PRUEBA TÉCNICA

PROMOCIÓN DE HOTELES

ÍNDICE

Contenido

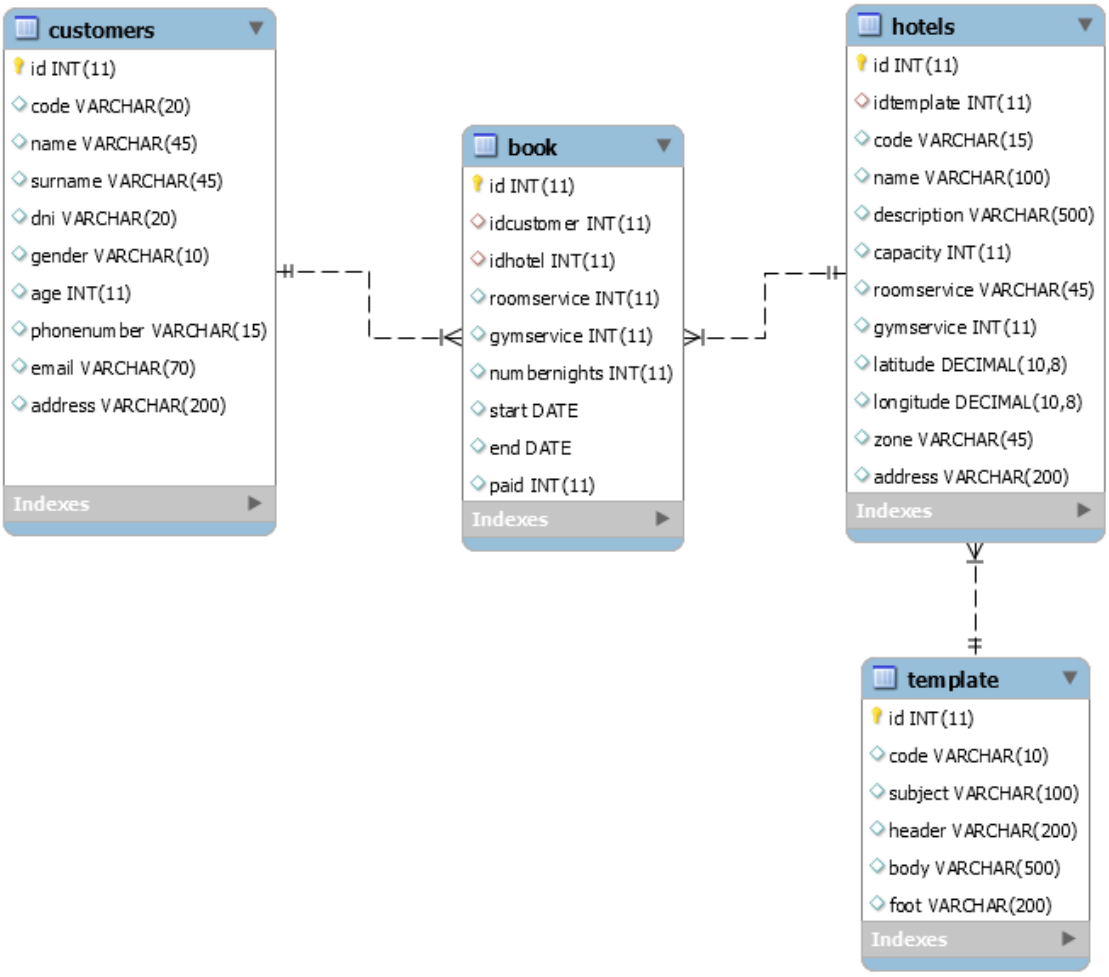
1. OBJETIVOS

Logitravel desea enviar a sus clientes un e-mail o sms, según los datos y preferencia de cada cliente, promocionando hoteles de la misma zona a los que ha contratado. Cada cliente ha podido reservar varios hoteles y cada tipo de hotel requiere una plantilla de mensaje diferente.

2. BASE DE DATOS

Para poder solucionar la prueba técnica he diseñado una base de datos que permitirá realizar las consultas necesarias para poder recomendar un hotel a un cliente en base a sus gustos, preferencia y ubicación. El motor de la base de datos se basa en MySQL.

