La Cadena de Hoteles “Buenas Noches” requiere una solución Backend basada en MS Rest, todo esto para su nueva página de internet que les ayudará al control de sus hoteles, habitaciones, reservaciones y huéspedes.

Descripción:

Hoteles

La cadena de hoteles Buenas Noches por ahora cuenta con 2 hoteles pero se requiere que la solución acepte agregar nuevos hoteles con o sin habitaciones.

Habitaciones

Se requiere poder agregar, actualizar o borrar habitaciones de todos los hoteles que pertenecen a la cadena hotelera Buenas Noches. Así mismo se necesita tener un control de las habitaciones que están desocupadas u ocupadas.

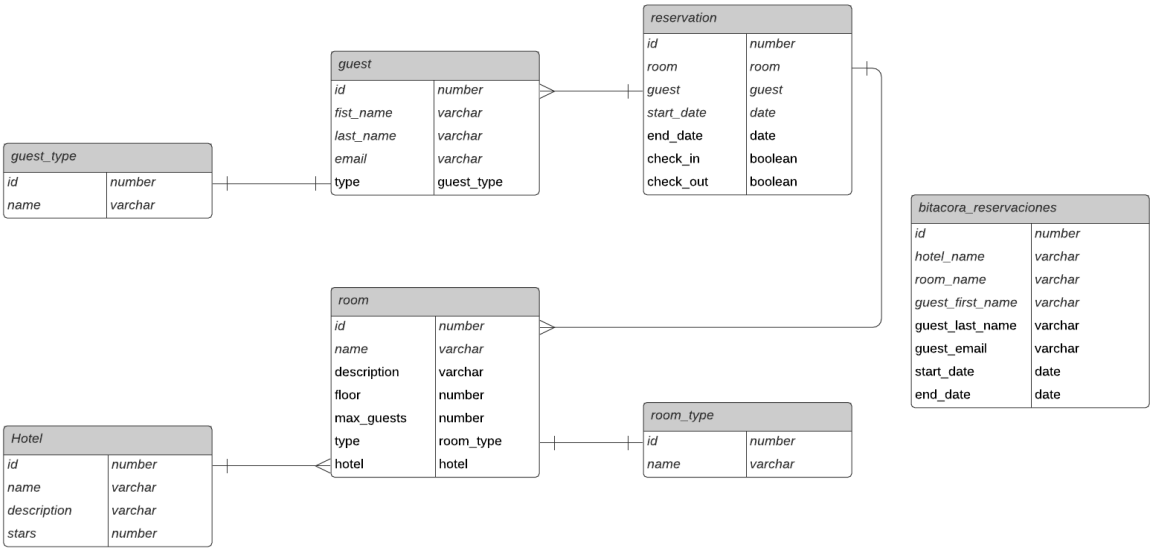
Reservas

Es importante tener un control de las reservaciones que se tiene de las habitaciones y de su check in y check out de los huéspedes.

Huéspedes

Se requiere tener un registro de los huéspedes de cada hotel y así mismo se necesita una bitácora que nos permita definir si un huésped merece ser un miembro VIP de nuestra cadena hotelera (después de 5 reservaciones con nosotros, el huésped se considera VIP) Los clientes VIP tiene acceso a las habitaciones tipo Suite.

Este es el modelo de la DB de nuestro cliente.



Hoteles buenas noches utiliza las siguientes herramientas, frameworks y buenas prácticas, las cuales se tienes que tomar en cuenta para este proyecto ya que si no lo hacemos el cliente regresará el desarrollo y nos tomará más tiempo hacer las correcciones:

- Git / github
- Spring Boot
- Spring Data
- MVC
- SOLID
- Swagger
- Gradle
- Lambdas y Streams en la medida de lo posible
- Unit Test / Mockito

Día 1:

Objetivo:

El cliente requiere que montes un ambiente en tu máquina local ya que él no posee un ambiente de desarrollo, para eso se necesita:

- Crear la DB en postgres
- Crear un proyecto nuevo Spring Boot, Gradle.
- Montar el repositorio en github

Url's de apoyo:

- https://spring.io/guides/gs/spring-boot/
- https://start.spring.io
- https://github.com/git-guides/git-init

Entregable:

Tener en github listo el proyecto de Spring boot con gradle y creada la DB en Postgres.

