

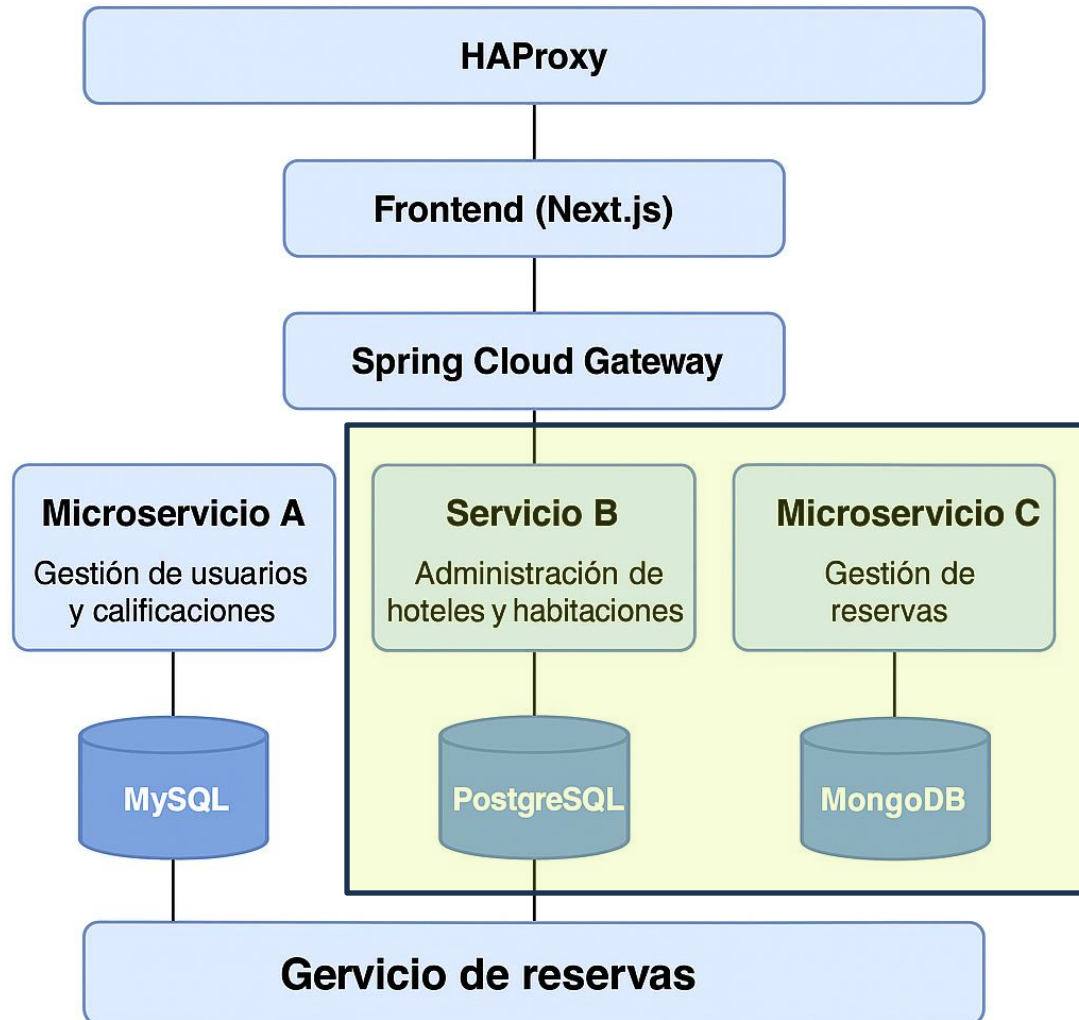
Universidad Tecnológica Nacional
Facultad Regional Santa Fe

