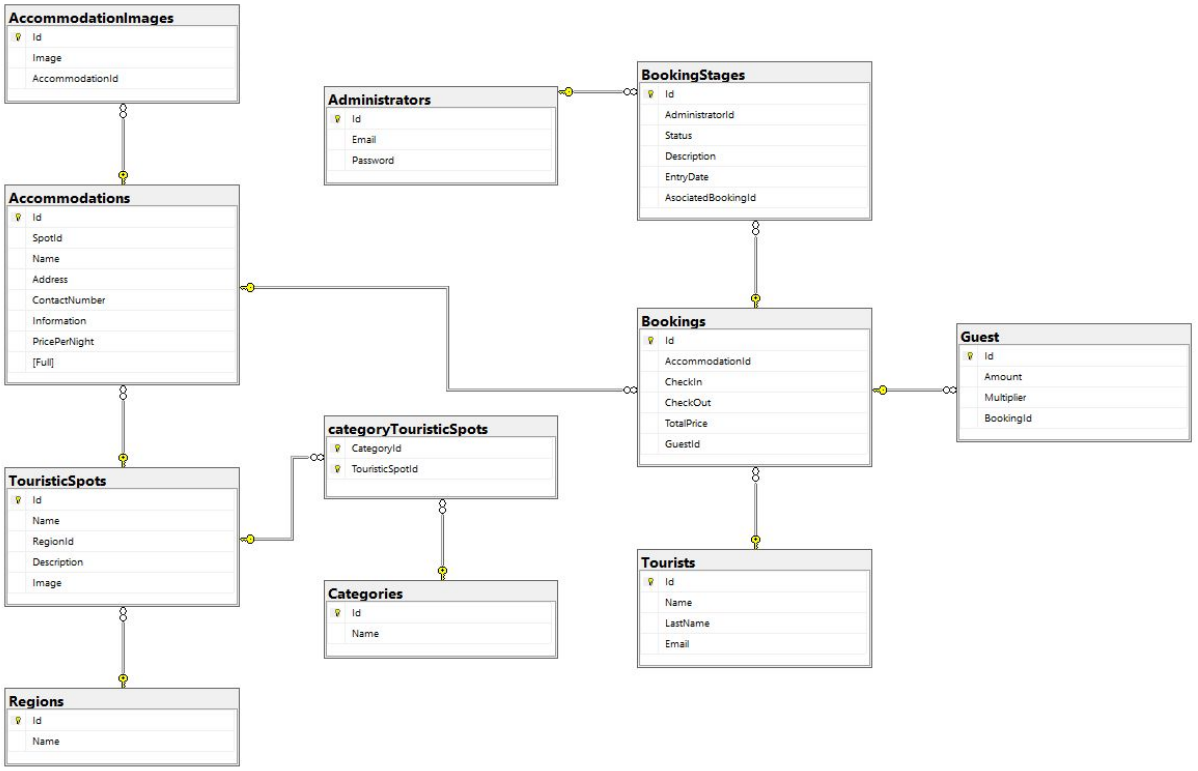
El objetivo de este paquete es definir las responsabilidades de los métodos usados por el AuthorizationFilter para otorgar la autorización y de entregar el token en el login.

WebApi.DTO

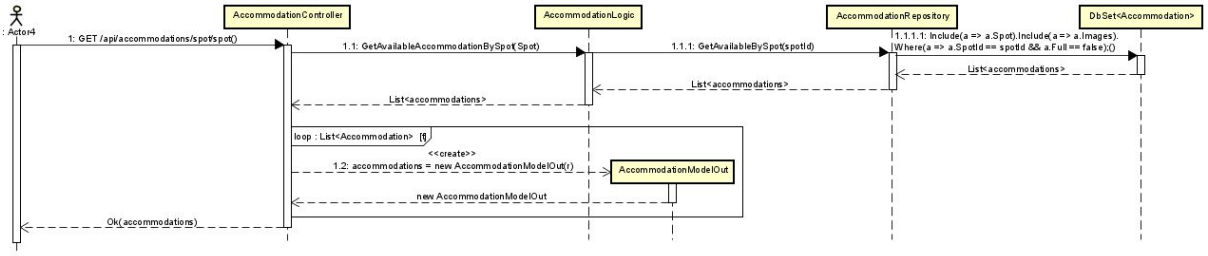
En este paquete tenemos los Data Transfer Objects, estos son modelos de entrada y salida. Estos objetos sirven para que a la salida nuestras entidades no muestren información innecesaria o sensible y para el caso de los modelos de entrada se sirve porque el usuario no tiene toda la información para crear una entidad de negocio entonces le pedimos solo lo que sabemos que puede llegar a tener.



