Introducción

El Servicio web de administración de alquileres es una aplicación basada en microservicios que permite administrar el proceso de alquiler de juegos y clientes. Proporciona funcionalidades para el registro y gestión de clientes, así como el manejo de juegos y alquileres. Este documento técnico proporciona una descripción general de la arquitectura, tecnologías utilizadas, y los principales componentes del servicio web.

Arquitectura

El servicio web se basa en una arquitectura de microservicios, lo que significa que está compuesto por varios servicios independientes y altamente cohesivos. Cada servicio se encarga de una funcionalidad específica y se comunica con otros servicios a través de protocolos RESTful.

La arquitectura consta de los siguientes microservicios:

- **Database Service:** Es el servicio de base de datos que utiliza una imagen de MySQL 8 para almacenar los datos del servicio web.

- **Load Balancer Service:** Es el servicio de balanceo de carga que utiliza una imagen personalizada "ejrebolledo/load-balancer-service" para distribuir las solicitudes entrantes entre los demás servicios.

- **Integration Service:** Es el servicio de integración que se encarga de integrar y coordinar las operaciones entre los diferentes servicios.

- **Client Management Service:** Es el servicio encargado de la gestión de clientes, como el registro, consulta y actualización de información de los clientes.

- **Game Management Service:** Es el servicio encargado de la gestión de juegos, incluyendo la creación, actualización y consulta de información de los juegos.

- **Rent Management Service:** Es el servicio encargado de la gestión de alquileres, incluyendo la creación, actualización y consulta de información de los alquileres.

Tecnologías utilizadas

El servicio web de administración de alquileres se desarrolla utilizando las siguientes tecnologías:

- **Spring Boot:** Se utiliza como el framework principal para el desarrollo de los servicios, proporcionando características de configuración automática, inyección de dependencias y facilidades para crear aplicaciones basadas en microservicios.

- **MySQL:** Se utiliza como el sistema de gestión de bases de datos relacional para almacenar los datos del servicio web.

- **Docker:** Se utiliza para contenerizar los servicios y facilitar su despliegue y distribución.

- **Docker Compose:** Se utiliza para orquestar y administrar los contenedores Docker de los diferentes servicios.

- **Maven:** Se utiliza como la herramienta de construcción y gestión de dependencias para el proyecto.

Dependencias de los servicios

En el proyecto del archivo POM que has proporcionado, se utilizan varias dependencias de Maven para agregar funcionalidades específicas al servicio web. A continuación, se explica el propósito de cada una de las dependencias incluidas:

- **spring-boot-starter-web:** Esta dependencia permite la creación de aplicaciones web utilizando Spring MVC. Proporciona todas las bibliotecas necesarias para desarrollar y ejecutar una aplicación web.

- mysql-connector-java: Esta dependencia agrega el controlador de MySQL para que la aplicación pueda conectarse a una base de datos MySQL y realizar operaciones de persistencia.

- **spring-boot-starter-data-jpa:** Esta dependencia facilita la integración de Spring Data JPA en la aplicación. Spring Data JPA es una biblioteca que simplifica el acceso y la manipulación de datos en una base de datos relacional utilizando el enfoque de programación orientada a objetos.

- **springdoc-openapi-starter-webmvc-ui:** Esta dependencia agrega soporte para la generación automática de documentación API utilizando el estándar OpenAPI. Permite exponer la documentación de la API en formato Swagger UI, lo que facilita la visualización y prueba de los puntos finales de la API.

- **spring-boot-devtools:** Esta dependencia proporciona herramientas de desarrollo adicionales para facilitar el ciclo de desarrollo. Incluye funciones como la reinicialización automática de la aplicación cuando se detectan cambios en el código fuente y la habilitación de una configuración más conveniente durante el desarrollo.

- **spring-boot-starter-test:** Esta dependencia agrega las bibliotecas necesarias para escribir pruebas unitarias y de integración en la aplicación utilizando Spring Testing Framework.

