UNIVERSIDAD DE LA SIERRA JUÁREZ

CARRERA:
LICENCIATURA EN INFORMÁTICA

ASIGNATURA:
SISTEMAS DISTRIBUIDOS

NOMBRE DEL TEMA:
MICROSERVICIOS

Profesor: M.T.C.A Francisco Javier Méndez Vázquez

Alumno: Bernardino Hernández Rojas
Semestre: Octavo

19 de junio de 2023

Universidad de la Sierra Juárez
Licenciatura en Informática

Índice

Resumen

El trabajo realizado ejemplifica la implementación de microservicios básicos, no obstante la sencillez no esta peleada con el aprendizaje, pues a pesar de ser sencillos nos permiten la amplia comprensión sobre los conceptos de microservicios, ademas aprendemos a usar otras archivos de datos como lo son Json y diferentes peticiones por medio de la url(Put, Get, Post, Delete).
Durante el proceso de desarrollo nos podremos encontrar con problemas, cabe mencionar que los problemas relacionados al desarrollo no son imposibles de solucionar, debido a que la comunidad de desarrollo en Java con el marco Sprig Boot son bastas y extensas.
Se desarrollaron tres microservicios satisfactoriamente, en este documento hablamos de manera general sobre el desarrollo del proyecto.

1. Introducción.

El siguiente reporte aborda la implementación de **microservicios** utilizando el lenguaje de programación ”Java” empleando el framework llamado **Spring Boot**, nos enfocamos en el principio de ”fragmentación”, el cual nos dice que una aplicación monolítica se puede descomponer en piezas de software más pequeñas independientes que se encargan de una tarea especifica dentro de una aplicación más grande.

La importancia de conocer el funcionamiento y la implementación de microservicios se ve refleja en las nuevas tecnologías que tienen como tendencia la migración a la nube...

2. Objetivo general

Desarrollar microservicios capaces de alimentarse mutuamente y con esto lograr el escalamiento hacia una aplicación más grande sin comprometer su funcionalidad, brindando flexibilidad, escalabilidad, y tolerancia a fallos.

2.1. Objetivos específicos

- Desarrollar microservicios capaces de implementar nuevas funcionalidades a un sistema mayor.
- Comprender la funcionalidad de un microservicio.
- Aprender a desarrollar microservicios con Spring Boot.
- Utilizar Postman para probar la funcionalidad de los microservicios desarrollados.
- Aprender a dar funcionalidad a un microservicio desde requerimientos generales.
- Analizar los requerimientos generales y proporcionar una solución lógica y genérica sobre el microservicio a desarrollar.

3. Desarrollo

3.1. Tecnologías y Software empleados

- IntelliJ IDEA 2023.1.2 (Ultimate Editon)
- Java correto-17 versión 17.0.17)
- Git versión 2.34.1
- MySQL Workbench 8 versión 8.0.32
- Mysql Ver 8.0.33-0ubuntu0.22.04.2
- Spring Boot 3.0.1
- Ubuntu 22.04 LTS
- Postman v10.15.1

3.2. Procedimiento

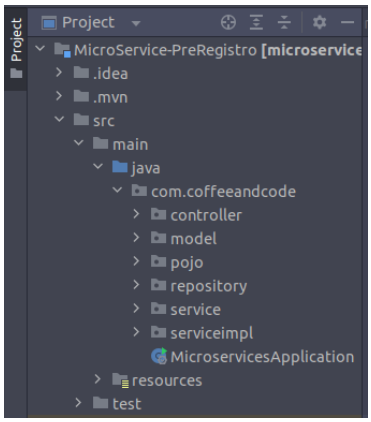
Desde un requerimiento general se debe realizar un análisis a profundidad sobre las necesidades, dicho análisis debe ser genérico; es decir debe valer para cualquier aplicación que requiera de esa funcionalidad.

Una vez teniendo el análisis se revisa la documentación de Spring Boot sobre la implementación de los microservicios, con esto podemos transportar el análisis realizado en papel a código de programación, en este caso utilizamos Java como lenguaje de programación principal apoyándonos de un framework que facilita la implementación de funcionalidades que son demandantes de tiempo al estar desarrollando como lo es la conexión a base de datos, facilidad en integrar dependencias, facilidad de visualizar el desarrollo en el navegador integrando un contenedor de aplicaciones web para ser desplegado.