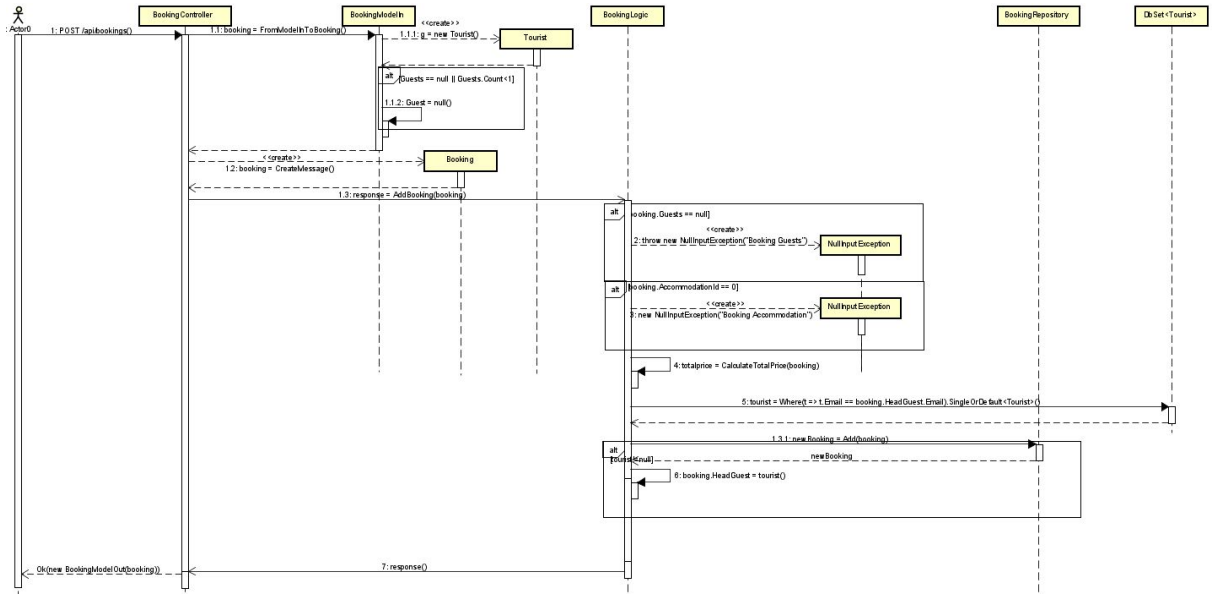
Modelo de Tablas de la Base de Datos



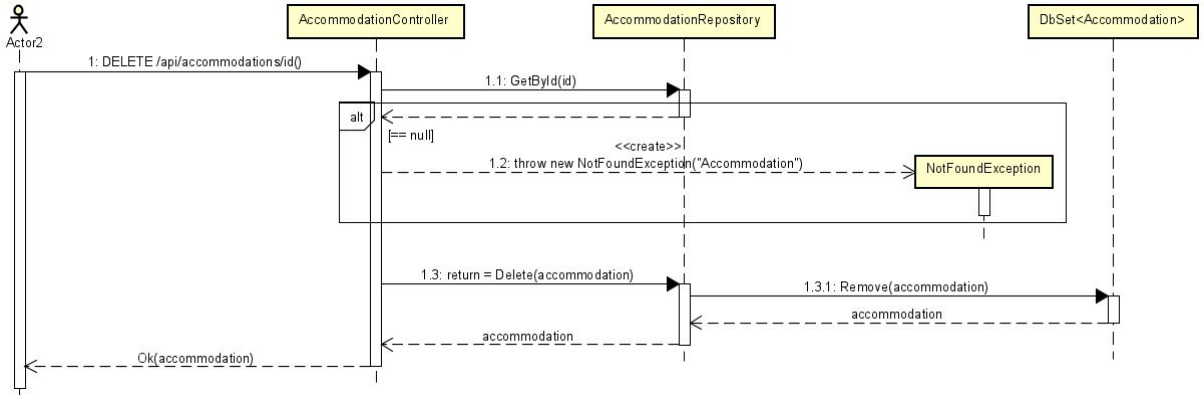
Buscar hospedajes para un cierto punto turístico con los parámetros especificados