Departamento Ingeniería en sistemas de Información

Arquitectura en la nube

TRABAJO PRACTICO DAN - 02

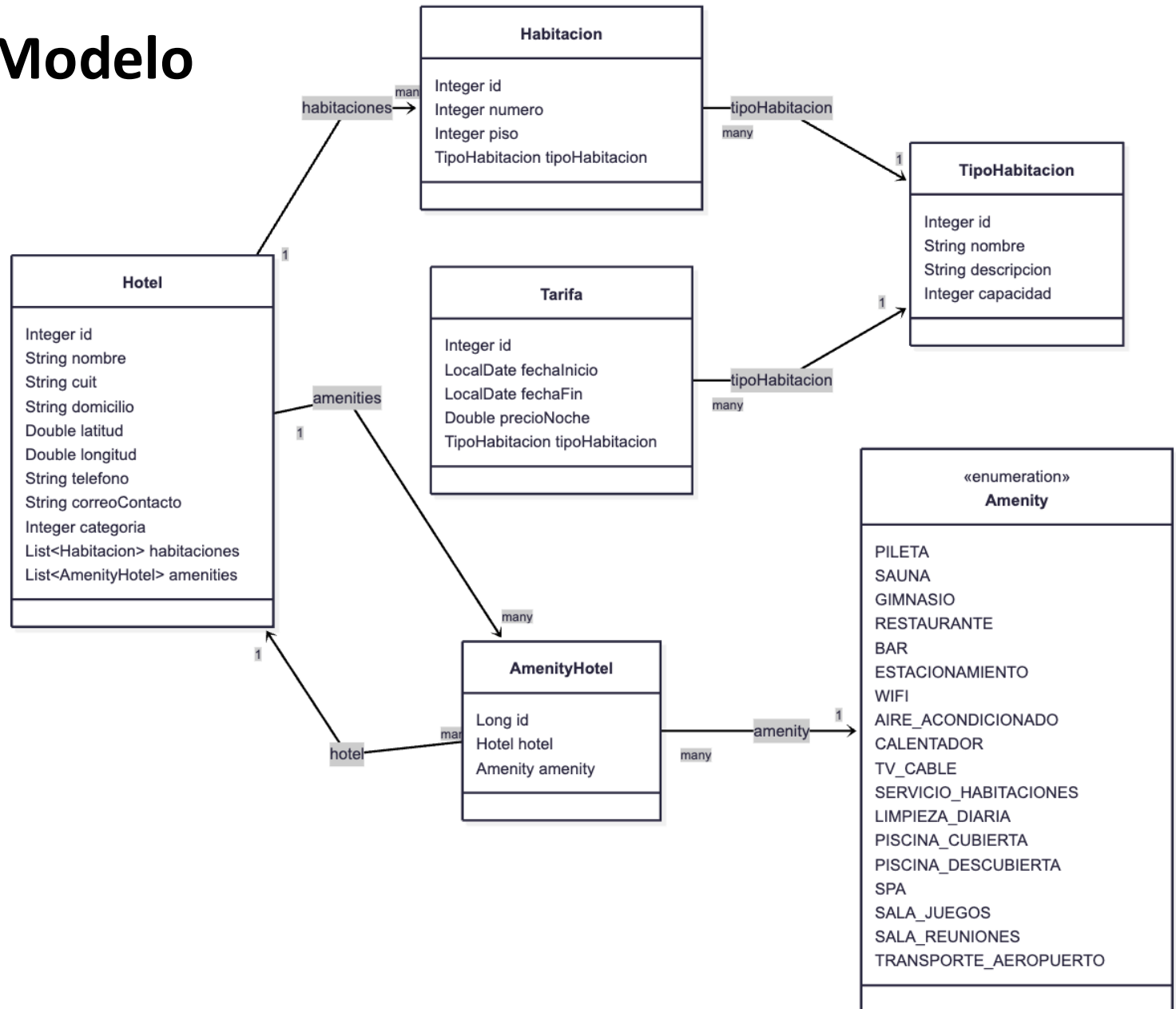
Arquitectura



Microservicio de administracion

- Dar de alta un hotel (todos obligatorios)
 - Nombre
 - CUIT
 - Direccion
 - Coordenadas (lat lng)
- Dar de alta habitaciones en el hotel
 - Numero
 - Piso
 - Tipo de Habitación
 - Disponible
- Tipos de Habitación
 - Nombre, descripción, capacidad
- Tarifa
 - Nombre Tarifa
 - Fecha Inicio
 - Fecha Fin
 - Precio
 - IdTipoHabitacion