2.1 MODELO RELACIONAL



2.2 EXPLICACIÓN DEL MODELO

A continuación, explicaré las tablas y sus relaciones para poder entender el modelo.

- **Customers:** almacena información de los clientes.
  - id: identificador único por cliente (primary key)
  - code: cadena alfanumérica de longitud 20
  - name: cadena de texto de longitud 45, adecuada para nombres compuestos
  - surname: cadena de texto de longitud 45, adecuada para poder almacenar clientes con 2 apellidos.
  - dni: cadena alfanumérica que representa el documente de identidad de un cliente. Puede ser un DNI, un NIE, pasaporte, CIF u otros
  - gender: cadena de texto que representa el sexo del cliente.
  - age: campo entero que representa la edad del cliente
  - phonenumber: dada su longitud de 15 caracteres podremos almacenar diferentes formatos de teléfono.
  - email: será importante para poder envía las recomendaciones de hoteles al cliente.
  - address: campo que puede servir para enviar documentos físicos en la residencia del cliente.

- **Hotels:** almacena información de los hoteles
  - id: identificador único por cliente **(primary key)**
  - idtemplate: este campo **(foreign key)** sirve para determinar que plantilla se tiene que enviar al cliente en caso de que este hotel sea uno de los recomendados para el **algoritmo de promoción** (más adelante hablaremos en detalle de él). Una observación importante es que según los requerimientos de la prueba técnica cada hotel debe tener una plantilla distinta al resto. Para solucionar este problema lo podemos hacer con un trigger(fuera del alcance del reto) en la base de datos o bien en el formulario de creación de hoteles (fuera del alcance del reto) validar que no se repitan. Conviene utilizar la segunda solución porque las operaciones en base de datos son más costosas computacionalmente hablando.
  - code: cadena alfanumérica de longitud 20
  - name: representa el nombre del hotel
  - description: almacena la descripción del hotel y se puede considerar como una carta de presentación de cara a los clientes.
  - capacity: representa el número máximo de clientes que el hotel puede alojar.
  - roomservice: indica si hotel dispone de este servicio.
  - gymservice: indica si el hotel dispone de este servicio

- latitude y longitude: estos campos servirán para calcular distancias entre hoteles.
- zone: representa la zona en donde se encuentra el hotel.
- address: dirección física del hotel.

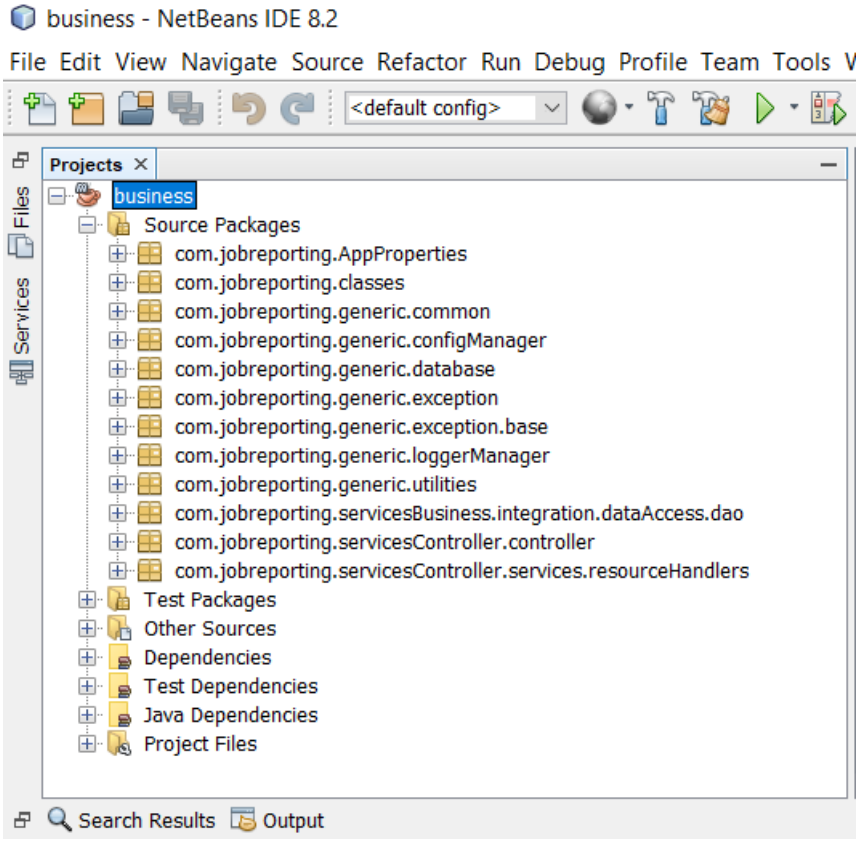
- **Book:** esta tabla almacena las reservas que ha hecho un cliente en un hotel.
  - Id: identificador único de la reserva **(primary key)**
  - code: representa el código de la reserva.
  - idcustomer: identificador del cliente **(foreign key)**
  - idhotel: identificador del hotel **(foreign key)**
  - roomservice: tipo entero que indica si el cliente ha reservado ese servicio o no.
  - gymservice: tipo entero que indica si el cliente ha reservado ese servicio o no.
  - numbernights: indica el número de noches que el cliente ha reservado.
  - start: indica la fecha de entrada.
  - end: indica la fecha de salida.
  - paid: indica si la reserva ha sido pagada o no.