3.2.1. Patrón MVC

Utilizamos el patrón de desarrollo MVC para logar la estructura de carpetas ordenadas sin mezclar la lógica de negocio con la funcionalidad a implementar. El patrón de desarrollo MVC se compone de esta estructura:

- Modelos: Representan los datos y la lógica de negocio de la aplicación, con ellas gestionamos las entidades de las bases de datos, validaciones a nivel de tipo de datos y limitantes de aceptación. En estos componentes es donde se establece la cardinalidad de las entidades a utilizar.
- Vista: Es la interfaz de usuario de la aplicación, se encarga de la representación visual de los datos y la interacción con el usuario, en este caso omitimos esta parte en microservicios ya que estos solo devuelven colecciones de tipo Json o xml según sean las necesidades de la aplicación a escalar.
- Controlador: Actuá como intermediario entre el modelo y la vista, además su parte principal es controlar el flujo de la aplicación, coordinando las diferentes acciones y actualizaciones internas del sistema.



Se decide utilizar este patrón de desarrollo, debido a que permite la modularidad y flexibilidad en el desarrollo de nuestro microservicio, cada componente tiene una responsabilidad especifica y se puede modificar o remplazar de manera independiente, la ejemplificación del patrón MVC la podemos observar en la siguiente imagen.

3.2.2. Servicios, interfaces de repositorios, implementación de interfaces.

Una de las facilidades que permite Spring Boot es la inyección de dependencias, con estas se hace referencia a los servicios e interfaces de repositorios, en otros componentes, esto similar al funcionamiento de los controladores.
Se utilizan para dividir las responsabilidades y facilitar la prueba y mantenimiento del código, se podría decir que también son pieza importante donde se permite la modularidad, escalabilidad y flexibilidad del sistema.

- Servicios: Aquí se implementa la lógica de negocio de una aplicación, representan la capa intermedia entre los controladores y los repositorios, con ellas se busca proporcionar funcionalidad especifica y operaciones más complejas.

- Interfaces de repositorios: Definen métodos para interactuar con la capa de almacenamiento de datos(bases de datos, o servicios web). Permiten la abstracción de la lógica de acceso a datos y proporcionan una forma estandarizada para las operaciones CRUD(Crear, Leer, Actualizar, Eliminar) sobre los objetos persistentes.

- Implementación de interfaces: Contienen la creación de clases que implementan las interfaces definidas, dichas clases contienen la lógica concreta para llevar a cabo las operaciones definidas en las interfaces, es decir aquí es en donde se desarrolla el método completo.

3.2.3. Desarrollo de microservicios

Iniciamos con el proceso de desarrollo de microservicios, primero definimos que tipo de datos aceptará y entregará nuestro microservicio, para este caso especifico se decide que será en Json.

Utilizando MVC definimos las entidades de nuestra base de datos en el apartado **model**, en este se establece la cardinalidad que hemos definido en nuestro diagrama entidad relación que diseñamos durante el análisis de requerimientos, es aquí donde definimos los atributos que contendrá la tabla, ademas del nombre de la misma todo con notaciones que pertenecen a dependencias utilizadas para **maven** apoyados del framework **Spring Boot.**

Utilizamos los controladores para poder manipular las re-direcciones dentro de nuestro sistema, ademas esta es la capa donde definimos métodos de entrada y salida de datos, es decir por aquí ingresan las peticiones y retornamos las respuestas, también en este apartado es en donde

inyectamos las dependencias InterfaceXXXXXService, dichas dependencias proveen los métodos para traer o insertar información a nuestra base de datos.

Utilizamos la implementación los servicios para poder comunicarnos con el repositorio, así podemos tener acceso a base de datos, definimos métodos como son listar, crear, actualizar, búsqueda por x campo. Dichos métodos serán bien desarrollados en ImplxxxxxxxService y serán consumidos en el controlador para brindar funcionalidad a nuestro microservicio.