Modelo



Reglas de Negocio - Hoteles

- Se pueden dar de alta ingresando los datos solicitados.
 - Solo se puede actualizar la categoria, el teléfono y el correo, el resto de los datos permanecen sin modificar.
- No se pueden dar de baja, solo se puede marcar como “Cerrado” (defina un atributo o una fecha de cierre) y automáticamente marcará como no disponibles todas las habitaciones.
 - Se envia un mensaje al servicio de reserva que el hotel cierra y se crea una reserva del tipo CERRADO, para todas las habitaciones con fecha inicio de hoy y fecha final null)
- Agregar un método PUT que permita agregar una amenity o una lista de amenities a un hotel.
- Agregar un método DELETE que permita eliminar una amenity de un hotel.
- Agregar un método GET que permita consultar hoteles por distintos criterios: nombre, categoria, amenities, direccion, etc.
- Validar con el API de validaciones que los campos sean ingresados

Reglas de Negocio - Habitaciones

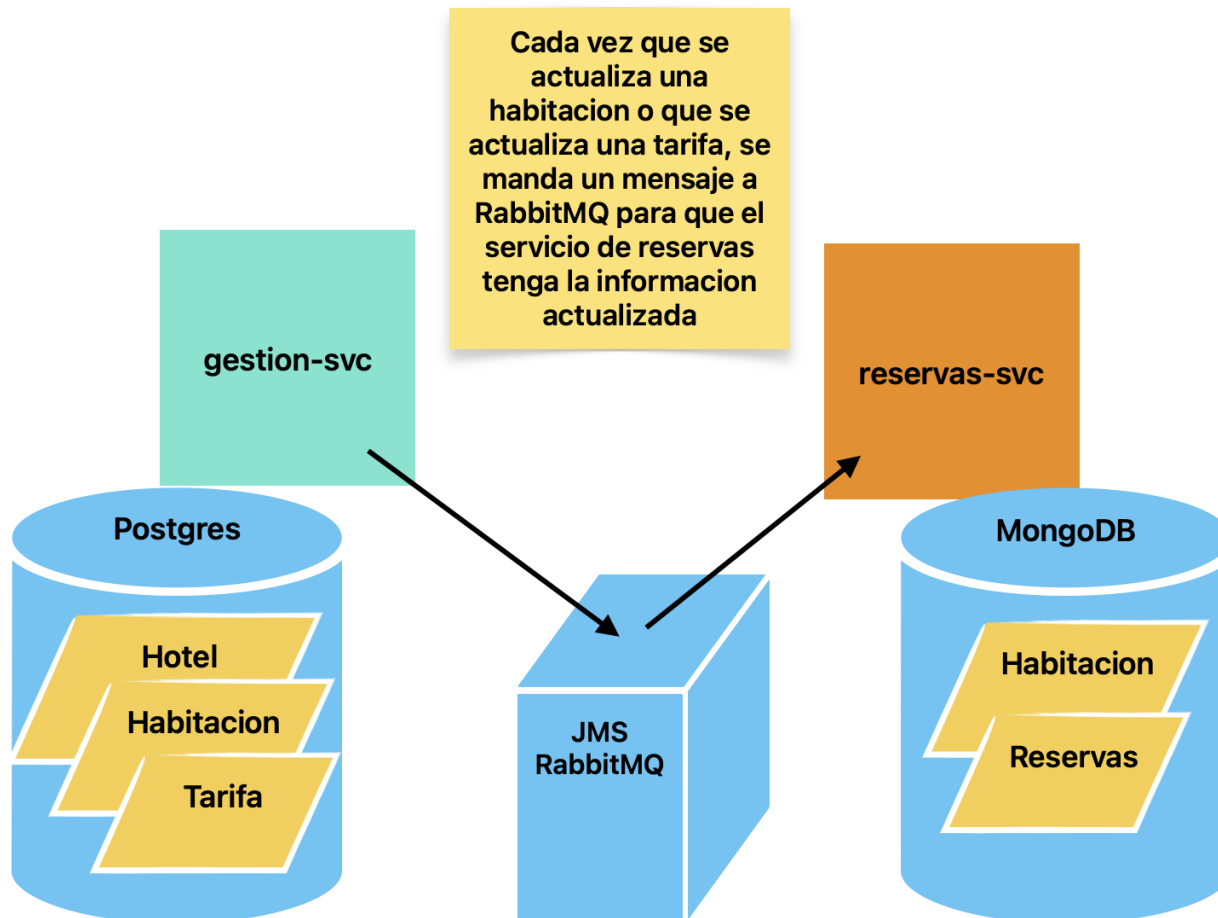
- Las habitaciones se pueden agregar, modificar borrar.
 - Cada vez que ocurre alguna de estas operaciones notificar por JMS al microservicio de reservas.
- Validaciones: el número y el piso deben ser no null y mayor que cero.
- Consultar habitaciones:
 - Por cantidad de huéspedes Y/O
 - Tipo de Habitación Y/O
 - Precio mínimo o máximo
- Crear un endpoint GET que para una habitación me retorne la tarifa vigente.