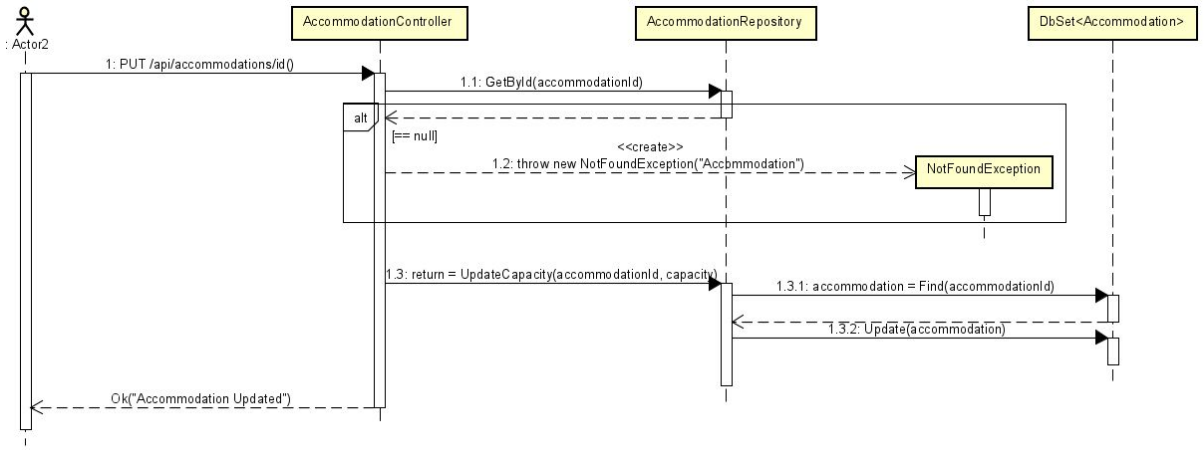
Realizar una reserva de un hospedaje



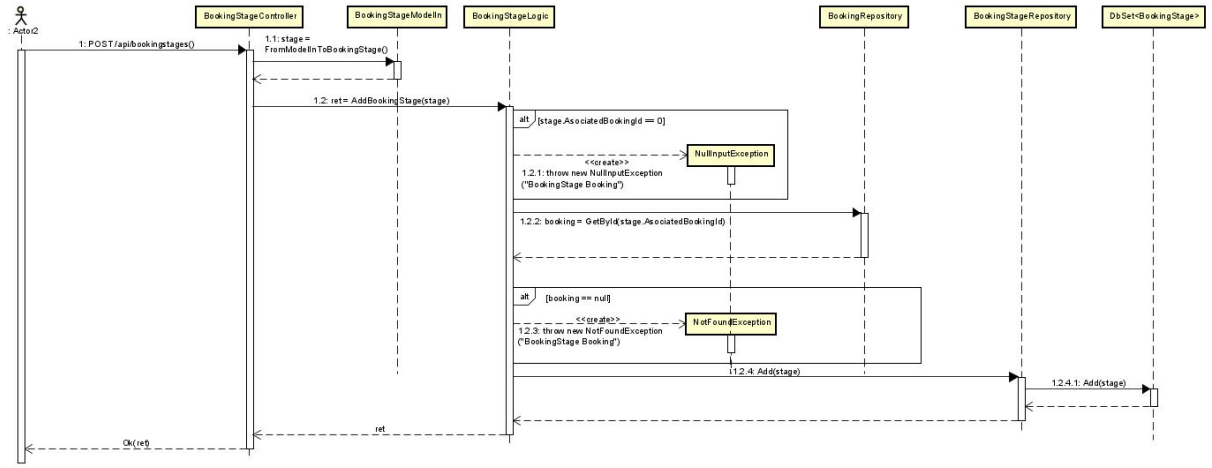
Dar de alta un nuevo hospedaje o borrar uno existente, para un punto turístico existente