Listing 1: Ejemplo de definición de interfacexxxxxxService

```
1  package com.coffeeandcode.service;
2
3  import com.coffeeandcode.model.AcademicDataModel;
4
5  import java.util.List;
6  import java.util.Optional;
7
8  public interface InterfaceAcademicDataService {
9      public List<AcademicDataModel> listallAcademicDatas();
10
11     public Optional<AcademicDataModel> getAcademicDataById(
           int idAcademicData);
12
13     public AcademicDataModel createAcademicData(
           AcademicDataModel academicDatamodel);
14
15     public AcademicDataModel saveAcademicData(
           AcademicDataModel academicDatamodel);
16
17     public AcademicDataModel updateAcademicData(int
           idAcademicData, AcademicDataModel academicDatamodel);
18
19     public boolean deleteAcademicData(int idAcademicData);
20 }
```

3.2.4. Prueba de funcionamiento con Postman

Para probar la funcionalidad de nuestro servicios, existen dos maneras, una es con un software llenado **Postman** y la otra es desde la terminal de nuestro sistema, donde estemos trabajando. En esta ocasión probaremos con Postman debido a la facilidad que nos provee, antes cabe mencionar que los métodos están desarrollados en el controlador, dichos metodos reciben json y devuelven json, el contenido es definido por la logia implementada.

Listing 2: Ejemplo de metodos desarrollados del lado del controlador

```
1  public class PersonalDataController {
2  private final InterfacePersonalDataService
       interfacePersonalDataService;
3
4  @Autowired
```

```
5          public PersonalDataController(InterfacePersonalDataService
               interfacePersonalDataService) {
6              this.interfacePersonalDataService =
                   interfacePersonalDataService;
7          }
8
9          @PostMapping("/save")
10         public ResponseEntity<Object> createPersonalData(@RequestBody
               PersonalDataModel personalData)...
11
12         @GetMapping("/list")
13         public ResponseEntity<Object> index()...
14
15         @PutMapping("/update/{id}")
16         public ResponseEntity<Object> updatePersonalData(
               @PathVariable int id, @RequestBody PersonalDataModel
               personalData)...
17
18         @DeleteMapping("/delete/{id}")
19         public ResponseEntity<String> deletePersonalData(
               @PathVariable int id)...
20
21         @GetMapping("/search/{id}")
22         public ResponseEntity<Object> getPersonalData(@PathVariable
               int id)
23
24         }
```

Se omite el desarrollo debido a que algunos métodos son demasiados largos para mostrarlos, procederemos a explicar el funcionamiento breve de cada uno de los métodos.

- createPersonalData: Nos permite ingresar un json con una estructura especifica y mediante interfacePersonalDataService podremos agregar los datos contenidos en el json a la base de datos, dicho método nos devuelve el json del elemento que se agrega, si por alguna cuestión no es posible agregar nos devuelve un mensaje de error diciendo que no fue posible el almacenamiento.
- index: ejecuta una consulta en la cual nos regresa todos los datos contenidos en la base de datos.
- updatePersonalData: Este método a razón de un id nos permite actualizar el contenido directamente de la base de datos del elemento deseado.
- deletePersonalData: Nos permite eliminar un elemento a razón de recibir el id de dicho elemento.
- getPersonalData: Nos devuelve un objeto especifico, lo pediremos a razón del id proporcionado.

Dichos métodos anteriormente explicados se pueden combinar entre si para proveer de más funcionalidades a nuestro microservicio, si alguna consulta no hace lo que se requiere y se pretende ser las especifico podemos definir una consulta nativa de SQL en el repositorio correspondiente.

4. Conclusión

La implementación de microservicios es similar al desarrollo de un sistema Web, con la única diferencia que se utilizan los métodos: Post para agregar, Put para actualizar, Get pata leer, Delete para eliminar.
Si no tenemos cuidado cuando se realicen la implementación de la cardinalidad entre nuestras entidades podemos tener errores de lectura como establecer bucles infinitos en las cardinalidades bidireccionales, no son tan complicadas de solucionar pero si pueden llegar a ser un dolor de cabeza, más si es nuestro primer acercamiento a microservicios, esto se debe a la serialización y deserialización de los objetos Json que manejamos para dicha implementación.
Estos microservicios se pueden consumir entre ellos mismo o podemos consumirlos desde aplicaciones monolíticas, cabe mencionar que cada uno de los tipos de desarrollo pueden o no ser fundamentales para un proyecto concreto y esto dependerá de las necesidades del proyecto, no se deben usar solo por novedad, se debe realizar un análisis minucioso de los requerimientos y poder verificar todas las alternativas adaptables al sistema por que puede suceder que el sistema a desarrollar no se pretende escalar entonces no tendría sentido implementar microservicios.

Podemos encontrar la implementación en la siguiente liga del repositorio GitHub:
https://github.com/BernardinoHeRo/MICROSERVICIOS.git