Reglas de negocio - Tarifas

- Las tarifas solamente se pueden crear (POST) o eliminar (DELETE).
- Una tarifa solo se puede eliminar si existe otra tarifa para ese tipo de habitación. Si la que se elimina es la tarifa actualmente vigente (de acuerdo a las fechas), entonces se busca la tarifa anterior y se la setea como vigente. Si es una tarifa no vigente, se la borra directamente.
- Cuando se agrega una tarifa para un tipo de habitación que ya existe si la misma no tiene fecha de inicio ni de fin entonces, se actualiza la tarifa actual con fecha de finalización el día anterior a hoy y crea un registro con la tarifa nueva con fecha de fin en null y fecha de inicio en el día de hoy
- Programar tarifas especiales: se puede crear una tarifa especial indicando la fecha de inicio y de fin, para ello:
 - Se actualiza la tarifa actual poniendo como fecha de fin el día anterior de la fecha de inicio de la tarifa promocional.
 - Se crea la tarifa promocional con fecha de inicio y de fin
 - Se crea otra tarifa con fecha de inicio el día posterior a la tarifa promocional y fecha de fin en null.

Compartir informacion de la habitacion

- Los datos se guarda en “gestion-svc” y se actualizan asincrónicamente en “reservas-svc”



Configurar RabbitMQ en Docker

- En docker-compose agregar la definición

```
rabbitmq:
  image: rabbitmq:3-management
  container_name: rabbitmq
  restart: unless-stopped
  ports:
    - "5672:5672" # puerto para aplicaciones
    - "15672:15672" # puerto para la consola web de administración
  environment:
    RABBITMQ_DEFAULT_USER: admin
    RABBITMQ_DEFAULT_PASS: admin
  volumes:
    # Monta archivo de definiciones en ruta q lee RabbitMQ al arrancar
    - ./rabbitmq/definitions.json:/etc/rabbitmq/definitions.json
```

Configurar RabbitMQ en Docker

- El archivo definitions tiene
 - El usuario y la password hashed
 - La definicion de un exchange que recibe los mensajes del productor y los enruta a las colas.
 - La definición de una cola que almacena los mensajes hasta que se leen.
 - Un binding para indicar que mensajes se rutean a que cola basados en la clave que se usa.

Configurar la libreria common - Payload

- Definimos un proyecto java standard que tendrá clases que podrán ser compartidas entre todos los proyectos java.
 - La librería esta en la carpeta common
 - En el pom.xml definimos que es hijo del pom parent en la carpeta raíz al igual que el resto de los proyectos spring boot.
 - Creamos un DTO que describa el evento que vamos a enviar via JMS RabbitMQ
 - Incluimos la dependencia en los servicios gestion-svc y reservas-svc.

