Diseño de aplicaciones 2

Obligatorio 2

Facundo Pujol 226033
Serafín Revetria 209143

Docente: Gabriel Piffaretti, Daniel Acevedo, Nicolas Hernandez

Facultad ORT Uruguay

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

Indice

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, como puede ser: “playas”, “históricos”, entre otros. Nosotros crearemos una API a la cual 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, hacer 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, para que el mismo no pueda recibir más reservas. Estas nuevas funcionalidades generan el nuevo concepto de acomodamiento, reserva, huésped, entre otros. Toda esta información se almacenará dentro de una base de datos.

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.

Para la segunda parte de esta entrega, se agregaron tres puntos claves. Primero, cada acomodamiento debe tener un puntaje asociado, que se determinara siendo el mismo el promedio de los puntajes otorgados en las reseñas de los huéspedes. Segundo, los administradores podrán solicitar un reporte que devolverá los acomodamientos que tengan reservas hechas dentro de las fechas solicitadas, siempre y cuando su estado de reserva no sea ni “rechazado” ni “expirado”. Y por último, se solicitó que se puedan agregar acomodamientos desde cualquier tipo de fuente, es decir que dado un archivo de tipo, por ejemplo .json, se agreguen los acomodamientos y el punto turístico asociado. Pero no podemos realizar un código que sea capaz de levantar cualquier tipo de archivo por lo que se tuvo que desarrollar una nueva funcionalidad que permite extender nuevas formas de importación implementadas por un tercero. Esto permite que si el cliente desea agregar un acomodamiento desde un archivo .txt, teniendo una implementación de nuestra interfaz podría importar el archivo .txt, y así con cualquier otro tipo de archivo.

Agregado a todo esto, se realizó la interfaz de usuario correspondiente a la página mediante un proyecto de Angular que implementa todas las funcionalidades del sistema.

Diagrama de Descomposición de Namespaces

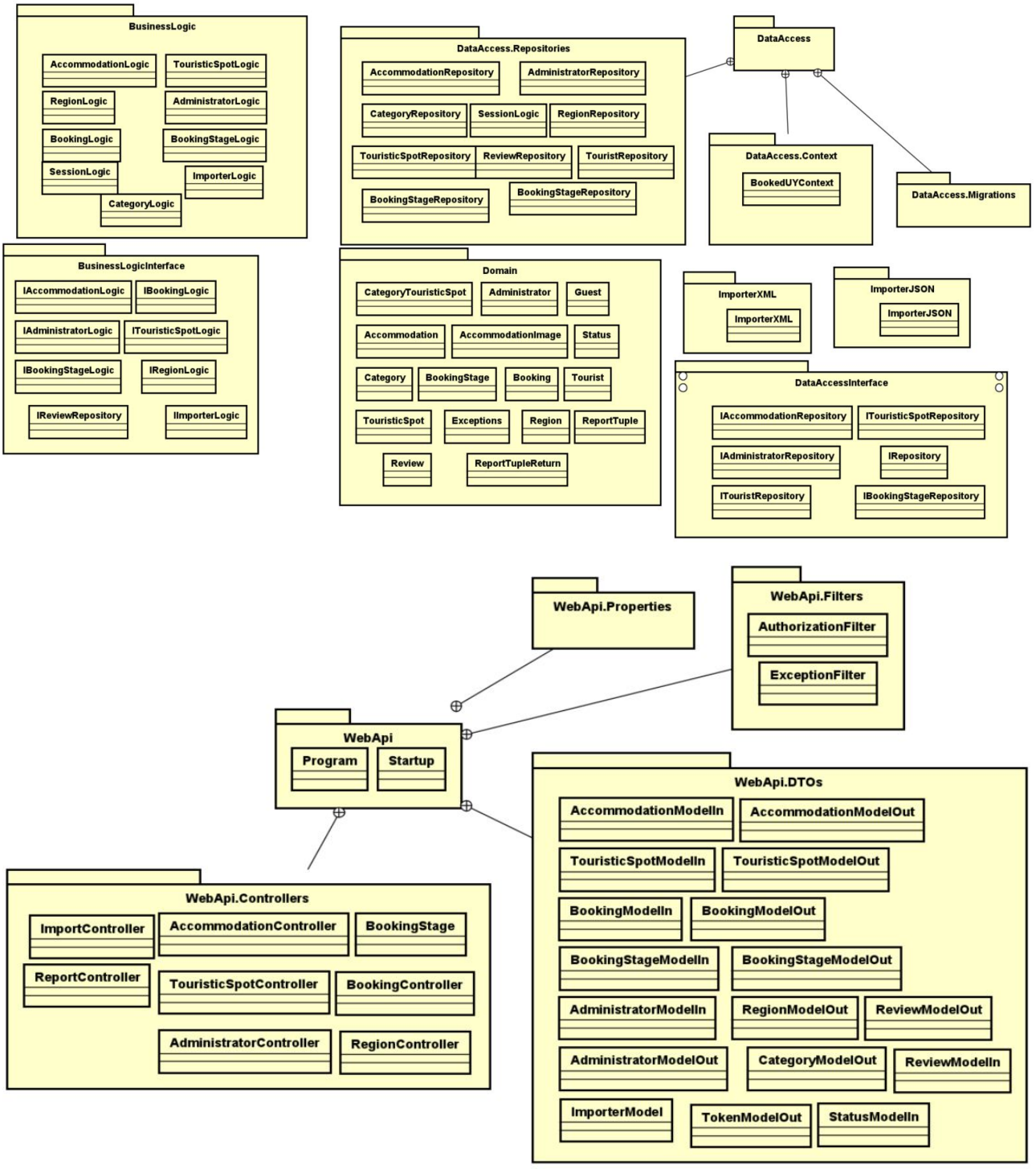
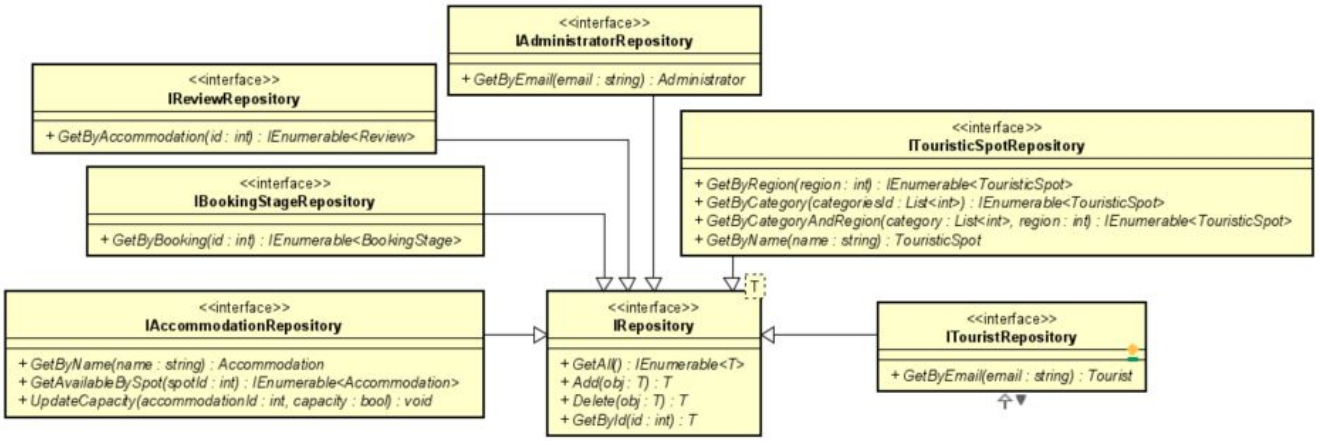


Diagrama general de paquetes

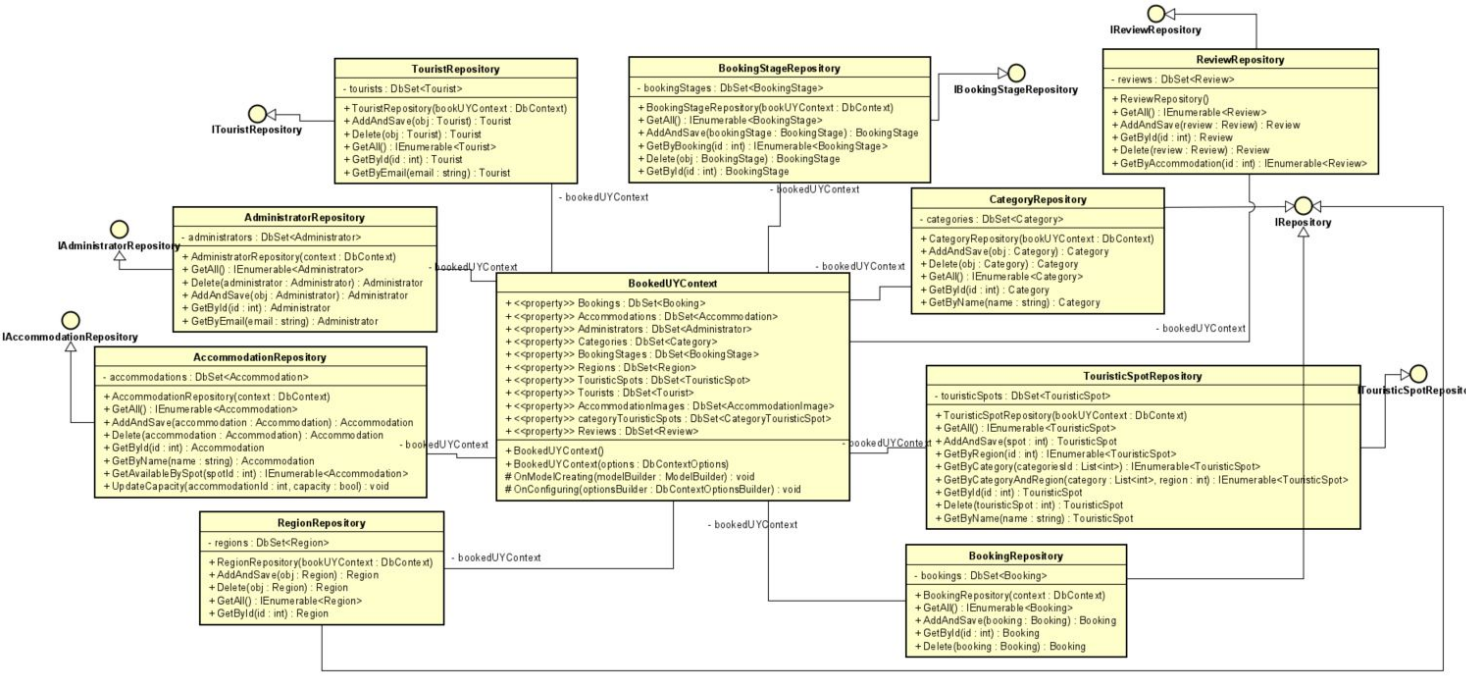
DataAccessInterface

Paquete encargado de definir las interfaces con sus métodos encargados del acceso a los datos dentro de la base de datos. Listado de clases por paquete en el anexo.