Modificar la capacidad actual de un hospedaje



Cambiar el estado de una reserva , indicando una descripción



Justificaciones de Diseño

En términos de la Base de Datos se realizaron varias migraciones. Se mantuvieron las migraciones que tienen valor para el proyecto, aquellas que se crearon y tenían errores fueron eliminadas o tenían conflicto al momento de ser creadas. Nos pareció importante que quede documentado los cambios progresivos de la base de Datos.

Mediante el patrón de inyección de dependencias se logró reducir el acoplamiento del sistema. Se logró esto mediante la utilización de implementaciones entre la relación de dos clases concretas. Además de esta forma ayudamos a que no se viole los principios de Open/Close y Single Responsibility.

Para que el sistema tenga una mantenibilidad lo más alta posible decidimos aplicar varias interfaces. Por ejemplo, todas las clases de DataAccess y de BusinessLogic son implementaciones de interfaces. Esto aumenta la mantenibilidad del sistema ya que si se deseara cambiar las implementaciones se deben cambiar solamente la implementación. De esta forma podemos además traer implementaciones de otro sistema e integrarlo con mucha facilidad.

Tomando en cuenta el Context y la base de datos decidimos que cada entidad que contiene una colección de otra entidad como es Accommodation tiene varias Bookings. Decidimos que la mejor opción es que una entidad conozca las entidades que dependen de ella. De esta forma si de a futuro se necesitan las reservas asociadas a un alojamiento podemos pedirlas al alojamiento podemos conseguir todos los códigos de reserva asociados a este alojamiento.

Para el caso de los estados de las reservas decidimos crear una entidad BookingStage, esta tiene el estado de la reserva, administrador que hizo el cambio y la fecha. De esta forma tenemos para cada Booking un historial de los cambios donde tenemos el reporte de los cambios de la Booking, de esta forma en un futuro se precisaría esta información pueda tenerle. Pensamos que guardar información nunca está de más.

También decidimos crear una entidad Tourist , esta está asociada a los Bookings, al igual que entidades anteriormente explicadas esta sirve para asociar un email a varias Bookings. De momento esto puede no ser necesario. Pero pensamos que en un futuro tendremos una entidad que albergue todos los bookings asociados a un email puede ser muy útil en otras funcionalidades.