Microservicio de reservas

Listado de habitaciones

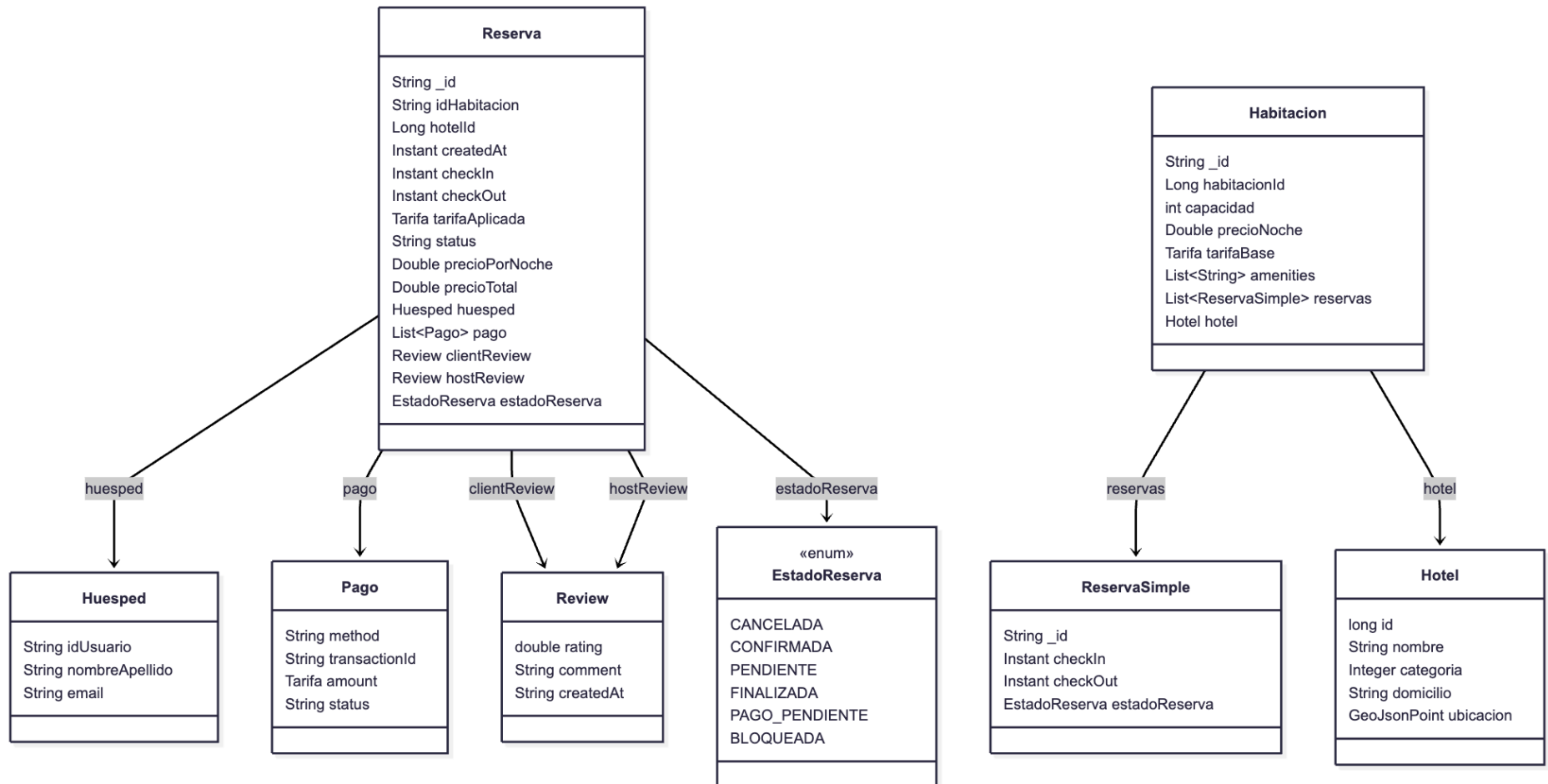
- Se reciben por JMS y se mantiene actualizada la colección de habitaciones.
- Se mantiene también una lista de las fechas de reservas que tiene cada habitación para facilitar la búsqueda de disponibilidad.
- No se pueden crear/actualizar/borrar por interface HTTP. "The source of truth" es el microservicio de gestión de habitaciones y hoteles.