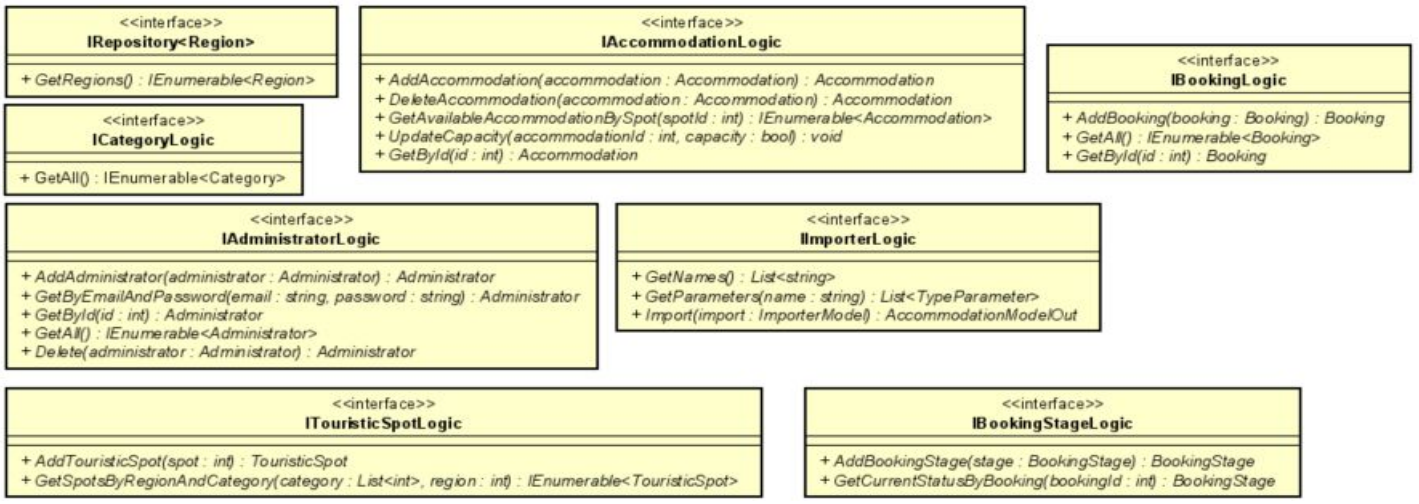
DataAccess

Implementación de las interfaces en DataAccessInterface, contiene el modelo de acceso a datos de cada entidad del sistema. Listado de clases por paquete en el anexo.



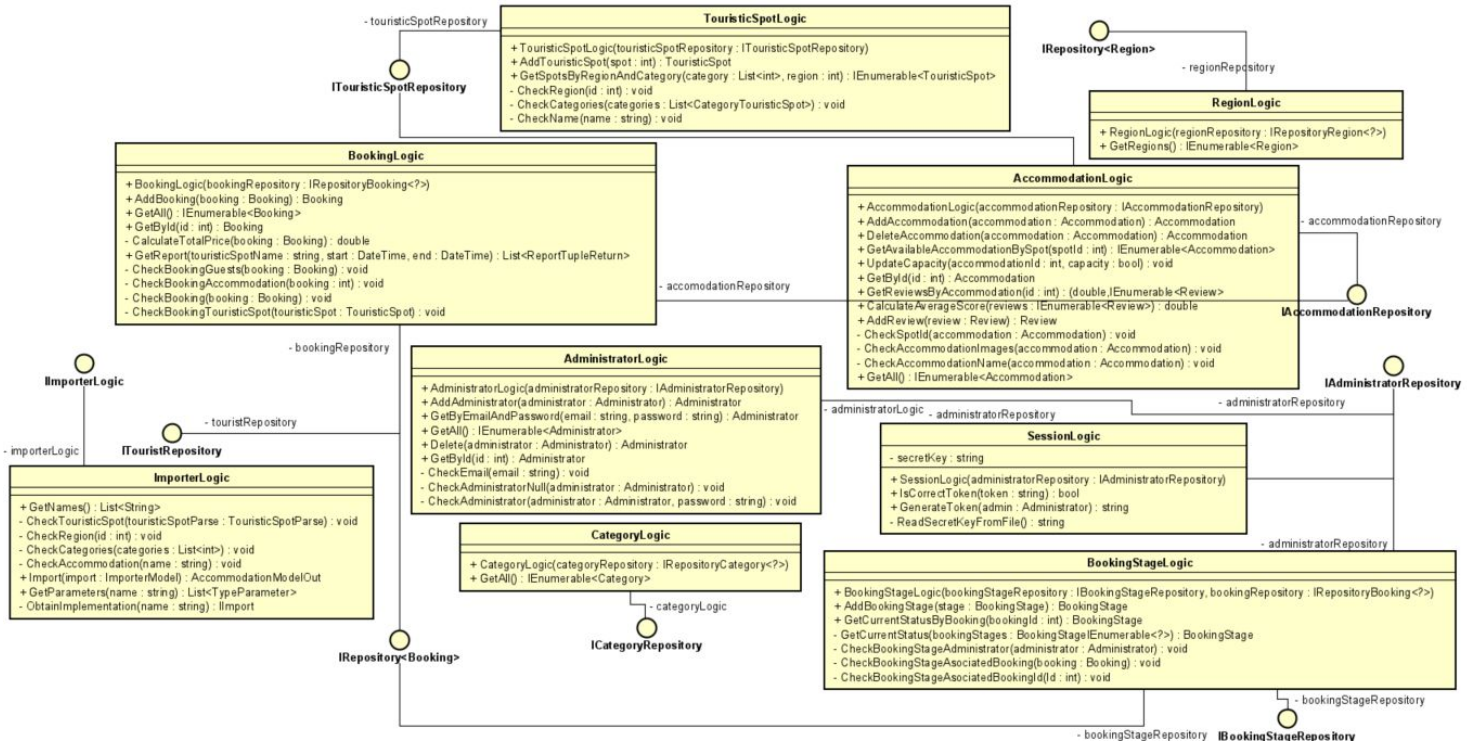
BusinessLogicInterface

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. Listado de clases por paquete en el anexo.



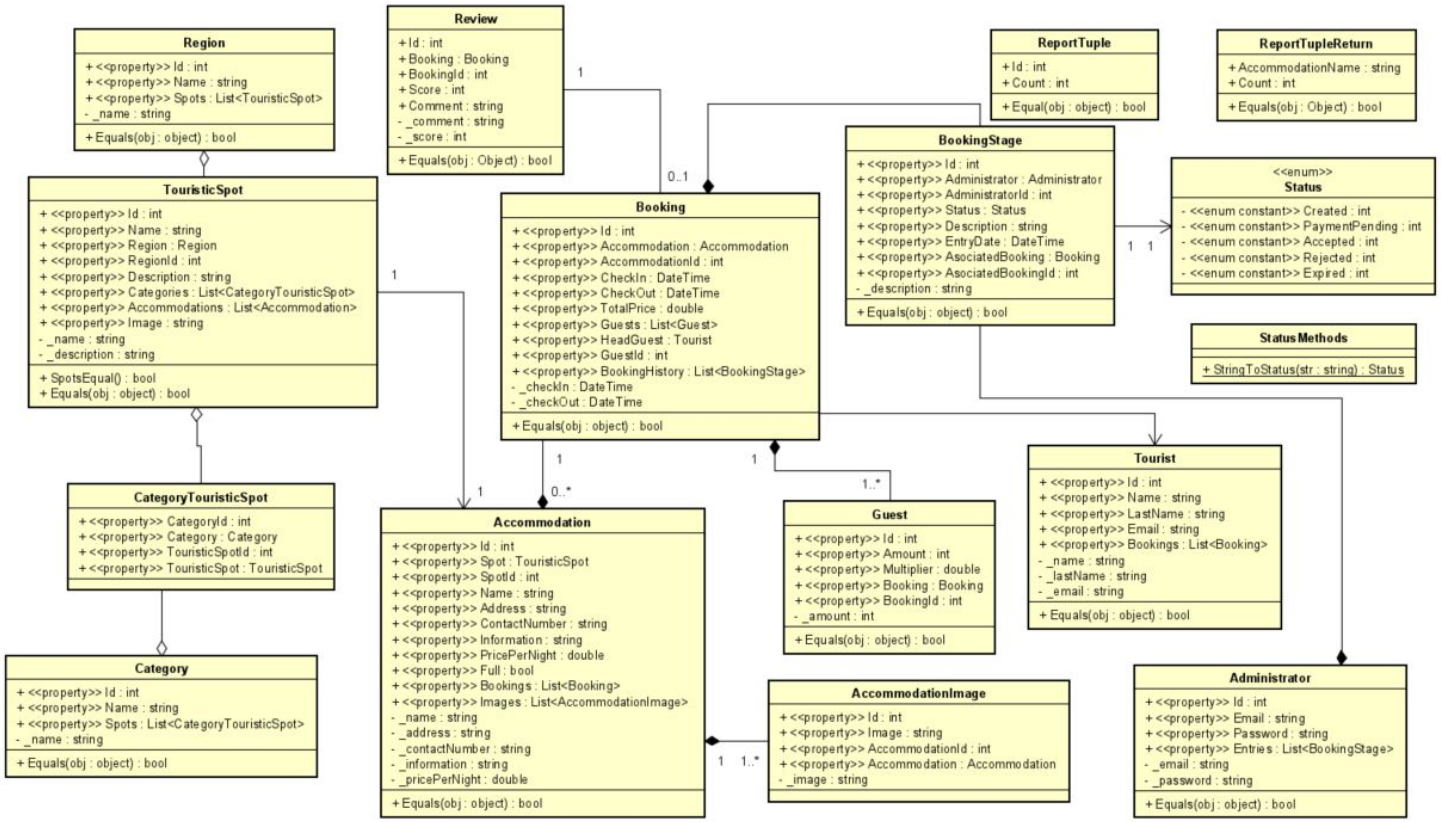
BusinessLogic

Implementación de las interfaces de BusinessLogicInterface con la implementación de métodos necesaria para nuestro sistema BookedUY. Listado de clases por paquete en el anexo.



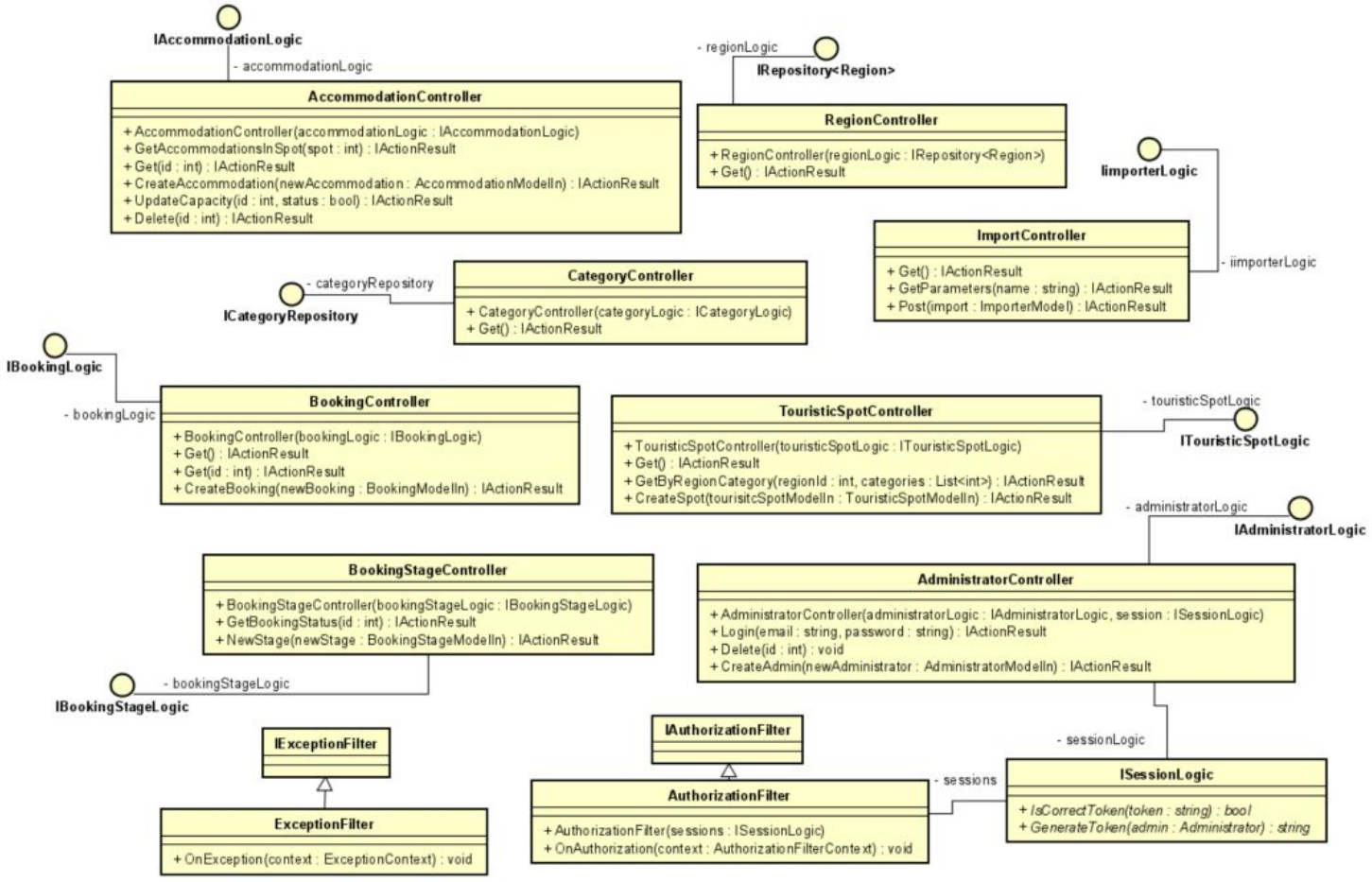
Domain

Paquete que contiene todas las entidades de negocio de nuestro sistema con sus respectivas relaciones entre ellas. Listado de clases por paquete en el anexo.