- **spring-cloud-starter-netflix-eureka-client:** Esta dependencia permite que la aplicación actúe como un cliente de Eureka, que es un servicio de registro y descubrimiento de microservicios. Permite que la aplicación se registre en un servidor Eureka y descubra otros servicios registrados en el mismo servidor.

- **spring-boot-starter-validation:** Esta dependencia agrega soporte para la validación de datos utilizando las anotaciones de validación de Bean Validation. Permite validar automáticamente los datos de entrada y generar mensajes de error en caso de violaciones de las reglas de validación.

Estas dependencias proporcionan funcionalidades clave para desarrollar un servicio web con Spring Boot, incluyendo la comunicación con una base de datos, la documentación de la API, pruebas automatizadas y características adicionales para facilitar el desarrollo y la validación de datos.

Base de datos

En el archivo docker-compose.yaml proporcionado, se define un servicio llamado database que utiliza la imagen de Docker mysql:8-oracle para crear y alojar la base de datos. A continuación, se muestra cómo se crea y se pobla la base de datos en el servicio rent-management-service:

1. Creación de la base de datos:
    - En el servicio database, se especifican las variables de entorno MYSQL_ROOT_PASSWORD y MYSQL_DATABASE. MYSQL_ROOT_PASSWORD define la contraseña del usuario root de MySQL, y MYSQL_DATABASE especifica el nombre de la base de datos que se creará.
    - Cuando se inicia el servicio database, se ejecuta una verificación de salud (healthcheck) para asegurarse de que el servicio de la base de datos esté disponible y funcione correctamente.
2. Creación de tablas
    - La dependencia spring-boot-starter-data-jpa permite la creación automática de tablas en la base de datos basándose en las entidades de dominio definidas en el proyecto. Hibernate, como parte de Spring Data JPA, se encarga de generar las tablas utilizando la estrategia de generación de esquema predeterminada.
3. Población de la base de datos:
    - En el servicio rent-management-service, se especifica un volumen que mapea el archivo populate.sql local (./populate.sql) al contenedor del servicio (/app/populate.sql).
    - El archivo populate.sql contiene instrucciones SQL para poblar la base de datos con datos iniciales.
    - En el método executePopulateSqlFile() de la clase TriggerManager, se verifica si la tabla renta_juego ya contiene datos. Si la tabla está vacía, se ejecuta el archivo populate.sql para poblar la base de datos.

- Se lee el contenido del archivo populate.sql y se divide en declaraciones SQL individuales. Cada declaración SQL se ejecuta utilizando jdbcTemplate.execute(sqlStatement) para insertar los datos en la base de datos.

Puede consultar el diseño de la base de datos, modelo Entidad-Relación y modelo relacional, en las imágenes adjuntas en el repositorio.

Instrucciones de ejecución

Para ejecutar el servicio web de administración de rentas, siga los siguientes pasos:

1. Asegúrese de tener Docker y Docker Compose instalados en su sistema.
2. Clone el repositorio del proyecto desde el repositorio proporcionado.
3. Asegúrese de que el archivo "docker-compose.yaml" y el archivo "populate.sql" (proporcionado en el repositorio) se encuentren en el mismo directorio.
4. Abra una terminal y navegue hasta el directorio donde se encuentran los archivos mencionados.
5. Ejecute el siguiente comando: docker-compose up
6. Esto iniciará los contenedores Docker para cada servicio y comenzará a ejecutar el servicio web.

Una vez que los servicios estén en funcionamiento, puede acceder a ellos a través del integration service, expuesto en la URL: *http://localhost:8200*

Puede utilizar herramientas como Postman o cualquier navegador web para interactuar con los servicios utilizando los endpoints proporcionados. Cuya documentación está disponible a través de la url *http://localhost:8200/swagger-ui/index.html* o se puede acceder a una versión más limpia de la documentación a partir de el archivo api_documentation.yaml que se encuentra en la carpeta del repositorio.