Día 2:

Objetivo:

El cliente nos solicita agregar dos dependencias y como es la primera vez que usa Gradle ha visto que dentro de las dependencias hay diferentes formas de declararlas, como "runtimeOnly", nos ha solicitado a modo de comentario una explicación de las formas de declarar dependencia en Gradle y así su equipo interno le sea más fácil entender la diferencia:

- Agregar la dependencia de lombok más reciente.
- Agregar la dependencia de apache commons más reciente.
- Será obligatorio usar las validaciones de apache commons y la funcionalidad de lombok.
- Tener la documentación en forma de comentario dentro del archivo Gradle, ¿Cuales son las formas de declaración de dependencias en Gradle?

Url's de apoyo:

- https://docs.gradle.org/current/userguide/declaring_dependencies.html
- https://www.arquitecturajava.com/spring-stereotypes/

Entregable:

El código debe estar actualizado en el github y tener la documentación en forma de comentario dentro del archivo Gradle, sobre ¿Cuáles son las formas de declaración de dependencias en Gradle?

Día 3:

Objetivo:

En este día el usuario requiere que la conexión con la DB ya esté lista, así como todos los componentes necesarios para su uso.

- Crear las entidades correspondientes, conforme a las tablas de la DB.
- Crear los Repository y/o DAO necesarios (Si es que son necesarios)
- Crear la configuración necesaria para la conexión a la DB

Url's de apoyo:

- https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

Entregable:

Tener listo los Repository, Entity y DAO's necesarios. Dentro del proyecto de Spring boot, garantizando la conectividad con la DB de postgres.

Día 4:

Objetivo:

Para este día se requiere tener los endpoints necesarios para la parte de administración interna de nuestro cliente sobre hoteles, como son:

- Añadir un nuevo hotel con o sin habitaciones.
- Borrar un hotel y sus habitaciones relacionadas
- Actualizar un hotel
- Lista de hoteles
- Utilizar anotaciones y estereotipos necesarios para la inyección de dependencias.

NOTA: El cliente ha solicitado que en la medida de lo posible se manejen streams a la hora de manejar listas y el cliente ha escuchado que hay diferentes tipos de formas de manejar arreglos y listas dentro de java y nos a solicitado que se deje una pequeño comentario para cada que se ocupe una lista justificando el por que ese tipo de lista es la mejor manera de atacar el problema y así su área de desarrollo pueda entender la diferencia entre listas usadas. Así mismo nos pide usar lambdas en todo momento que sea posible.

NOTA: El cliente nos pide que usemos anotaciones y estereotipos para el manejo del flujo de los microservicios. Esto se tiene que aplicar a todo el desarrollo.

Url's de apoyo:

- https://spring.io/guides/tutorials/rest/
- https://www.baeldung.com/java-8-streams

- http://tutorials.jenkov.com/java/lambda-expressions.html
- http://tutorials.jenkov.com/java-collections/streams.html
- https://www.baeldung.com/java-8-lambda-expressions-tips

Entregable:

El cliente nos pide que el código de los endpoints de hoteles ya debe estar arriba en el repositorio de github y aunado a eso nos solicita que le demos una demo consumiendo los endpoints por medio de Postman, de tal manera que pueda ver que es posible agregar, actualizar, borrar y consultar hoteles.

Día 5:

Objetivo:

Para este día se requiere tener los endpoints necesarios para la parte de administración interna de nuestro cliente sobre habitaciones, como son:

- Añadir una habitación a un hotel
- Actualizar una habitación (Tipo, piso, descripción)
- Borrar una habitación de un hotel
- Lista de habitaciones de un hotel

NOTA: El cliente ha solicitado que en la medida de lo posible se manejen streams a la hora de manejar listas y el cliente ha escuchado que hay diferentes tipos de formas de manejar arreglos y listas dentro de java y nos a solicitado que se deje una pequeño comentario para cada que se ocupe una lista justificando el por que ese tipo de lista es la mejor manera de atacar el problema y así su área de desarrollo pueda entender la diferencia entre listas usadas. Así mismo nos pide usar lambdas en todo momento que sea posible.