- **Template:**
  - id: identificador único del template **(primary key)**
  - subject: tanto si se enviaría un email o un sms es importante indicar el asunto. Por ejemplo: “No pierda esta oportunidad de reservar…”
  - header: la cabecera del sms/email también es importante en la que podemos indicar el nombre del hotel, por ejemplo.
  - body: se pueden indicar los servicios que se promocionan
  - foot: podríamos incluir datos (texto e imágenes) corporativos e incluso algún tipo de información para atraer la atención del cliente por ejemplo “No se pierda esta oportunidad…oferta válida hasta el…”

3. ESTRUCTURA DEL PROYECTO

Una vez he comentado el modelo relacional ahora presentaré la estructura del proyecto. Para realizarlo utilicé un Web Service basado en Java que nos proporcionar un API RESTful para realizar operaciones con la base de datos. Como IDE de desarrollo he utilizado Netbeans. Utilicé un servidor Tomcat desplegado en un servidor Debian de Google Cloud Platform. Por este motivo se puede acceder desde una URL para consumir servicios. La base de datos MySQL se almacena en otro servidor Nginx desplegado en un servidor Debian también dentro de la misma red GCP.

A continuación, voy a enseñar como es la estructura de carpetas del proyecto para poder entender mejor su funcionamiento.

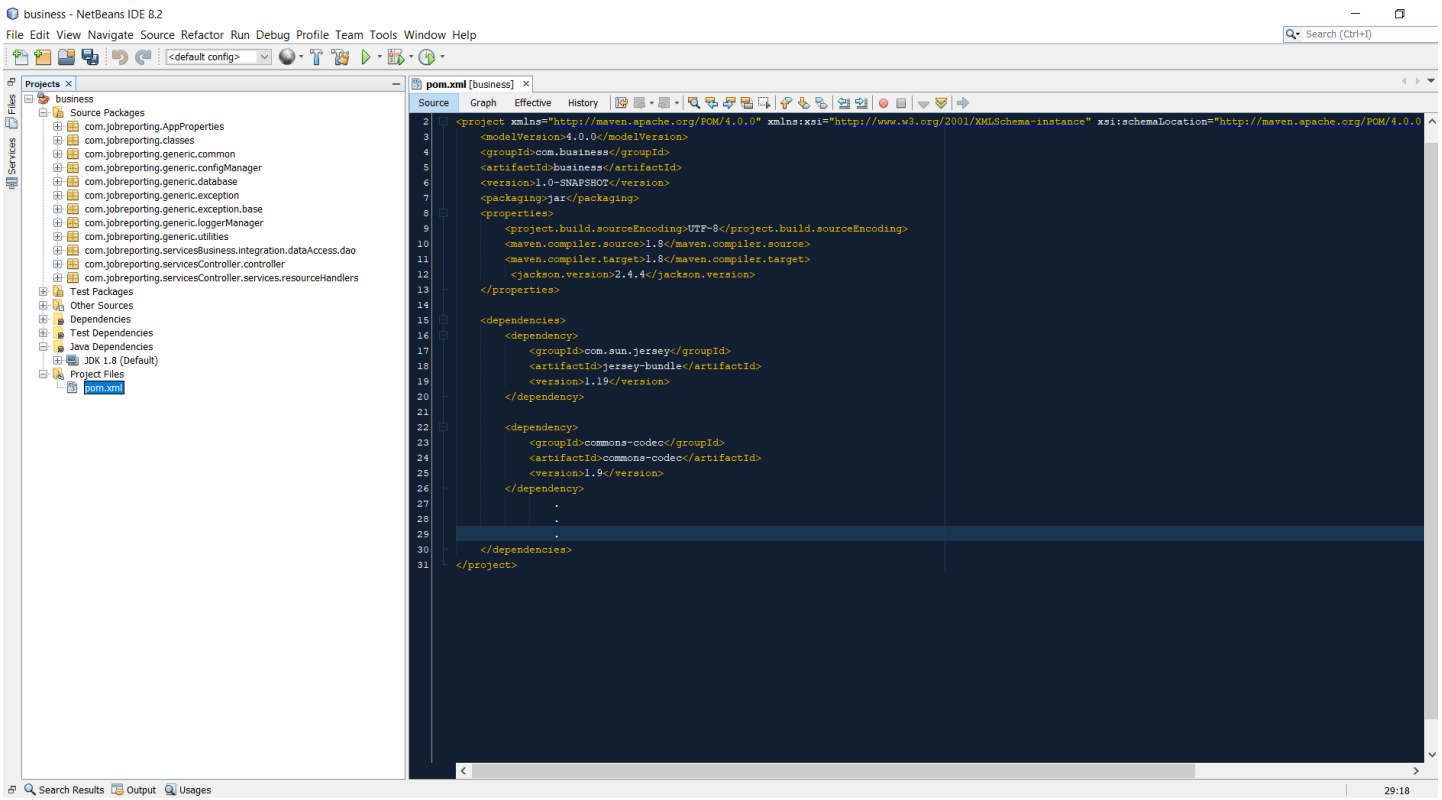


- AppProperties: contiene ficheros de configuración referente a la base de datos y log
- Clases: contiene las clases que serán utilizadas por los servicios del API
- Common: contiene clases en la que defino constantes