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. Listado de clases por paquete en el anexo.



Factory

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

Migrations

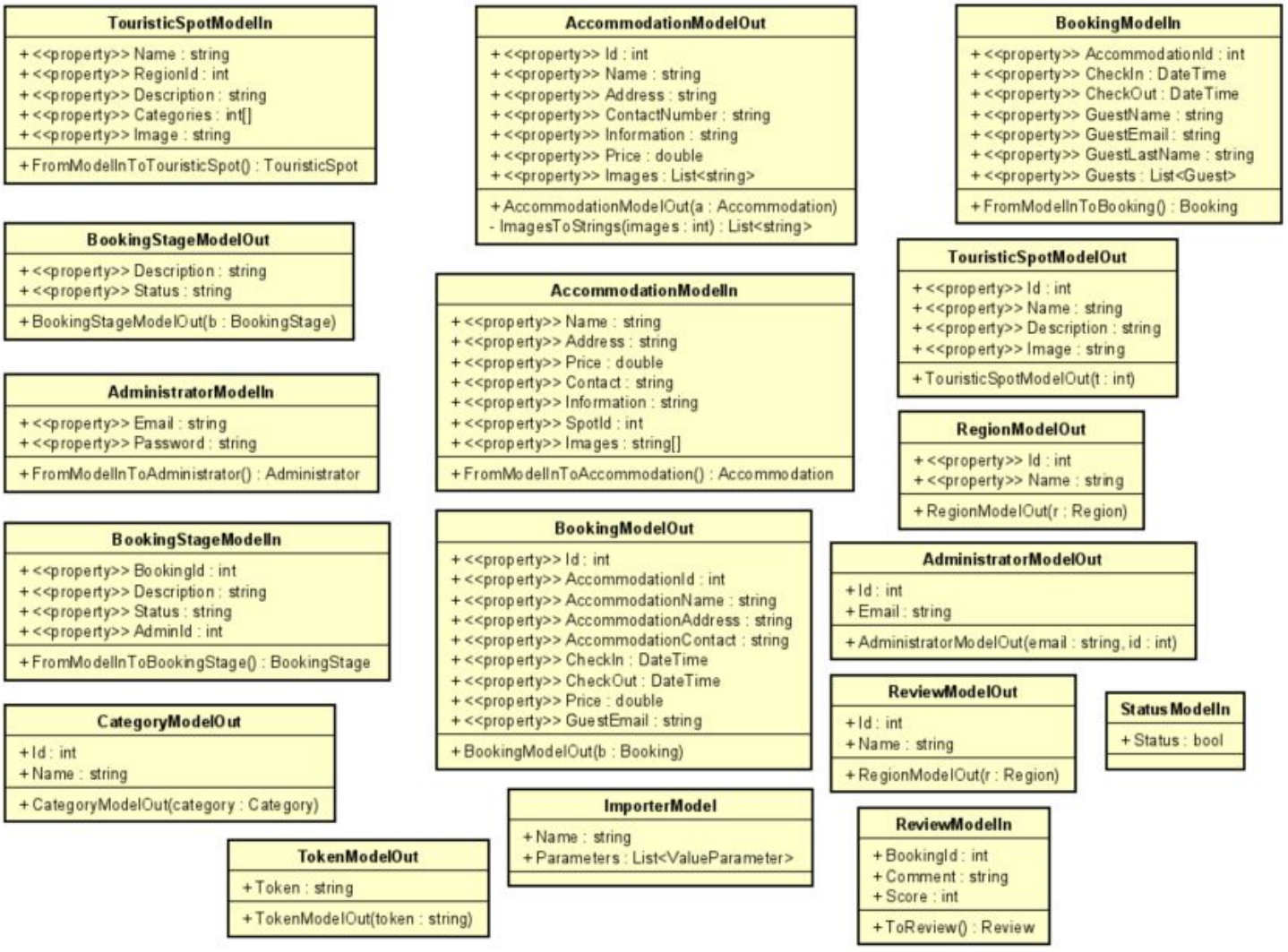
Este paquete está creado para la generación de migraciones de la base de Datos. Para definir correctamente quien es responsable de este proceso. Listado de clases por paquete en el anexo.

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. Listado de clases por paquete en el anexo.

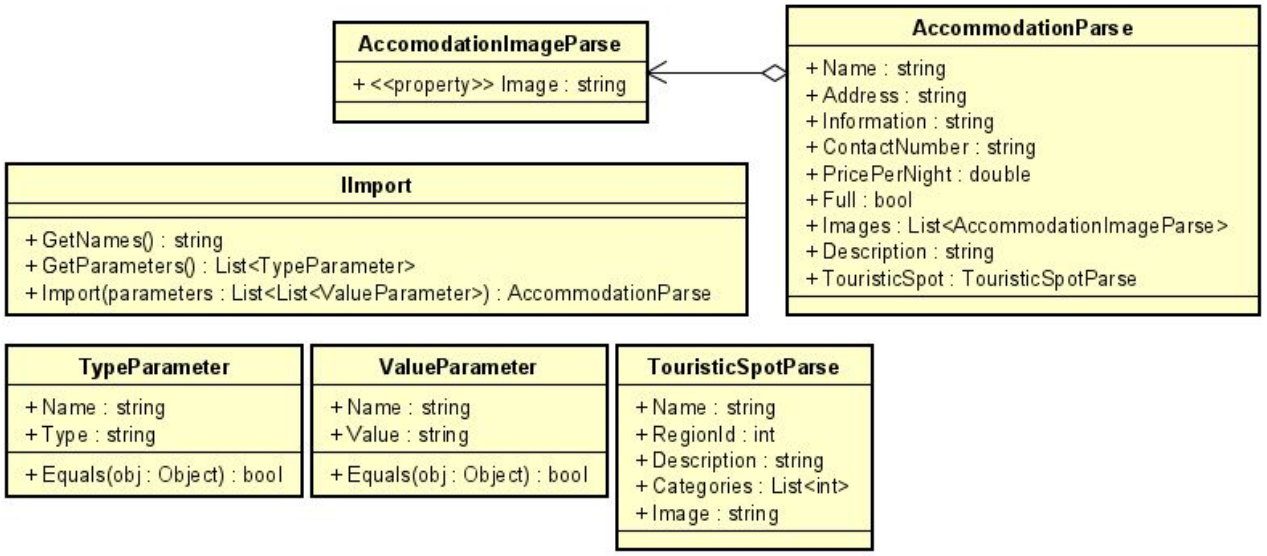
WebApi.DTO

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. Listado de clases por paquete en el anexo.



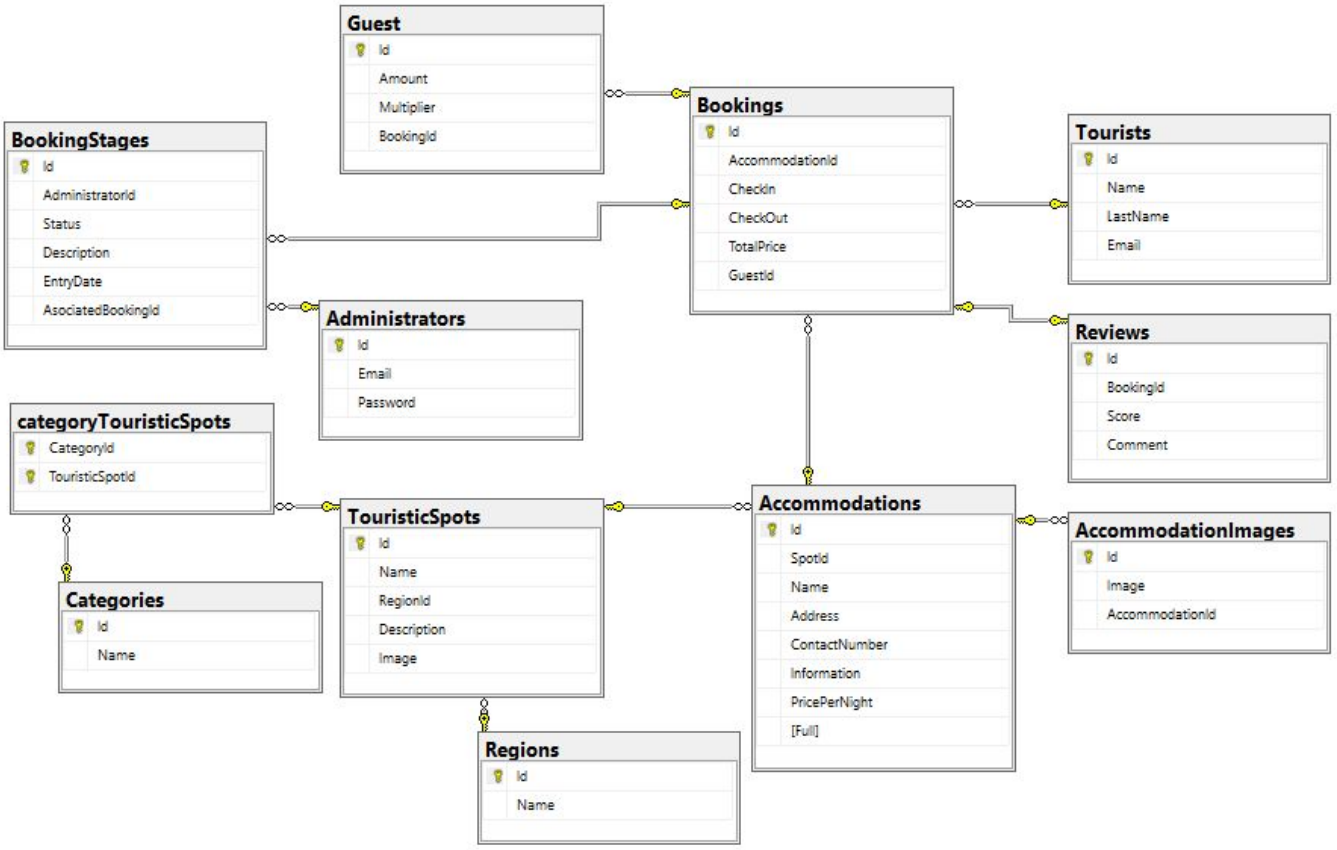
ImporterInterface

En este paquete se realizó todo lo necesario para el funcionamiento de reflection, acá podemos ver que hay una interfaz IImport qué es la que los 3ros deberán implementar para generar sus propios importadores. Tambien tenemos todos los Parseos necesarios, es decir AccommodationParse, AccommodationImageParse y TouristicSpotParse. Que es a lo que nos devuelven los importadores con la información levantada del file.