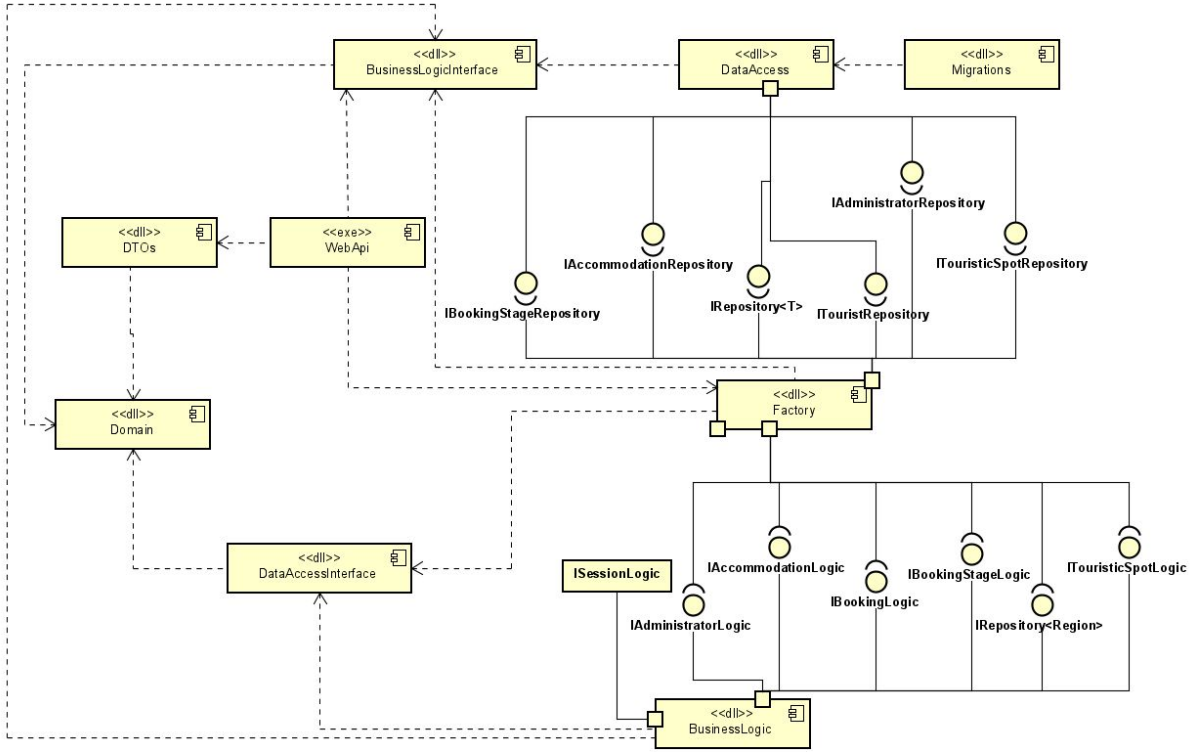
En el caso de los huéspedes de la Booking decidimos aplicar una lista de objetos de tipo Guest, guest tiene un int:Amount y double:Multiplier. Si lo hacemos de esta forma tenemos de manera extensible los tipos de huéspedes. En la documentación se pedía tres tipos Adulto, Niño y Bebe, pero no sabemos si en un futuro tenemos un tipo Anciano con nuestra estructura se podría agregar sin problema. Nosotros tomamos en cuenta que la persona que usará la API sabe sus multiplicadores por tipo de huésped y a su vez sabe mapear un multiplicador con un tipo de huésped.

Para resolver el tema del precio máximo de la reserva nuestra estructura de lista de huéspedes es tan simple como iterar sobre ella y sumar Multiplier*PricePerNight de cada elemento de la lista de huéspedes.

Para manejar las excepciones aplicamos un ExceptionFilter para tener un catch del mismo y este retorne los códigos de error correspondientes al usuario. Para no tener muchos Catch en esta clase aplicamos una clase APIException propia que se extiende a Exception. Esto nos permite tener una tipo de excepción la cual también tiene un código de respuesta HTTP. De esta forma cuando se genera una excepción podemos pasarle el mensaje que deseamos crear y que código HTTP deseamos que retorne. A su vez, al ver que muchas excepciones eran similares creamos otras excepciones personalizadas que aplican APIException como por ejemplo NotFoundException y NullInputException.

Mediante la clase ServiceFactory pasamos la responsabilidad de crear las dependencias a una clase externa a WebApi marcando correctamente las responsabilidades y para así no romper SRP. La factory se encarga de la inyección de dependencia de las varias interfaces de sistemas. Elegimos hacer los métodos Static ya que pensamos que tener una instancia de esta clase no tiene ningún valor.

Diagrama de Componentes



Se decidió la división de componentes según sus responsabilidades y funcionalidades. Cada componente se ocupa de una sola cosa ya sea interacción con la base de datos o el recibimiento de consultas del usuario.