- ConfigManager: contiene clases con métodos para poder acceder a los ficheros de configuración.
- Database: clases con métodos para poder hacer operaciones con la base de datos. Utilizo un pool de conexiones para usuarios simultáneos ya que es más eficiente.

- Exception: clases que nos permiten lanzar excepciones.
- LoggerManager: proporciona todos los métodos para escribir en el log.
- Utilities: métodos genéricos que se pueden utilizar para tratar strings u otras funcionalidades.
- Dao: contiene clases con método que permiten hacer operaciones con la base de datos.
- Controller: contiene clases para iniciar el servlet, base de datos y registro de logs.
- ResourceHandlers: contiene clases que agrupan diferentes servicios que podrán ser consumidos a través del API. Los servicios los identificaremos mediante anotaciones.

3.1 GESTOR DE DEPENDENCIAS

Para incluir todas las librerías necesarias para el proyecto utilizo Maven como gestor de dependencias. Para ello, dispondremos de un archivo llamado pom.xml en el que declararemos todas las librerías que nos harán falta. A modo ilustrativo enseño la estructura del fichero.



3.2 SERVICIOS DEL API RESTful

Como hemos comentado en la estructura del proyecto, el paquete ResourceHandlers contiene diferentes clases que agrupa diferentes servicios. Para la prueba teórica simplemente he creado los servicios necesarios para consultar la base de datos. Las operaciones de creación, edición y eliminación de algún quedan fuera del alcance. Más adelante cuando hablemos sobre la escalabilidad del proyecto veremos que simple es añadir un servicio. Para probar los servicios he utilizado Postman.