Modelo de tablas

Debajo se encuentra el diagrama de tablas de la base de datos en el cual se ve cada una de las tablas que utilizamos. Se pueden apreciar los atributos de cada tabla y las relaciones entre las tablas. Esto fue generado a través de Entity Framework, y este diagrama fue generado a través del Management Studio.



Diagramas de Secuencia

Los diagramas de secuencia son diagramas de interacción, por ende capturan la interacción entre objetos. En los mismos se destaca el orden de la interacción. Es decir, que es de gran importancia en que orden un objeto interactúa con otro.

En este caso podremos ver generalmente la interacción entre el Controller con la BusienssLogic, para luego ver la interacción entre la BusinessLogic con la DataAccess. Los Controllers son los encargados de resolver las diferentes consultas HTTP, y ellos son los que interactúan con la capa lógica. En la BusinessLogic se maneja todo lo que es logica de negocios, es decir validaciones, cálculos, en otras palabras, se prepara el objeto para luego agregarlo. Siguiendo con esto, la DataAccess, que es quien recibe el objeto pronto se encarga de agregarlo a la base.

Debajo se encuentran los diagramas de secuencia de las funcionalidades claves de la solución

Diagrama de secuencia de funcionalidad: GetAvailableAccommodationBySpot

Consideramos que esta funcionalidad es principal ya que es la que se encarga de mostrar acomodaciones dado un punto turístico (como dice el nombre). Esto es lo que le da la posibilidad al cliente de ver donde se puede hospedar dentro del lugar al que quiere ir, ya descartando aquellos que se encuentran sin capacidad. Y es lo que diferencia a nuestra página con la de “Uruguay Natural”, permitiendo ir más allá de ver a donde ir. En el mismo se ve todo el transcurso desde el Get que pasa por el AccommodationController, luego por la BusinessLogic de Accomodamiento para luego pasar por el AccommodationRepository que es quien sabe como traer las respectivas acomodaciones.

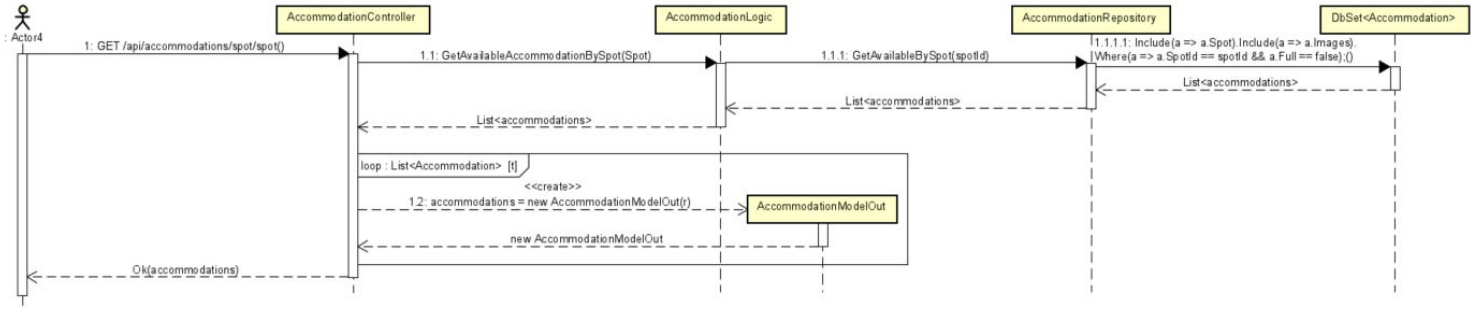


Diagrama de secuencia de funcionalidad: CreateBooking

Siguiendo con la idea anterior, esta funcionalidad también la consideramos de gran importancia ya que es la funcionalidad que permite registrar una reserva de un cliente hacia un acomodamiento. En este caso se puede ver como hay un ciclo más largo ya que primero se transforma el BookingModelIn, es decir lo que entra, en una entidad del dominio, para luego sí comenzar con el proceso para crear esta nueva reserva. Podemos ver como la BusinessLogic de Booking realiza las validaciones correspondientes y realiza también el cálculo del precio total según la cantidad y tipo de turistas a hospedar. Una vez que esto se realizó, ahora si se interactúa con el BookingRepository para que el mismo se encargue de agregar la reserva, ya que la capa de DataAccess es la encargada de comunicarse con la base de datos.

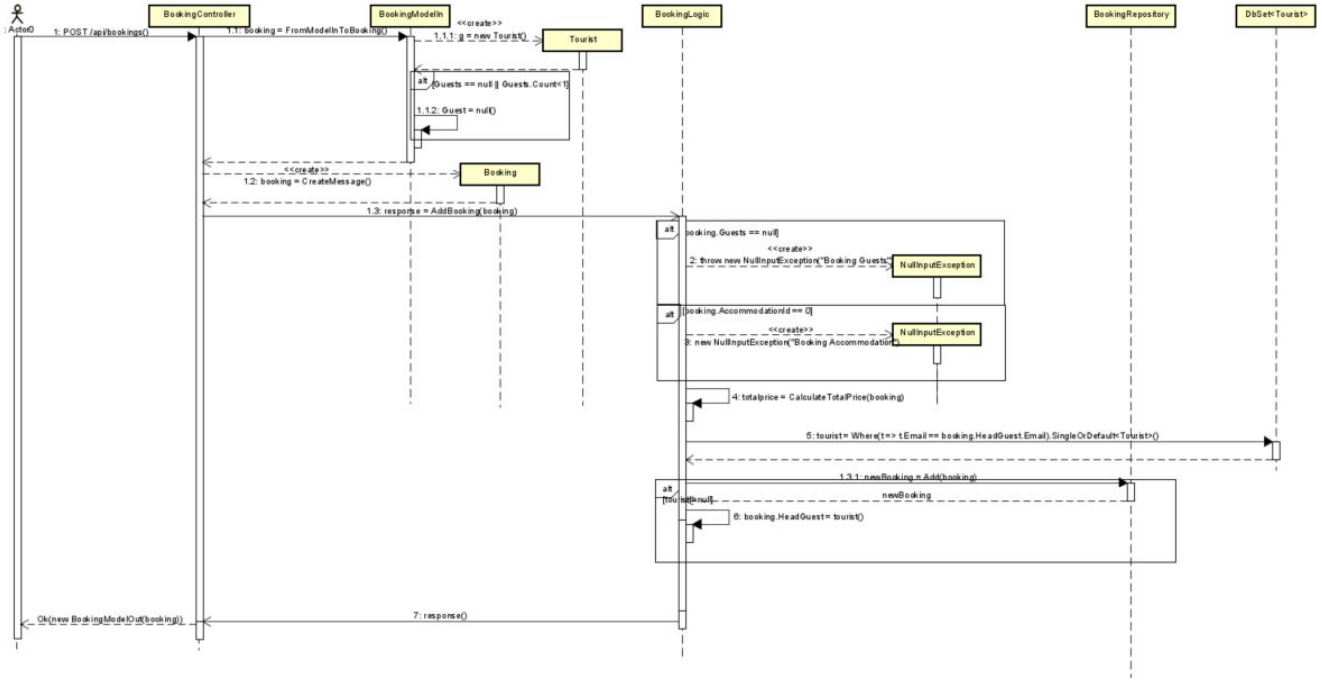


Diagrama de secuencia de funcionalidad: CreateBookingStage

Otra de las funcionalidades a destacar consideramos que es esta, ya que es la encargada de permitir a los administradores cambiar el estado de una reserva, es decir que proporciona a nuestro sistema una interacción entre el cliente y el acomodamiento, ya que el estado de su reserva puede ir cambiando, por ejemplo si aun no se pago, se marca la reserva en estado de “PaymentPending”, o en caso de haber sido rechazada por alguna razón, como puede ser un método de pago erróneo, el estado pasaría a ser “Rejected”.

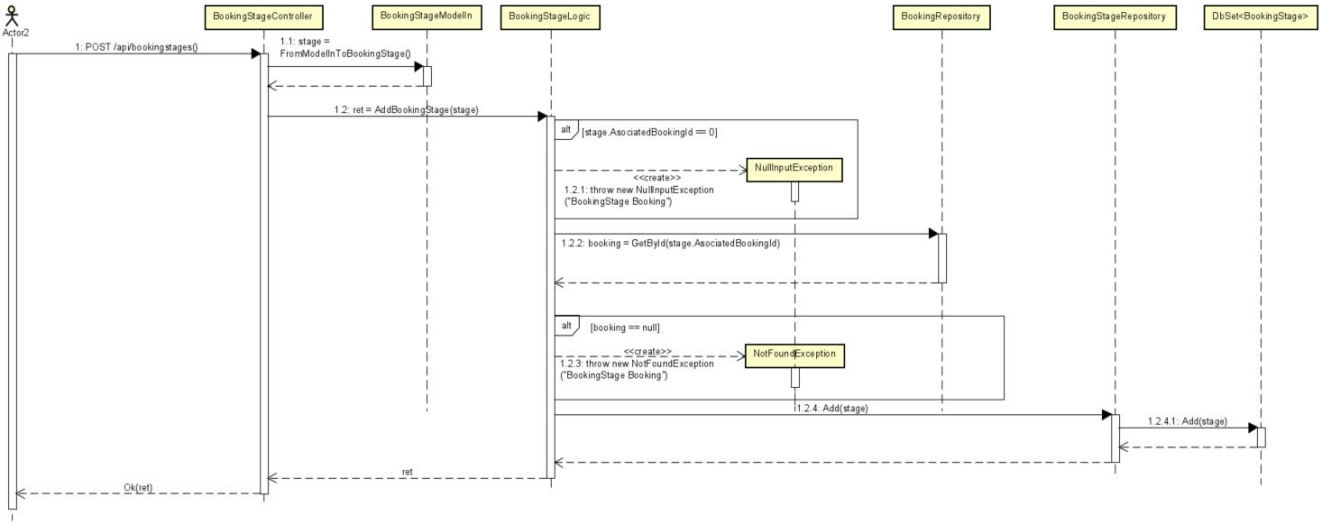
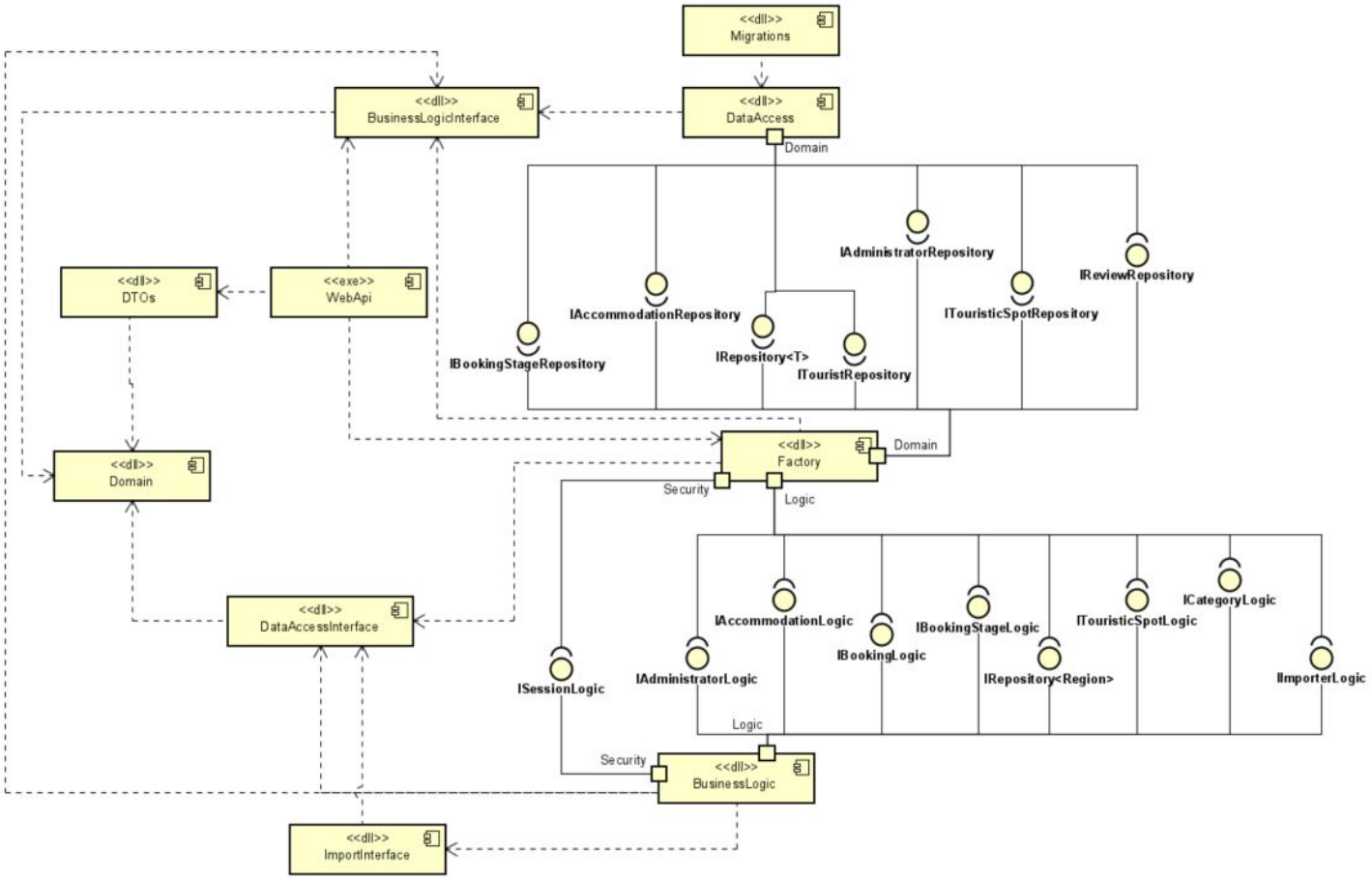