Reservas

- Se pueden dar de alta reservas indicando la habitación, la fecha de check-in y de check-out.
- La reserva quedará en estado REALIZADA
- Cuando se paga al menos la mitad queda CONFIRMADA
- Luego de que el cliente ingresa al hotel queda EFECTUADA.
- Se puede actualizar agregando información de rating luego de la fecha de checkout.
- Puede existir un estado particular de reservas como BLOQUEADA o CERRADA para indicar que una habitación no está disponible

Modelo de Reservas

Diagrama de Clases - reservas-svc model



Reservas Logica

- Cada vez que llega un mensaje con un evento de una habitacion (crecion, actualizacion de datos, actualización de tarifas, o eliminación) se debe procesar y actualizar la colección local de habitaciones.
- Cada vez que se registra una reserva, se debe agregar a la lista de reservas de una habitación que la misma ha sido creada. De esta manera cuando buscamos habitaciones disponibles, buscaremos todas aquellas que no tengan reservas en las fechas dadas.
 - Cuando una reserva se crea se lo hace en el estado RESERVADA.
- Cambios de estado
 - Cuando la reserva tiene al menos un pago pasa a estado CONFIRMADA.
 - Mientras la reserva no tenga un pago puede ser CANCELADA. Cuando es cancelada se elimina de la lista de reservas de una habitación.
 - Cuando se la marca como FINALIZADA es obligatorio que el dueño deje un review y que el pago esté completo caso contrario se marca como ADEUDADA.
 - Las reservas BLOQUEADA se usan para no ofrecer la habitacion.

Microservicio de reservas

- Crear un endpoint que permita buscar habitaciones disponibles por distintos criterios
 - Fecha ingreso y salida
 - Cantidad de huéspedes
 - Precio minimo y maximo
 - Cantidad de estrellas de los hoteles
 - Amenities
 - Distancia en metros o kilometros de una latitud longitud dada como referencia.
 - Para esto es necesario usar un indice Geo2D
<https://www.mongodb.com/docs/manual/core/indexes/index-types/index-geospatial/>
- Una habitación está disponible para una fecha de entrada y de salida si no tiene reservas cuya fecha de ingreso sea menor o igual que la fecha de entrada y la fecha de salida sea mayor o igual que la fecha de ingreso

Ejecutar la aplicacion

En primer lugar ejecutar en el directorio raiz (tp-2025)

`./mvnw clean install -DskipTests`

esto creará todos los artefactos de las librerías y los proyectos.

```
martindominguez@martbook tp-2025 % ./mvnw clean install -DskipTests
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
[INFO] tp2025 [pom]
[INFO] user-svc [jar]
[INFO] dan-common-lib [jar]
[INFO] gestion-svc [jar]
[INFO] reservas-svc [jar]
[INFO]
```

Reactor Summary:

```
tp2025 1.0.0-SNAPSHOT ..... SUCCESS [ 0.069 s]
user-svc 0.0.1-SNAPSHOT ..... SUCCESS [ 1.782 s]
dan-common-lib 0.0.1-SNAPSHOT ..... SUCCESS [ 0.249 s]
gestion-svc 0.0.1-SNAPSHOT ..... SUCCESS [ 0.672 s]
reservas-svc 0.0.1-SNAPSHOT ..... SUCCESS [ 0.596 s]
```