- **Clase Customers:** contiene todos los servicios referentes a clientes.
  - **Servicio 1:**
    - **Nombre:** List
    - **Descripción:** devuelve el listado de clientes
    - **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Customers/List
    - **Method:** POST
    - **Consumes:** nothing
    - **Produces:** @Produces(MediaType.APPLICATION_JSON)

- **Clase Hotels:** contiene los servicios referentes a hoteles.
  - **Servicio 1:**
    - **Nombre:** ListRaw
    - **Descripción:** devuelve el listado de hoteles. La tabla hoteles contiene el foreign key de la tabla template. Por eso llamamos a este servicio LIsRaw ya que nos devuelve el listado con el idtemplate en vez de su código.
    - **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Hotels/ListRaw
    - **Method:** POST
    - **Consumes:** notihng
    - **Produces:** @Produces(MediaType.APPLICATION_JSON).
  - **Servicio 2:**
    - **Nombre:** ListFormated
    - **Descripción:** similar al anterior, pero en vez de traer el idtemplate trae el código del template.
    - **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Hotels/ListFormated
    - **Method: POST**
    - **Consumes:** nothing
    - **Produces:** @Produces(MediaType.APPLICATION_JSON).

- **Clase Booking:** contiene todos los servicios referentes a reservas.
    - Servicio 1:
        - **Nombre:** ListRaw
        - **Descripción:** devuelve el listado de reservas con las foreign keys.
        - **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Booking/ListRaw
        - **Method:** POST
        - **Consumes:** nothing
        - **Produces:** @Produces(MediaType.APPLICATION_JSON).
    - Servicio 2:
        - **Nombre:** ListFormated
        - **Descripción:** devuelve el listado de reservas remplazando las foreign keys por valores.
        - **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Booking/ListFormated
        - **Method:** POST
        - **Consumes:** nothing
        - **Produces:** @Produces(MediaType.APPLICATION_JSON).

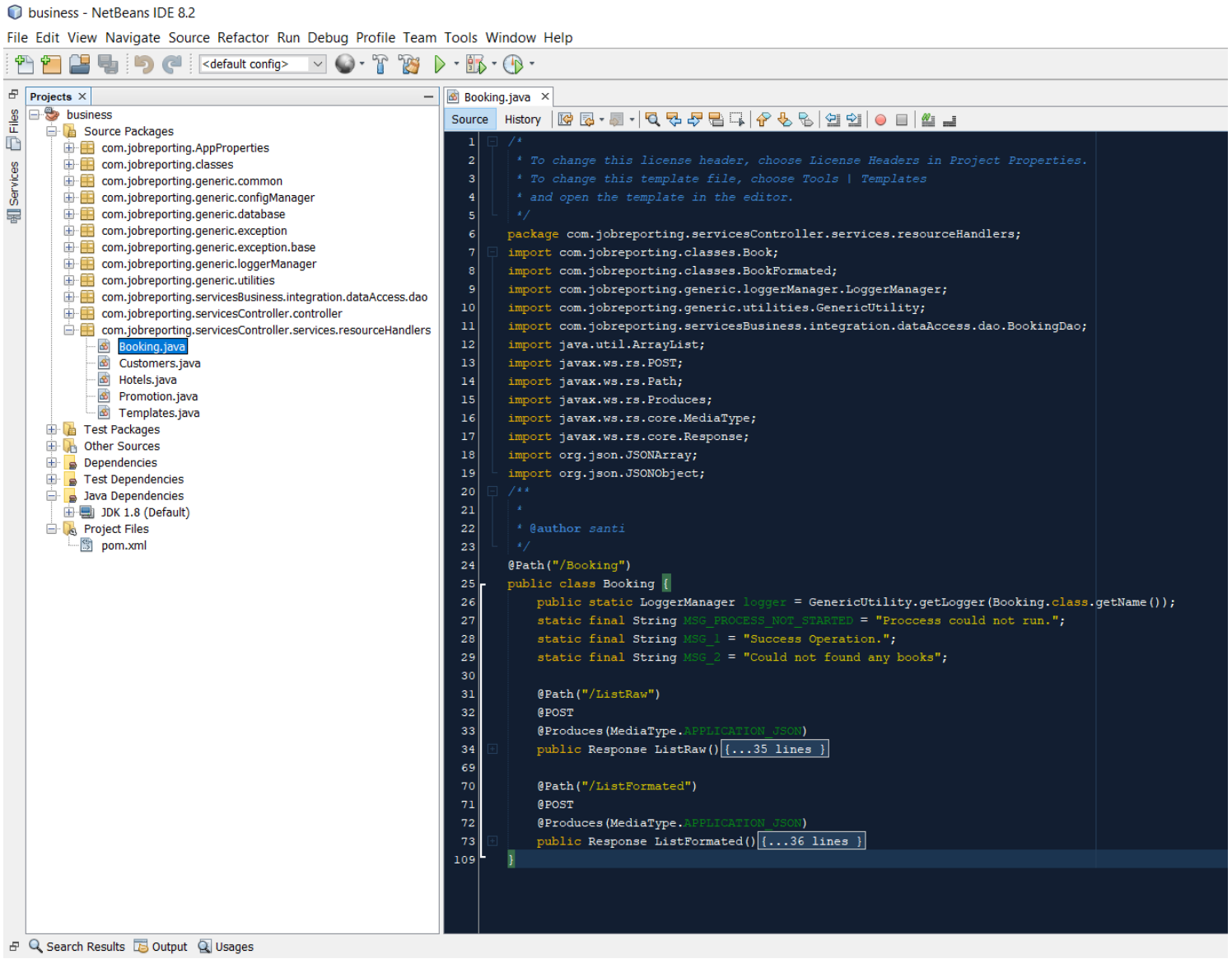
- **Clase Template:** contiene los servicios referentes a los template de mensaje de los hoteles.
    - Servicio 1:
        - **Nombre:** ListRaw
        - **Descripción:** devuelve todos los templates que hay en la base de datos.
        - **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Template/ListRaw
        - **Method:** POST
        - **Consumes:** nothing
        - **Produces:** @Produces(MediaType.APPLICATION_JSON).