Diagrama de componentes

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.



Justificación y explicación del diseño

Para la funcionalidad en la que se propone poder importar desde cualquier tipo de archivo un acomodamiento, se utilizó reflection. Reflection consiste en crear una interfaz base, en este caso es la interfaz de los importadores, es decir tiene un método Import, un método GetName y un método GetParameters. La idea de esta herramienta es que se pueda implementar esta interfaz y utilizarla en nuestro sistema. Si por ejemplo un cliente quiere importar un archivo .o9sj (ejemplo), debe implementar la interfaz, generar una dll que la implemente y colocarla en la carpeta DLL del sistema, para que luego el sistema pueda utilizar esta implementación y levantar este file.o9sj. Esto genera una gran extensibilidad ya que podremos levantar cualquier tipo de archivo siempre y cuando se tenga la implementación del mismo, y de no tenerla se puede implementar para luego utilizarla.

El principal patrón de diseño utilizado es el patrón Facade, este lo podremos ver reflejado en el paquete WebAPI. Esta es claramente una fachada del sistema completo, este paquete contiene una serie de EndPoints los cuales son puntos de acceso al sistema. A pesar de que son varios puntos de acceso se encuentran únicamente en un paquete por lo que la “cara” del sistema es WebAPI. Se encontró toda la complejidad del sistema y se centró todas las salidas y entradas en una única entrada. Esto lo podemos ver en el diagrama de descomposición de Namespaces, en la capa superior tenemos únicamente el paquete WebAPI con su subpaquete WebAPI.DTO y el Factory. En esta capa se encuentran los paquetes de primera interacción con el usuario, en el caso de Factory se encuentra en este paquete porque se encarga del DI. Podemos ver

como WepAPI utiliza varios paquetes en capas debajo como la lógica porque WepAPI recibe las interacciones y las comunica a paquetes inferiores de lógica y de acceso a datos.

La implementación de Reflection y de los Importers se realizó mediante el patrón Strategy, como los dll para los importers deben servir cada uno para su extensión de archivo o método de importación, el código encargado de la importación debe estar abierto para recibir cualquier tipo de importación y saber que dll utilizar para lograrla. Por lo tanto, el ImporterInterface es una interfaz la cual no se le asigna ninguna implementación sino que esta implementación se decide dependiendo del tipo de importación se está utilizando en el momento. Como podemos ver en el diagrama de clases del paquete ImporterInterface no hay una implementación concreta de la interfaz IImport sino que se espera que se agreguen dlls que contengan implementaciones concretas de esta interfaz.

En la User Interface los varios servicios utilizados siguen el patrón Singleton, ya que estos son una única instancia de la misma clase, cuando los componentes consultan variables o métodos en estos servicios siempre están consultando a una misma instancia de estos servicios convirtiendo a estos en Singletons.

La primera mejora que se logró es corregir todos aquellos puntos de nuestro codigo que podrian llegar a violar el Single Responsability Principle, se movieron algunos metodos de clases a otras las cuales decidimos despues de discutir que debian encargarse de ello, como por ejemplo decidimos que el guest se encarge utilizando su multiplier y sabiendo su cantidad de Guests de su tipo debia pasarle a la Logic el calculo de su respectivo costo.

Pusimos gran enfoque en mejorar nuestro código aplicando más rigurosamente los principios de Clean Code. De la mano de las correcciones de la anterior entrega y con otras decisiones en equipo empezamos arreglando todos los tests poniendo especial foco en seguir los principios AAA separando el código de cada test en Arrange, Act y Assert poniendo un “enter” al terminar cada sección de la prueba. Seguimos con el refactoring del código extrayendo a métodos privados todas aquellas lógicas que infringieron con la responsabilidad única de los métodos. Como por ejemplo todos los métodos de la lógica tenían en su cuerpo la verificación de los datos de entrada, se decidió hacer varios métodos privados Check para verificar los datos entregados desde la WebAPI. También se arreglaron todas las variables que no contengan nombres mnemotécnicos en algunas clases de Logic y de WepAPI.

Como explicamos anteriormente la aplicación del patrón Strategy en ImporterInterface aportó a la extensibilidad de nuestro sistema ya que realizar cambios a nuestro nuestro código base podemos implementar diferentes y nuevas formas de importación, por lo que la extensibilidad de esta lógica del sistema es importante.

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. 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.

En términos del diseño no se realizaron grandes cambios, esto se debe según nuestro análisis que gracias a algunas previsiones que tuvimos en la primera parte del obligatorio ayudaron a que no sea necesario cambios de diseño. Más adelante expresaremos en términos de cada cambio pedido con qué dificultad y de qué forma se lograron.

La primera funcionalidad que decidimos realizar fue la creación de reseña, decidimos crear una relación única entre Booking y Review. Entonces de esta manera cada Review conoce una Booking, entonces al conocer la Booking y esta última también está relacionada con un Tourist se puede conseguir fácilmente el nombre del Guest. A su vez los Bookings están relacionados a un Accommodation por lo que teniendo en conocimiento el Id del Accommodation podríamos solicitar todas la Bookings de ese Accommodation y de estas Bookings solicitar las Reviews que contengan su Id en la referencia.

Precisamos crear una entidad de negocio (Review), una clase repositorio, una lógica y dos DTOs. Decidimos que el controller de Accommodation se encargará de las Reviews ya que el Id de estas son la referencia al pedir todas las reseñas de un Accommodation. Debimos realizar pequeños cambios en el AccommodationController y el AccommodationLogic para cumplir con la funcionalidad.

La siguiente funcionalidad que decidimos hacer es el nuevo tipo de Guest que serían los Jubilados (o Elderly), esta funcionalidad no tuvo gran impacto en el diseño debido a que la lógica del multiplicador la contiene la entidad Guest. De esta manera tuvimos que cambiar solamente el algoritmo de cálculo del precio total.

Para los reportes decidimos que se realizarán con una única sentencia LINQ para así mejorar el rendimiento al momento de consultar un reporte, para esto se precisa únicamente crear un modelo de salida. El modelo de salida era totalmente necesario ya que para que la consulta LINQ pueda retornar un COUNT y un dato de la tabla tuvimos que crear un modelo que reciba el COUNT y el Id del accommodation para asi despues mapear el Id con el nombre del mismo para después pasarlo como la respuesta. Realizamos esta lógica y consulta desde el punto de vista del Booking ya que consideramos que podemos calcular cómodamente las bookings que cumplen con la fecha y al conocer su Accommodation podemos agruparlos por el AccommodationId.

```
public IList<ReportTuple> GetReport(int touristicSpotId, DateTime start, DateTime end)
{
    return this.bookings.Include(b => b.Accommodation).Include(b => b.BookingHistory).
        Where(b => b.Accommodation.SpotId == touristicSpotId
        && b.BookingHistory.OrderByDescending(c => c.EntryDate).FirstOrDefault().Status != Status.Rejected
        && b.BookingHistory.OrderByDescending(c => c.EntryDate).FirstOrDefault().Status != Status.Expired
        && ((b.CheckIn >= start && b.CheckIn <= end) || (b.CheckOut <= end && b.CheckOut >= start) || (b.CheckIn<=start && b.CheckOut>=end)))
        .GroupBy(b => b.Accommodation.Id).Select(b => new ReportTuple(){ Id = b.Key, Count = b.Count() })
        .OrderByDescending(b=>b.Count).ThenBy(b=>b.Id).ToList();
}
```

Calidad de diseño

Las justificaciones van acompañadas de los diagramas que se encuentran debajo:

El proyecto DataAccessInterface es muy abstracto y estable. El ser estable se debe a la cantidad de relaciones entrantes, es decir varios paquetes dependen de él y él depende de muy pocos. En este caso la BusinessLogic depende de él, y él

no depende de nadie. También podemos ver que es abstracto, y esto se debe a que el 100% de las clases son abstractas e interfaces. Por todo esto, podemos deducir que es más difícil de cambiar.

El proyecto SessionInterfce es muy abstracto y estable> El ser estable se debe a la cantidad de relaciones entrantes, es decir varios paquetes dependen de él y él depende de muy pocos. En este caso la WebApi depende de él, y él no depende de nadie. También podemos ver que es abstracto, y esto se debe a que el 100% de las clases son abstractas e interfaces. Por todo esto, al igual que con DataAccessInterface, podemos decir que es difícil de cambiar, ya que afectaría a todos los que dependen de él.

El proyecto BusinessLogicInterface también es abstracta ya que el 100% de las clases son abstractas o interfaces. Se encuentra al límite entre estabilidad e inestabilidad. La WebAPI depende de él.

El proyecto ImportInterface es poco asbtracto y estable, esto no es bueno ya que se considera que no son extensibles y de cambiar afecta a muchos

El proyecto Domain está en la misma situación que el ImportInterface, en este caso se puede explicar ya que hace referencia a las entidades de negocio. Tiene una muy baja abstracción ya que ninguna clase es abstracta, y tiene una alta estabilidad ya que muchos dependen de él, lo que es de esperar ya que son las entidades de negocio. Zona de dolor, se considera que no son extensibles y de cambiar afecta a muchos.

El proyecto Factory tiene una baja abstracción y una alta inestabilidad. la alta inestabilidad lo vuelve muy extensible, ya que pocos dependen de él. Y por otro lado podemos decir que es un paquete concreto, dado su baja abstracción. Este paquete se encuentra ubicado sobre la secuencia principal, lo que es bueno.