Url's de apoyo:

- https://spring.io/guides/tutorials/rest/
- https://www.baeldung.com/java-8-streams
- http://tutorials.jenkov.com/java/lambda-expressions.html
- http://tutorials.jenkov.com/java-collections/streams.html
- https://www.baeldung.com/java-8-lambda-expressions-tips

Entregable:

El cliente nos pide que el código de los endpoints de habitaciones ya debe estar arriba en el repositorio de github y aunado a eso nos solicita que le demos una

demo consumiendo los endpoints por medio de Postman, de tal manera que pueda ver que es posible agregar, actualizar, borrar y consultar habitaciones.

Día 6:

Objetivo:

En este punto el cliente necesita que todos los endpoint necesarios para crear reservaciones (reutilizando código que ya tengamos) esten listos y agregar nuevas funcionalidades:

Cuando un cliente necesita hacer una reservación

- Hoteles “Buenas noches” quiere poder determinar si su huéspedes son clientes VIP o no, en ese caso nos han pedido que nunca se borre el registro que está en la tabla de guest y que cuando un cliente ya haya alcanzado 5 reservaciones exitosas con ellos en cualquier hotel de su cadena, este cliente pase a ser un cliente VIP, con acceso a las habitaciones VIP o Suites.
- Buscar el cliente si ya está registrado en nuestro sistema y si es VIP o no.
- Saber el listado de habitaciones disponibles de un hotel en específico, el cual debe de decirnos que tipo de habitación es, que piso, y cuantas personas maximo permite esa habitación.
- Registrar a cada uno de los huéspedes que no estén aún en nuestra DB y después hacer la reservación en la habitación que ha escogido, con fecha de inicio y fin de su estancia y a la hora de que el huésped llegue al hotel hacer check in en su reservación. **NOTA**: el cliente necesita que le ayudemos ya que se ha percatado que para los huéspedes es muy molesto registrar a sus hijos menores de edad y por consiguiente necesita que le demos una solución a este problema.
- Guardar la reserva en DB.
- A la hora de hacer check in, se necesita que la reservación se guarde en la bitácora para llevar un registro de los clientes que nos visitan
- A la hora de ver la lista de habitaciones de un hotel, nos aparezcan los nombres de cada huésped relacionado a la habitación.

NOTA: El cliente ha solicitado que en la medida de lo posible se manejen streams a la hora de manejar listas y el cliente ha escuchado que hay diferentes tipos de formas de manejar arreglos y listas dentro de java y nos a solicitado que se deje una pequeño comentario para cada que se ocupe una lista justificando el por que ese tipo de lista es la mejor manera de atacar el problema y así su área de desarrollo pueda entender la diferencia entre listas usadas. Así mismo nos pide usar lambdas en todo momento que sea posible.

Url’s de apoyo:

- https://spring.io/guides/tutorials/rest/
- https://www.baeldung.com/java-8-streams
- http://tutorials.jenkov.com/java/lambda-expressions.html
- http://tutorials.jenkov.com/java-collections/streams.html
- https://www.baeldung.com/java-8-lambda-expressions-tips

Entregable:

Tenemos que tener todo nuestro código ya en el repositorio de github y poder hacer un flujo completo de una nueva reservación por medio de Postman. Así mismo necesitamos mostrar la solución que implementamos para poder resolver las dos mejoras que el cliente nos ha solicitado. Asignar a un cliente si es VIP o no después de tener 5 reservaciones exitosas en cualquier hotel de la franquicia, Así mismo que solucion propusimos para no tener que registrar a niños menores de edad en una habitación pero al mismo tiempo si consultamos las habitaciones sepamos que son hijos de los adultos en otra habitación o que están en la misma habitación.

Día 7:

Objetivo:

En este punto el cliente necesita que todos los endpoint necesarios para cancelar reservaciones (reutilizando código que ya tengamos) estén listos.

Cuando un cliente necesita cancelar una reservación

- Buscar la habitación donde se encuentra un cliente por medio de Primer nombre, Primer apellido o email. Esto nos dará toda la información de la reserva, para saber si ya hizo check in o no, o si su fecha de inicio de la reserva ha iniciado o no.
- En caso de que la fecha de inicio de la reservación y que el huésped no haya hecho check in aun, será posible hacer la cancelación de la reservación, de lo contrario debería ser imposible cancelar la reservación.
- Cancelar la reservación y no guardarla en bitácora.