- **Clase Promotion:** contiene un servicio que es el encargado de calcular que hoteles recomendar a un cliente.
    - Servicio 1:
        - **Nombre: SendPromotion**
        - **Descripción:** calcula que hoteles se pueden recomendar teniendo en cuenta las reservas que ha hecho un cliente. Se tendrá en cuenta las preferencias del cliente y las zonas donde ha reservado

- **URL:** http://tomcat.9sistemes.com/wslogitravel/services/Promotion/SendPromotion
- **Method:** POST
- **Consumes:** @Consumes(MediaType.APPLICATION_JSON). Recibe el nombre y el apellido del cliente.
- **Produces:** @Produces(MediaType.APPLICATION_JSON).

3.2 ESTRUCTURA DE UN SERVICIO

En la siguiente imagen vemos como se construyen los servicios y su simplicidad.



En la línea 24, establecemos la ruta para llegar a esta clase utilizando la anotación @Path. Luego dentro de la clase Booking definiremos diferentes servicios. Por ejemplo, para el servicio ListRaw especificamos 3 etiquetas:

- @Path: determina el nombre del servicio o ruta que se encadena con la línea 24
- Method: especificamos el tipo de método a utilizar (GET, POST…)
- @Consume: especificamos que tipo de parámetro espera recibir el servicio: JSON, XML, OCTECT-STREAM (para peticiones serializadas)
- @Produces: la misma idea que el anterior pero ahora se especifica que tipo de respuesta dará el servicio.

3.2 DETALLANDO EL SERVICIO PROMOTION/SENDPROMOTION

Es el servicio encargado de calcular que hoteles poder recomendar a un cliente en base a sus preferencias y zonas en las que ha reservado.

3.2.1 CONSIDERACIONES GENERALES

Recibe como parámetros el nombre del cliente y su apellido. Para resolver la prueba técnica y por temas de simplicidad emplearemos 3 parámetros: roomservice, gymservice y zone.

El uso de estos parámetros es simplemente ilustrativo para el caso, pero podríamos tener muchísimas preferencias lo cual ayudaría al algoritmo a hacer mejores recomendaciones. Describo un par de ejemplo para poder demostrar esta idea.