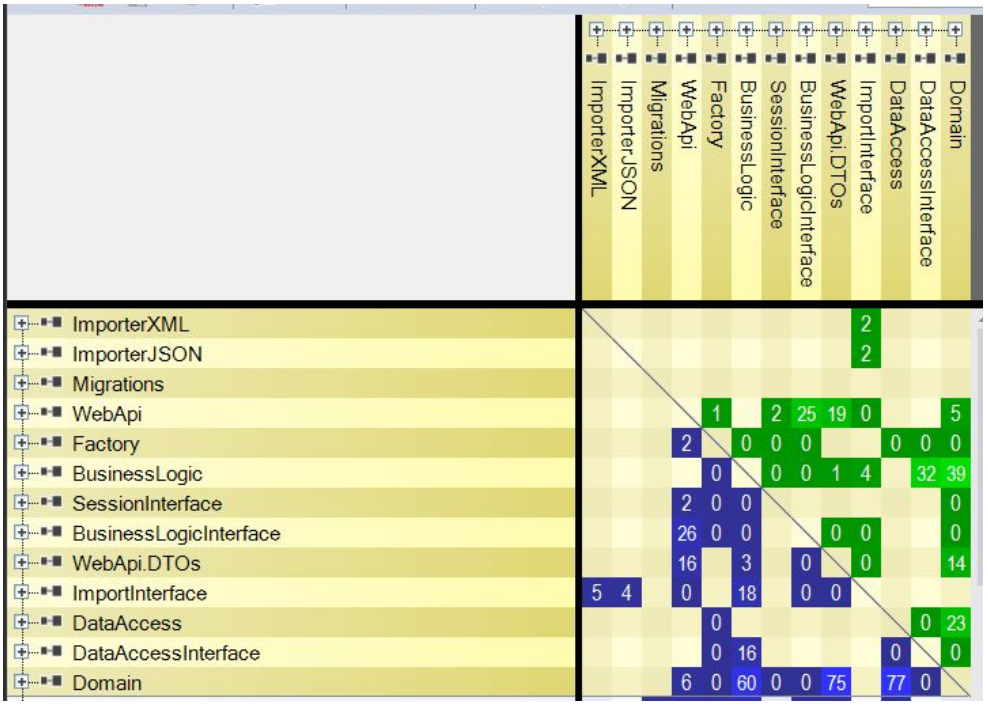
El proyecto BusinessLogic se encuentra sobre la secuencia principal ya que tiene baja abstracción y es inestable. Como se explicó anteriormente, estas características hacen del paquete, un paquete muy extensible, afectando a pocos en caso de cambiar.

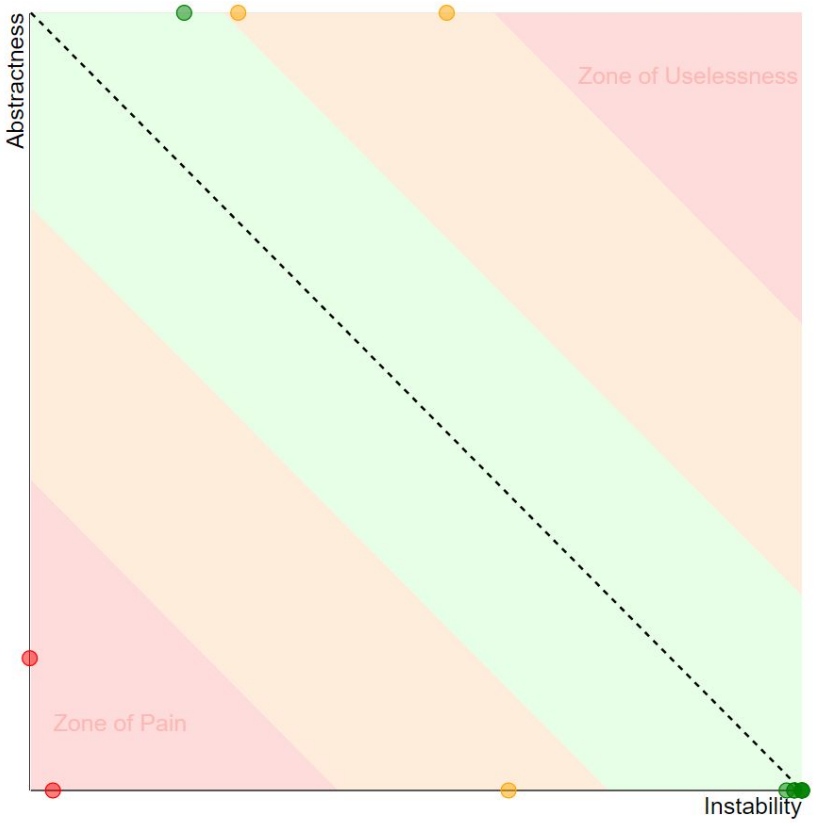
El proyecto WebApi tiene una baja abstracción y es inestable, lo que lo sitúa en la secuencia principal, lo que indica que es extensible y que pocos dependen de él.

Y por último tenemos el paquete DataAccess que también se encuentra situado en la secuencia principal debido a su baja abstracción y su inestabilidad. Como en los casos anteriores, su baja abstracción se da ya que no contiene clases abstractas,

mientras que su inestabilidad se da ya que nadie depende de él y él depende de muchos. Por lo que lo convierte en un paquete muy extensible, ya que sus cambios no afectarian a nadie

Debajo podemos ver la matriz de dependencias autogenerada por NDepend, donde se pueden corroborar todos los datos utilizados anteriormente. También contamos con el diagrama de abstracción versus inestabilidad, donde se puede corroborar que paquetes se encuentran en la zona de dolor y cuales en la secuencia principal.





Explicación de Importer

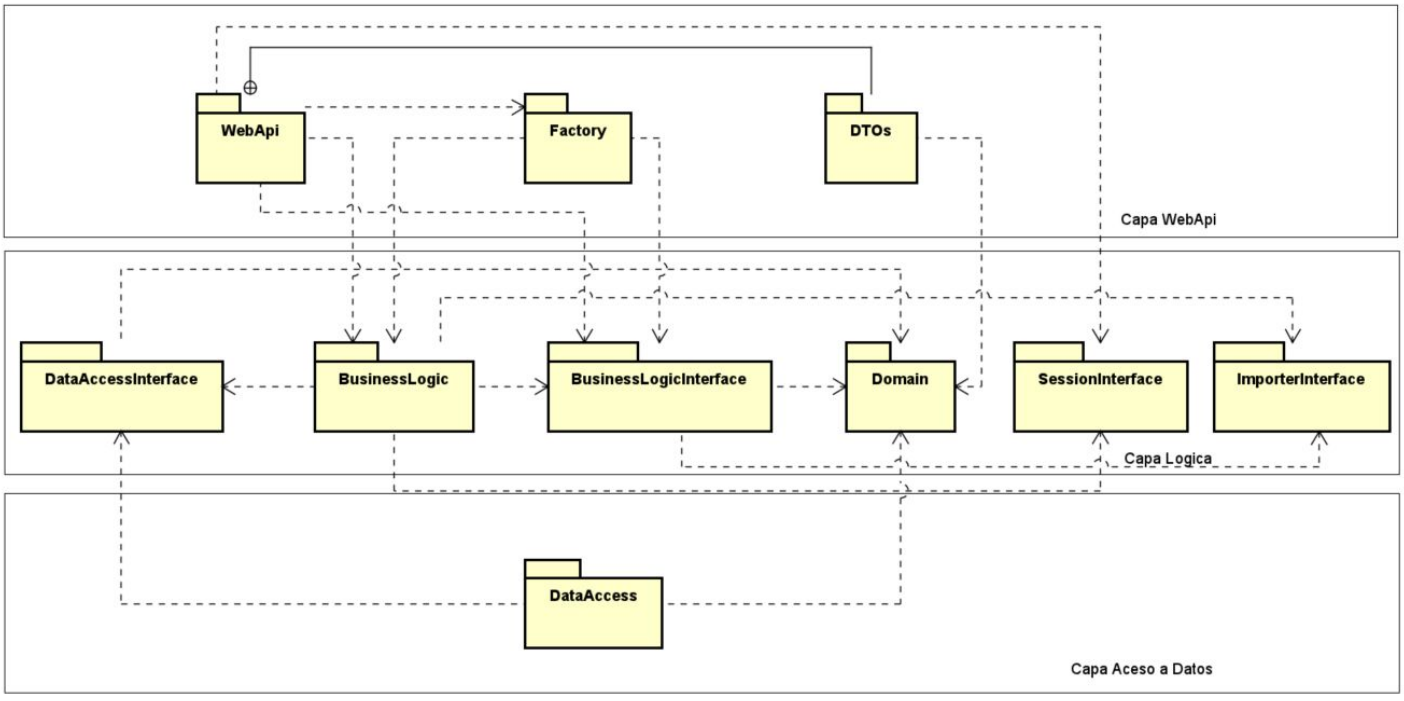
Para poder agregar un Importer nuevo debemos crear una dll y agregarla a la carpeta DLL situada en BookedUY. Esta dll debe implementar la interfaz IImport, especificando el método GetName que se encarga de decir el nombre del importador, en los casos ejemplo son “JSON” y “XML”. Luego debe implementar el método GetParameter que indica que parámetros debo conocer para hacer el import, en los casos ejemplo ambos solicitan como parámetros la ruta donde se ubica el file. Se intentó hacer esto lo más extensible posible y no limitar solo a una ruta ya que por ejemplo se puede querer importar de una base de datos y necesitar algo más que una ruta, como puede ser una contraseña, o en otro caso un puerto. Y por último se encuentra el método más importante, el Import que se encarga de transformar lo que hay en el file a un Accommodation parse, que nuestro sistema reconocerá y podrá utilizar y validar para luego agregarlo

Anexo

Referencias

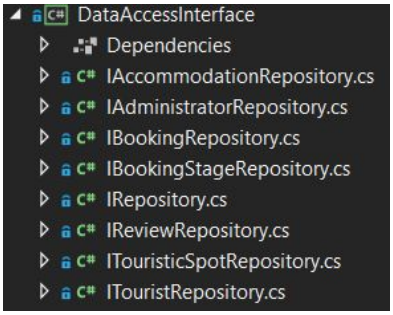
Diseño de aplicaciones 2 entrega de obligatorio 1 Pujol Revetria