NOTA: El cliente ha solicitado que en la medida de lo posible se manejen streams a la hora de manejar listas y el cliente ha escuchado que hay diferentes tipos de formas de manejar arreglos y listas dentro de java y nos a solicitado que se deje una pequeño comentario para cada que se ocupe una lista justificando el por que ese tipo de lista es la mejor manera de atacar el problema y así su área de desarrollo pueda entender la diferencia entre listas usadas. Así mismo nos pide usar lambdas en todo momento que sea posible.

Url’s de apoyo:

- https://spring.io/guides/tutorials/rest/
- https://www.baeldung.com/java-8-streams
- http://tutorials.jenkov.com/java/lambda-expressions.html
- http://tutorials.jenkov.com/java-collections/streams.html
- https://www.baeldung.com/java-8-lambda-expressions-tips

Entregable:

Tenemos que tener todo nuestro código ya en el repositorio de github y poder hacer un flujo completo de una cancelación por medio de Postman.

Día 8:

Objetivo:

El cliente se ha percatado que debe mejorar el manejo de excepciones y necesita controlar todas las excepciones del código en su totalidad y personalizarlas.

- Se necesita controlar y personalizar todas las excepciones de una manera óptima.

Url's de apoyo:

- https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc
- https://howtodoinjava.com/spring-boot2/spring-rest-request-validation/

Entregable:

Todo el código debe estar actualizado en nuestro github y haber migrado todas las Excepciones de nuestra APP a este nuevo manejador de errores.

Día 9:

Objetivo:

Ha surgido una necesidad imprevista ya que nuestro cliente ha contactado con varios periódicos y revistas digitales para poder distribuir sus noticias y promociones a estos medios de comunicación de una manera óptima y desacoplada.

- Se necesita un endpoint donde recibirá una noticia y dicha noticia sea distribuida a todos los medios que estén suscritos a las notificaciones

Url's de apoyo:

- https://www.baeldung.com/java-observer-pattern

Entregable:

Que todo el código actualizado en tu github, así mismo presentar un demo de la funcionalidad del endpoint que hemos desarrollado al cliente con el patrón observer, el cual debe simular que tiene dos suscriptores y estos recibir la noticia que hemos mandado por medio del endpoint en dos diferentes archivos, simulando que son los canales de comunicación que recibirán dicha noticia o promoción.

Día 10:

Objetivo:

El cliente necesita calcular el tiempo que tarda cada endpoint en responder, para esto necesita que utilicemos programación orientada a aspectos para no repetir código en cada servicio REST.

- Implementar un aspect con la funcionalidad necesaria para calcular el tiempo que tarda cada endpoint en resolver y responder. Así como implementar este aspect en cada uno de los endpoints.

Url's de apoyo:

- https://dzone.com/articles/implementing-aop-with-spring-boot-and-aspectj
- https://docs.spring.io/spring-framework/docs/4.3.15.RELEASE/spring-framework-reference/html/aop.html

Entregable:

El código este mergeado en github y que se muestre en el log el tiempo que tarda cualquier endpoint de nuestra solución en ejecutarse por medio de aspectos.

Día 11:

Objetivo:

Al final el cliente nos ha pedido que realicemos Unit Test con Mockito de todos los Controller y Service de nuestro código y así mismo que integremos Swagger en nuestra solución para poder pasar las URL de Swagger a su área de Frontend.

- Crear las pruebas unitarias de todos los Controller y Service de nuestro código con ayuda de Mockito.
- Integrar Swagger a nuestro proyecto (El cliente nos ha dicho que no importa si usamos la versión 2 o 3 de swagger)

Url's de apoyo:

- https://howtodoinjava.com/spring-boot2/testing/spring-boot-mockito-junit-example/
- https://site.mockito.org/
- https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api
- https://www.baeldung.com/spring-rest-openapi-documentation

Entregable:

Que todo nuestro código esté en github y que todos los test este corriendo perfectamente, así como todas clases Controller y Service estén cubiertas al 100% con pruebas unitarias. También se necesita que se proporcione la URL de swagger para verificar todos los endpoints desarrollados y se pueda pasar esta documentación a su área Frontend.