Imaginemos que un cliente le gusta mucho ir a un hotel que tenga piscina porque detectamos que en todas sus reservas lo ha hecho con hoteles que tienen piscinas. La mayoría de piscinas cuentan con barra de bebidas en la que los clientes pueden pedir un coctel o refresco y disfrutar de la piscina. También sabemos que en el área de la piscina hay hamacas o tumbonas disponibles para el cliente. A es cliente le gusta reservar una tumbona en una posición concreta referente a la piscina (primera línea, segunda, a la sombra, con mayor descuento en tragos según posición…) para toda su estancia. ¿Podemos hacer un matching con hoteles que permitan este tipo de servicios?

Otro ejemplo, imaginemos que el cliente es un apasionado del spa. Al cliente le gusta hacer 2 sesiones de sauna por la mañana y una de fitness por la noche. ¿Podemos hacer un matching con hoteles que permitan este tipo de servicios en el rango de horas que cliente desea? ¿Y si combinamos los 2 anteriores? ¿O muchos tipos de servicios más?

A medida que disponemos de más información de preferencias del cliente el sistema de recomendación es más complejo lo que requiere mayor coste computacional, pero a la vez podemos hacer mejores recomendaciones. Si el volumen de información de preferencias es muy grande podríamos acudir al Maching Learning utilizado algoritmos de predicción de modelos.

También, uno de los requerimientos de la prueba teórica es utilizar el área donde ha reservado anteriormente. Para ello, utilizaremos el campo zona que especificamos en la base de datos, que consideramos Calviá y Palma.

Realmente utilizar solo el campo zona puede resultar un poco inexacto ya que dentro de una zona puede haber bastante distancia entre hoteles. Podríamos calcular la distancia en calles entre hoteles o en km utilizando las coordenadas latitud y longitud. Es decir, podríamos recomendar hoteles en el radio de 2KM. Para esta prueba teórica utilizaremos el campo zona tal cual por simplicidad.

Por último, para este caso he decidido que también se pueda recomendar el mismo hotel que ha reservado, pero esto es una cuestión de marketing de la empresa podido excluirlo fácilmente.

3.2.2 ALGORITMO DE RECOMENDACIÓN

El algoritmo nos devolverá un listado con las recomendaciones de cada hotel por cada reserva. Una recomendación es un mensaje que se enviará al cliente.

Pasos:

- **1.** Validamos que el cliente pasado en el query exista en la base de datos
- **2.** Comprobamos que el cliente ha hecho alguna reserva
- **3**. Si el cliente ha hecho alguna reserva entonces
    - Obtenemos el listado de reserva
    - Para cada reserva hacer
        - Obtenemos preferencias (roomservice, gymservice y zona)
        - Buscamos todos los hoteles que coincidan con las preferencias y nos traemos el contenido de la plantilla del mensaje para cada hotel añadiendo estos datos al response.
    - Fin para
- **4.** Fin si
- **5.** Devolver response

4. ESCALABILIDAD E INTEROPERABILIDAD

La estructura del proyecto nos permite añadir fácilmente servicios o grupos de servicios. Para añadir un nuevo grupo de servicios creamos una nueva clase en el paquete ResourseHandlers y para añadir nuevos servicios los anidamos dentro de la clase correspondiente.

Desde el punto de vista de la interoperabilidad, utilizamos un API REST basada en Java porque lo que la podríamos desplegar en cualquier servidor ya sea físico o virtualizado.

5. CONCLUSIONES

Hemos visto como mediante el desarrollo de una API REST pudimos crear un algoritmo capaz de recomendar hoteles a un cliente de una manera sencilla y escalable. Cierto es que hemos utilizado pocas preferencias, pero me ha servido para demostrar la potencia de las API REST.

Este proyecto puede crecer mucho añadiendo nuevos servicios incluso consumir servicios de otras APIs.

Para finalizar, comentar que me ha gustado el reto y espero estar a la altura de vuestras expectativas para el puesto de Senior Developer, creo que puedo aportar mucho a la empresa.