Diagrama de Paquetes

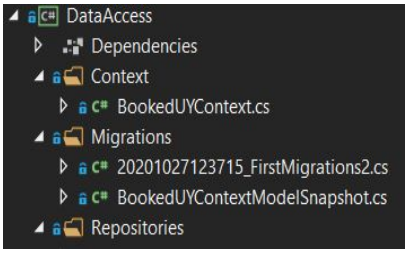


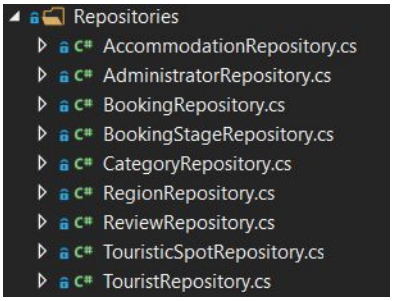
Listado de clases por paquete

DataAccessInterface

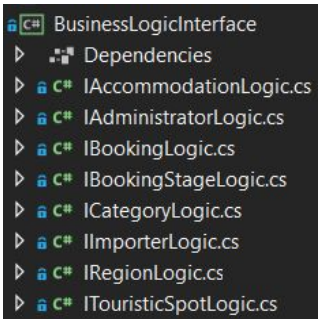


DataAccess

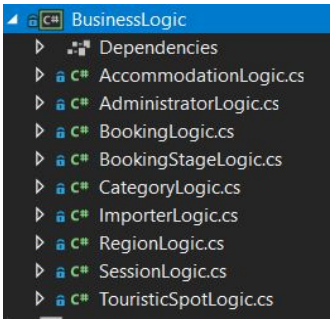




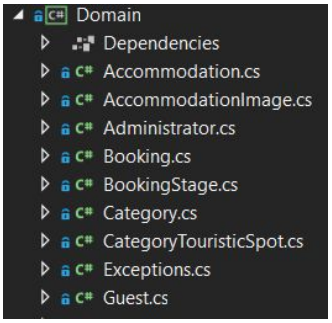
BusinessLogicInterface

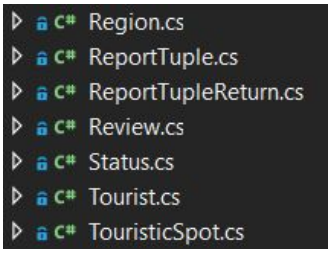


BusinessLogic

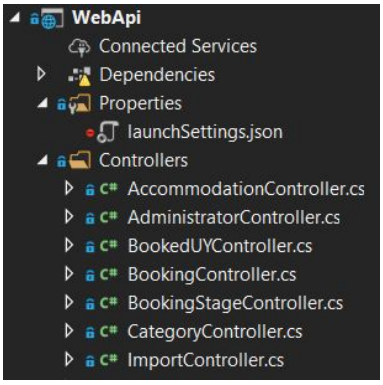


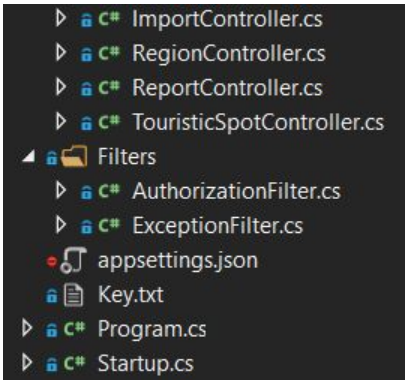
Domain



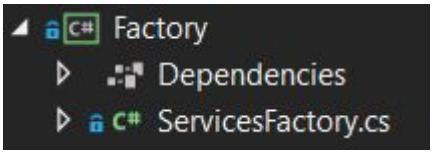


WebApi

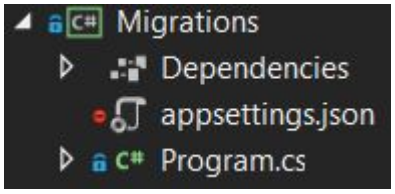




Factory

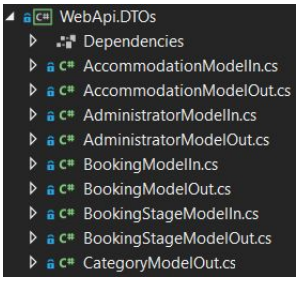


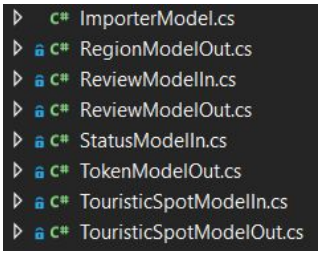
Migrations



SessionInterface

WebApi.DTO





Informe de cobertura

Se logró una cobertura total de un 96.18% , la cual consideramos aceptable ya que la mayoría del código está probado. Esto es importante ya que reducimos la posibilidad de que el cliente/usuario sea quien se tope con un bug o error no esperado. en ciertos paquetes se logró una menor cobertura, en algunos ya desde el obligatorio anterior, como fue el caso de WebApi, que de todas formas mejoró en parte su cobertura. Las clases que tienen una menor cobertura se debe a que por falta de tiempo no se lograron testear al 100%.

facun_LAPTOP-9TPOE6JA 2020...	328	3.82%	8254	96.18%
businesslogic.dll	39	8.11%	442	91.89%
businesslogic.tests.dll	33	1.37%	2380	98.63%
dataaccess.dll	21	1.73%	1194	98.27%
dataaccess.tests.dll	2	0.11%	1830	99.89%
domain.dll	44	10.05%	394	89.95%
domaintests.dll	72	9.41%	693	90.59%
importinterface.dll	16	25.81%	46	74.19%
webapi.dll	24	13.95%	148	86.05%
webapi.dtos.dll	77	20.59%	297	79.41%
webapi.tests.dll	0	0.00%	830	100.00%

Evidencia del diseño y especificación de la API

Discusión de los criterios seguidos para asegurar que la API cumple con los criterios REST

- Dentro de las URIs:
    - Se utilizó sustantivos en minúscula y en plural, como fue recomendado en clase. Como ejemplo de esto tenemos: administrators, regions o touristicspots. Consideramos que son intuitivos y simples para que sea más fácil de utilizar, ya que solo mediante la URI se logra comprender lo que hace el endpoint.
    - Se utilizó nombres concretos antes que abstractos como en el caso del enpoint de administrators donde se podía haber puesto persons en vez de administrators.
    - En cuanto a los niveles solo existe un endpoint que llega a los dos niveles maximos, que es touristicspots que tiene regionId y categories. Se utiliza ? en ciertos endpoints, como es el caso de touristicspots?RegionId=3&Categories=1. Y por último no existen endpoints que tengan verbos en la URI
- Verbos HTTP:
    - Como fue enseñado en clase, utilizamos sólo los verbos GET POST PUT DELETE con únicamente su funcionalidad. No se utilizó ningún Verbo HTTP extra a estos como PATCH o TRACE. En el caso de los errores se utilizaron los HTTP status code y también nos ocupamos de que haya una respuesta entendible para la persona que recibe ese código de estado
- Respuestas parciales:
    - Se utilizaron DTOs (Data Transfer Object) donde se implementó que debería ser expuesto al usuario, en caso de los DTOs de salida. En el caso de los DTOs de entrada se determinó que información debe aportar el usuario, tomando en cuenta que no tiene todo la información de la entidad de negocio.
- Stateless:
    - Consideramos que nuestra API es stateless ya que, como lo dice la definición de stateless, en cada request se necesita toda la información para que el servidor nos comprenda. Uno de los principales ejemplos está en el login, ya que el sistema nos proporciona un token, pero si no se envía el token en la próxima request, él mismo nos pedirá autenticación, es necesario que nosotros se lo enviemos al sistema.
- Separado por capas:
    - Otro criterio REST es lograr una API separada en capas, en nuestro caso tenemos una capa de WebApi, que es quien se encarga de recibir los requests y enviar la data a una nueva capa, la capa lógica que contiene la lógica de negocios. Para luego enviar data a la capa de acceso a datos que es quien hace contacto con la base de datos y se encarga tanto de enviar la información a la base de datos, como de extraerla y comenzar el proceso en dirección contraria, es decir hacia la capa lógica.

Descripción del mecanismo de autenticación de requests.

Para la autenticación decidimos utilizar el método de autenticación simple mediante tokens, los cuales son generados por JWT (JSON Web Token), de esta manera podemos asegurarnos que los tokens serán hasheado de manera correcta y segura. Para la autentificación utilizamos un IAutenticationFilter que se encarga de llamar a la clase SessionLogic la cual tiene la responsabilidad tanto de crear el token en el login de administrators como de validarlo al momento de realizar una request que requiera autorización. A la hora de hacer login se crea un token utilizando como clave el email del administrador y se lo devuelve al usuario, el cual deberá incluirlo en el header de las consultas que requiera autorización.

Descripción general de códigos de error

200 - OK
Lo encontramos en por ejemplo el login de un administrador con email y contraseña correcta.

Objetivo: Informar al usuario que la solicitud ha tenido éxito.

400 - Bad Request
Lo encontramos por ejemplo al intentar crear un administrador sin contraseña.
Objetivo: Informar al usuario que el servidor no pudo interpretar la solicitud. Asociado a sintaxis.

401 - Unauthorized
Lo encontramos por ejemplo al intentar crear un administrador con email y contraseña válidas pero sin token.
Objetivo: Informar al usuario que debe autentificarse para que la solicitud sea exitosa.

403 - Forbidden
Lo encontramos por ejemplo al intentar crear un administrador con email y contraseña válidas pero con un token invalido para esa consulta.
Objetivo: Informar al usuario que a pesar de haberse autenticado, el mismo no posee los permisos necesarios para realizar esa solicitud.

404 - Not Found
Lo encontramos por ejemplo al intentar crear un administrador con email o contraseña no válida.
Objetivo: Informar al usuario que el servidor no pudo encontrar el contenido el cual solicitó.

Descripción de los recursos de la API

Administrators

URI: /api/administrators

```
{
   "email": string,
   "password": string
}
```

Get
Chequea si existe un administrador con ese email y esa contraseña.
URI: /login?email=admin&password=123

email	Identificador único de un administrador	Debe existir
password	Se chequea en caso de coincidir el email necesario	Debe ser igual, en caso de existir el email

200 - token (devuelve un token)
404 - Administrator Not Found
Post
Crea un nuevo administrador con ese email y esa contraseña. El usuario debe estar autenticado para realizar esta consulta.
Header: Authorization
Body:

```
{
   "email": "test@test.com",
   "password": "contrasena"
```

}

email	Identificador único de un administrador	No debe existir en la BD
password	Se asocia a el email y se va a solicitar para hacer Login	

200 - Administrator created successfully
401 - No token provided
400 - The Admin Email cannot be null
400 - The Admin Password cannot be null
400 - The Administrator Already Exists

Delete
Borra un administrador del sistema
URI: /2
Header: Authorization

AdministratorId	Debe existir un administrador con esa id	Debe ser un número

200 -
404 - Administrator not found
401 - No token provided
400 - { "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "traceId": "|8eed864a-4902b34a6f04b64f.",
  "errors": {
    "id": [
      "The value 'a' is not valid."]}}

Regions

URI: /api/regions

```
{
   "id":int,
   "Name":string
}
```

Get

Trae todas las regiones

URI: -

200 -

```
   [{
      "id": 1,
      "name": "Region Oeste"
   }]
```

Touristic Spots

URI: /api/touristicspots

```
{
   "Name":string,
   "RegionId": int,
   "Description": string,
   "Image": string,
   "Categories": [int]
}
```

Post

Crea un nuevo punto turístico

Header: Authorization

Body: {

```
   "Name":"La Paloma",
   "RegionId": 3,
   "Description": "Pueblo de hermosas playas",
   "Image": "FOTO",
   "Categories": [1,2]
}
```

Name	Identifica a un TouristicSpot	Debe ser un string, requerido
RegionId	Identifica a la región asociada al TouristicSpot	Debe ser un int y debe existir un región con ese id, requerido
Description	Describe al TouristicSpot	Debe ser un string, required
Image	Imagen del TouristicSpot	Debe ser un string

Response:

200 - {

```
   “id”:3,
   "name":"La Paloma",
   "description": "Pueblo de hermosas playas",
   "Image": "FOTO"
}
```

401 - No token provided

400 - The Spot Already Exists

400 - The Spot Categories cannot be null

400 - The Spot Name cannot be null

400 - The Spot Region cannot be null

400 - The Spot Description cannot be null

400 - {

```
   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
   "title": "One or more validation errors occurred.",
   "status": 400,
   "traceId": "|b5aea160-4b77b2f80b3aa9c0.",
```

```
   "errors": {
      "$.Name": [
           "The JSON value could not be converted to System.String. Path: $.Name | LineNumber: 1 |
BytePositionInLine: 12."
      ]
   }
}
```


Get

Retorna puntos turísticos, dependiendo de si cumplen o no con las características solicitadas. De no haber características, devuelve todos.