BUILD SUCCESS


Ejecutar aplicaciones spring boot en docker

- Para ejecutar las aplicaciones spring boot en docker se realizaron algunas acciones:
 - En cada aplicación spring boot se agregó un dockerfile que construye la imagen de la aplicación

```
FROM eclipse-temurin:21-jre-alpine
WORKDIR /app

COPY target/*.jar app.jar

EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```



Para que exista el jar en el directorio target es que primero tenemos que realizar el build con maven

Ejecutar aplicaciones spring boot en docker

- En los archivos `application.properties` las bases de datos y los servidores JMS entre otros, ahora apuntan al nombre DNS del contenedor donde ejecutan
 - por ejemplo `spring.datasource.url=jdbc:postgresql://postgres-db:5432/appdb`
- Dado que ahora tanto las aplicaciones spring boot como las bases de datos y servidores JMS ejecutan en contenedores docker

Ejecutar aplicaciones spring boot en docker

- Se actualizó el archivo docker-compose agregando cada servicio

▷ Run Service

user-svc:

build:

context: ../services/user-svc

container_name: user-svc

restart: unless-stopped

ports:

- "8081:8080"

depends_on:

- mongodb

- mysql

- rabbitmq

environment:

SPRING_PROFILES_ACTIVE: default

reservas-svc:

build:

context: ../services/reservas-svc

container_name: reservas-svc

restart: unless-stopped

ports:

- "8082:8080"

depends_on:

- mongodb

- rabbitmq

environment:

SPRING_PROFILES_ACTIVE: default

Ejecutar aplicaciones spring boot en docker

- Se actualizó el archivo docker-compose agregando cada servicio

```
gestion-svc:
  build:
    context: ../services/gestion-svc
  container_name: gestion-svc
  restart: unless-stopped
  ports:
    - "8083:8080"
  depends_on:
    - mysql
    - rabbitmq
  environment:
    SPRING_PROFILES_ACTIVE: default
```

Notar que cada servicio ahora se accedera en un puerto diferente:

- 8081 usuarios
- 8082 reservas
- 8083 gestion

Ejecutar aplicaciones spring boot en docker

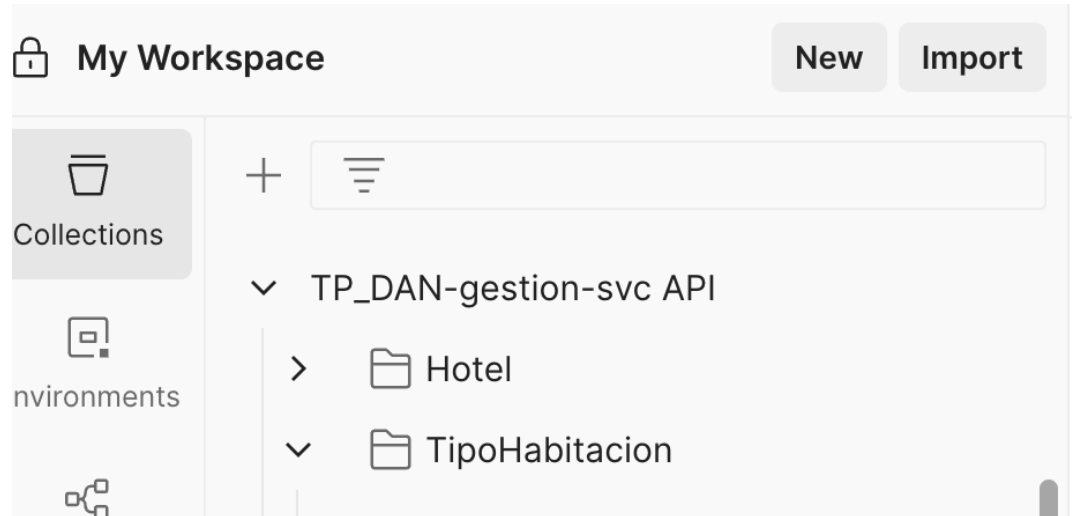
- Ejecutar
 - `docker compose -f infra/docker-compose.yml up --build -d`