RegionId	Identifica a qué región deben estar asociadas los puntos turísticos a traer	Debe ser un int, opcional, va en la query
Categories	Identifica a qué categoria deben pertenecer los puntos turísticos a traer	Debe ser un int, opcional, va en el query


```
200 - [{
      "id": 3,
      "name": "La Paloma",
      "description": "Pueblo de hermosas playas",
      "image": "FOTO"
   }]
200 - []
400- {
   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
   "title": "One or more validation errors occurred.",
   "status": 400,
   "traceId": "|183746ac-4338205a586477e7.",
   "errors": {
      "categories": [
         "The value '' is invalid."
      ]
   }
}
```

Accommodations

URI: /api/accommodations

```
 {
    "id": int,
    "name": string,
    "address": string,
    "contactNumber": string,
    "information": string,
    "price": int,
    "images": [string]
}
```

Post

Crea una nueva accommodation

Header: Authorization

Body: {

```
    "Name":"Hilton",
    "Address":"Mar Sereno 1045",
    "Price":48.2,
    "Contact":"+598",
    "Information":"Un Hotel muy lindo",
    "SpotId":1,
    "Images":["fotoafuera","fotohabitacion","fotojardin"]
}
```

Name	Identificador de una accommodation, nombre de la misma	Debe ser un string, requerida
Address	Dirección de la accommodation	Debe ser un string, requerida
Price	Precio por noche de la accommodation	Double, requerido
Contact	Contacto de la accommodation	Debe ser un string, requerido
Information	Breve informacion de la accommodation	Debe ser un string, requerido
SpotId	Id del Spot al que se encuentra asociada la accommodation	Debe ser un int, requerido
Image	Imagenes del accommodation	Debe ser un string, requerido

Response:

```
200 - {
    "id": 4,
    "name": "Hilton",
    "address": "Mar Sereno 1045",
```

"contactNumber": "+598",
"information": "Un Hotel muy lindo",
"price": 48.2,
"images": [
"fotoafuera",
"fotohabitacion",
"fotojardin"
]}
401 - No token provided
400 - The Accommodation Already Exists
400 - Spot ID
400 - The Accommodation Name cannot be null
400 - The Accommodation Address cannot be null
400 - The Accommodation Spot cannot be null
400 - The Accommodation Images cannot be null
400 - The Accommodation Contact cannot be null
400 - The Accommodation Info cannot be null
400 - {
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
"title": "One or more validation errors occurred.",
"status": 400,
"traceId": "|183746be-4338205a586477e7.",
"errors": {
"$.Name": [
"The JSON value could not be converted to System.String. Path: $.Name | LineNumber: 1 | BytePositionInLine: 12."
]}}
400 - {
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
"title": "One or more validation errors occurred.",
"status": 400,
"traceId": "|183746c0-4338205a586477e7.",
"errors": {
"$.Price": [
"The JSON value could not be converted to System.Double. Path: $.Price | LineNumber: 3 | BytePositionInLine: 16."
]}}
400 - {
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
"title": "One or more validation errors occurred.",
"status": 400,
"traceId": "|183746c2-4338205a586477e7.",
"errors": {
"$.Price": [
"The JSON value could not be converted to System.Double. Path: $.Price | LineNumber: 3 | BytePositionInLine: 15."
]}}

Get
Retorna accommodations que cumplan con las características solicitadas.

Id	Identifica a una accommodation	Debe ser un int, opcional, va en la query. Si hay Id no hay SpotId URI: /1
SpotId	Identifica a qué TouristicSpot deben pertenecer las accommodations a traer	Debe ser un int, opcional, va en el query. Si hay SpotId no hay Id URI: /spot/1

200 - {
"id": 3,
"name": "La Paloma",
"description": "Pueblo de hermosas playas",
"image": "FOTO"
}
200 - [{
"id": 3,
"name": "La Paloma",
"description": "Pueblo de hermosas playas",
"image": "FOTO"
}]
200 - []
400- {"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
"title": "One or more validation errors occurred.",
"status": 400,
"traceId": "|8eed8660-4902b34a6f04b64f.",
"errors": {
"spot": [
"The value 'f' is not valid."]}}

Put
Actualiza la capacidad de una accommodation
Header: Authorization
Body: true

Id	Identifica a que accommodation se le quiere cambiar la capacidad	Debe ser un int, requerido, va en query URI: /Id

Response:
200 - Accommodation updated
404 - Accommodation Not Found
415 - {
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.13",

```
    "title": "Unsupported Media Type",
    "status": 415,
    "traceId": "|8eed8668-4902b34a6f04b64f."
}
```


Delete
Elimina un accommodation de la base de datos
Header: Authorization

Id	Identifica que accommodation se quiere borrar	Debe ser un int, requerido, va en la query URI: /Id

Response:
404 - Accommodation Not Found
401 - No token provided

```
200 - {
    "id": 7,
    "spot": {
        "id": 1,
        "name": "La Paloma",
        "region": null,
        "regionId": 3,
        "description": "Pueblo de hermosas playas",
        "categories": null,
        "accommodations": [],
        "image": "FOTO"
    },
    "spotId": 1,
    "name": "Nirvana",
    "address": "Capicua 1045",
    "contactNumber": "+598",
    "information": "Gran Hotel",
    "pricePerNight": 50.3,
    "full": false,
    "bookings": null,
    "images": [
        {
            "id": 13,
            "image": "fotoafuera",
            "accommodationId": 7,
            "accommodation": null
        },
        {
            "id": 14,
            "image": "fotohabitacion",
            "accommodationId": 7,
```

```
            "accommodation": null
        },
        {
            "id": 15,
            "image": "fotojardin",
            "accommodationId": 7,
            "accommodation": null
}]}
```

Bookings

URI: /api/bookings

```
 {
    "accommodationId": int,
    "checkIn": string,
    "CheckOut": string,
    "GuestEmail": string,
    "GuestName": string,
    "GuestLastName": string,
    "Guests": [
        {
            "Amount": int,
            "Multiplier": double
}]}
```

Post

Crea una nueva booking en el sistema

Body: {

```
    "accommodationId": 1,
    "checkIn": "2020-12-12",
    "CheckOut": "2020-12-24",
    "GuestEmail": "test@test.com",
    "GuestName": "test",
    "GuestLastName": "lastTest",
    "Guests": [
        {
            "Amount": 2,
            "Multiplier": 1
}]}
```

accommodationId	Identifica a que accommodation está asociada la booking	Debe ser un int, requerido
checkIn	Identifica en qué fecha se ingresa al hospedaje	Debe ser un string, requerido
checkOut	Identifica en qué fecha se retira hospedaje	Debe ser un string, requerido
GuestEmail	Identifica el email del HeadGuest	Debe ser un string, requerido
GuestName	Identifica el Name del HeadGuest	Debe ser un string, requerido
GuestLastName	Identifica el LastName del HeadGuest	Debe ser un string, requerido

Guests	Lista de Guests	Debe ser un <int,double>, requerido

Response:

200 - {
"id": 4,
"accommodationId": 1,
"accommodationName": null,
"accommodationAddress": null,
"accommodationContact": null,
"checkIn": "2020-12-12T00:00:00",
"checkOut": "2020-12-24T00:00:00",
"price": 6,
"guestEmail": "test@test.com"
}
400 - The Booking Guests cannot be null
400 - The Tourist Last Name cannot be null
400 - The Tourist Name cannot be null
400 - The Tourist Email cannot be null
400 - The CheckIn entered is invalid
400 - The CheckOut entered is invalid
400 - The Booking Accommodation cannot be null

Get

Retorna bookings que cumplan con las características solicitadas. En caso de no haber características trae todas las bookings.

Id	Identifica que booking debe traer	Debe ser un int, es opcional, va en la quey URI: /1.

Response:

200 - {
"id": 1,
"accommodationId": 1,
"accommodationName": "Hilton",
"accommodationAddress": "Mar Sereno 1045",
"accommodationContact": "+598",
"checkIn": "2020-12-12T00:00:00",
"checkOut": "2020-12-24T00:00:00",
"price": 6,
"guestEmail": "test@test.com"
}
404 - Booking Not Found
200 - [
{

```
    "id": 1,
    "accommodationId": 1,
    "accommodationName": "Hilton",
    "accommodationAddress": "Mar Sereno 1045",
    "accommodationContact": "+598",
    "checkIn": "2020-12-12T00:00:00",
    "checkOut": "2020-12-24T00:00:00",
    "price": 6,
    "guestEmail": "test@test.com"
  },
  {
    "id": 4,
    "accommodationId": 1,
    "accommodationName": "Hilton",
    "accommodationAddress": "Mar Sereno 1045",
    "accommodationContact": "+598",
    "checkIn": "2020-12-12T00:00:00",
    "checkOut": "2020-12-24T00:00:00",
    "price": 6,
    "guestEmail": "test@test.com"
  }
]
```

BookingStage

URI: /api/bookingstages

```
 {
   "BookingId": int,
   "Description": string,
   "Status": string,
   "AdminId": int
}
```

Post

Crea un bookingStage asociado a una booking

BookingId	Identifica a qué booking está relacionado	Debe ser un int, debe existir esa booking,required

404 - BookingStage Booking Not Found

200 - {

```
   "description": "Se Pago",
   "status": "Accepted"
}
```

404 - Status Not Found

400 - The Booking Stage Description cannot be null

Get

Retorna el bookingStage actual de una booking en específico.

bookingId	Identifica de qué booking debe traer el actual bookingStage	Debe ser un int, debe existir esa booking, required

200 - {

```
   "description": "Se Pago",
   "status": "Accepted"
}
```

200 - The booking has not been changed by an Administrator

Reports

URI: api/reports
Header: Authorization
Get
Trae el reporte para ese punto turistico y dentro de esas fechas
URI: ?touristicSpotName=La Paloma&start=2020-12-12&end=2020-12-30

touristicSpotName	Identificador único de punto turistico. Indica sobre que punto turistico se debe realizar el reporte	Debe existir un punto turistico con ese nombre
start	Fecha de inicio del reporte	Debe ser una fecha, requerido
end	Fecha de fin del reporte	Debe ser una fecha, requerido

200 -

```
[
   {
      "accommodationName": "Hilton",
      "count": 2
   },
   {
      "accommodationName": "Non Hilton",
      "count": 1
   }
]
```

404 - Touristic Spot Not Found
404 - The report failed because there were no bookings between the dates selected in the Touristic Spot selected.

Importers

URI: /api/importers

Get

Retorna los tipos de importers que puede utilizar para importar archivos.

200 - {
  “JSON”,
  “XML”
}
400 - There's no valid implementation in the DLL folder

Get

Retorna los parámetros a completar para realizar la importación, de un respectivo importador.

importerName	Identifica de qué importador debe traer los parámetros	Debe ser un string, debe existir un importador con ese nombre

200 - {
  "name": "Path",
  "type": "string"
}
404 - Importer Not Found

Post

Levanta de un file un accommodation, con ayuda de un importer en específico

Body: {
  "name": "JSON",
  "parameters": [
    {
      "name": "Path",
      "value": "C:\\Users\\seraf\\Desktop\\Jd.json"
    }
  ]
}

name	Identifica el nombre del importador a utilizar para levantar el accommodation	Debe ser un string y debe de existir un importer con ese nombre
parameters	Identifica todos los parámetros solicitados por el sistema a rellenar para poder realizar la importación de manera correcta	Debe ser del tipo que se solicitó por el sistema

404 - Importer Not Found

400 - The file selected could not be parsed, please read the documentation to keep with the format.

200 - {
    "Name":"Hilton",
    "Address":"Mar Sereno 1045",
    "Price":48.2,
    "Contact":"+598",
    "Information":"Un Hotel muy lindo",
    "SpotId":1,
    "Images":["fotoafuera","fotohabitacion","fotojardin"]
}

Data del file, ejemplo JSON:

```
{
  "Name":"TestReflectionJson",
  "Address":"TestAddress",
  "ContactNumber":"TestContactNumber",
  "Images" :
  [{"Image":"a"}],

  "Description":"TestDescription",
  "Full":false,
  "Information":"TestInformation",
  "PricePerNight":1,
  "TouristicSpot":{
          "Name":"ABC",
          "RegionId":1,
          "Description":"a",
          "Categories":[1],
  }
}
```

Data del file, ejemplo XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<AccommodationParse>
   <Address>TestAddressTS</Address>
   <ContactNumber>TestContactNumberTS</ContactNumber>
   <Description>TestDescriptionTS</Description>
   <Full>false</Full>
   <Images>
                  <AccommodationImageParse>
                          <Image>a</Image>
                  </AccommodationImageParse>
                  <AccommodationImageParse>
                          <Image>b</Image>
                  </AccommodationImageParse>
   </Images>
```

```
  <Information>TestInformationTS</Information>
  <Name>TestReflectionXML</Name>
  <PricePerNight>1</PricePerNight>
  <TouristicSpot>
                <Categories>
                        <int>1</int>
                </Categories>
                <Description>a</Description>
                <Name>ABC</Name>
                <RegionId>1</RegionId>
                <Image>imageOfTS</Image>
  </TouristicSpot>
</AccommodationParse>
```