```
martindominguez@martbook tp-2025 % docker compose -f infra/docker-compose.yml up --build -d
WARN[0000] /Users/martindominguez/Documents/utn/dan/2025/tp-2025/infra/docker-compose.yml: the attri
bute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 3.9s (22/22) FINISHED                                docker:desktop-linux
=> [gestion-svc internal] load build definition from Dockerfile      0.0s
```

```
✓ gestion-svc           Built
✓ reservas-svc          Built
✓ user-svc              Built
✓ Container mongodb      Running
✓ Container mysql-users  Running
✓ Container rabbitmq      Running
✓ Container postgres-db  Running
✓ Container mongo-express Running
✓ Container pgadmin       Running
✓ Container phpmyadmin    Running
✓ Container gestion-svc  Started
✓ Container reservas-svc Started
✓ Container user-svc     Started
```

Probar los endpoints con postman

- Cada proyecto tiene un archivo “zzzz.postman_collection.json”.
- Debe importarlo en postman



> target	--	Folder	Today at 5
> src	--	Folder	Today at 1
reservas-svc.postman_collection.json	7 KB	JSON File	Today at 1
pom.xml	2 KB	XML	Today at 1
mvnw.cmd	7 KB	Document	Today at 1
mvnw	11 KB	Unix Ex...ble File	Today at 1
modelo.mmd	2 KB	Document	Today at 1
Dockerfile	165 bytes	Document	Today at 1

Probar los endpoints con postman

- Ajustar cada URL a los puertos de la aplicacion (8081 / 8082 / 8083)
- Ajustar los datos si se desea realizar distintas pruebas).
- Ejemplo creación de un hotel

The screenshot shows the Postman application interface. On the left, a sidebar lists API collections under 'space'. The main area displays a 'POST Crear Hotel' request to 'http://localhost:8083/hoteles'. The 'Body' tab is selected, showing a JSON payload for creating a hotel. The response area at the bottom shows a '200 OK' status with a JSON response containing the created hotel's details.

Request:

```
POST http://localhost:8083/hoteles
```

Body (JSON):

```
{
  "nombre": "Hotel Test 2",
  "cuit": "20123456789",
  "domicilio": "Calle 123",
  "latitud": -31.4167,
  "longitud": -64.1833,
  "telefono": "3511234567",
  "correoContacto": "contacto@hotel.com",
  "categoria": 3
}
```

Response:

```
{
  "id": 2,
  "nombre": "Hotel Test 2",
  "cuit": "20123456789",
  "domicilio": "Calle 123",
  "latitud": -31.4167,
  "longitud": -64.1833,
  "telefono": "3511234567"
}
```

Probar los endpoints con postman

- Ejemplo creación de una habitacion

The screenshot shows the Postman interface for a POST request to `http://localhost:8083/habitaciones`. The request body is a JSON object with the following structure:

```
{
  "numero": 499,
  "piso": 52,
  "tipoHabitacion": {"id": 4},
  "hotel": {"id": 1}
}
```

The response is a 200 OK status with a response time of 580 ms and a body size of 434 B. The response body is a JSON object with the following structure:

```
{
  "id": 7,
  "numero": 499,
  "piso": 52,
  "tipoHabitacion": {
    "id": 4,
    "nombre": null,
    "descripcion": null,
    "capacidad": null
  },
  "hotel": {
    "id": 1,

```


Probar los endpoints con postman

- Ejemplo consultas de habitaciones disponibles para reservar.

The screenshot shows a Postman interface with a GET request to `http://localhost:8082/habitaciones`. The response is a 200 OK status with a response time of 193 ms. The response body is displayed in JSON format, showing an array of room objects.

Params Auth Headers (6) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body 200 OK • 193 ms •

{ } JSON Preview Visualize

```
1  [
2    {
3      "id": "6855a4c72a6144ee19f74230",
4      "habitacionId": 7,
5      "capacidad": null,
6      "precioNoche": 0.0,
7      "amenities": null,
8      "reservas": null,
9      "hotel": null,
10     "idTipoHabitacion": null,
11     "tipoHabitacion": null
12   }
13 ]
```