



ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

**DEPARTAMENTO DE ELÉCTRICA Y
ELECTRÓNICA**

INGENIERÍA EN SOFTWARE

CERTIFICACIÓN II

INTEGRANTES:

VIQUE ALMEIDA DAVID ALBERTO

LUIS ARMANDO CONTERON CHICAIZA

DOCENTE:

ALVARO DANILO UYAGUARI UYAGUARI

NRC: 2072

PERÍODO ACADÉMICO

NOVIEMBRE 2020 – ABRIL 2021

Tema: Diseño e implementación de una arquitectura de microservicios para un escolástico de matriculación.

Objetivos:

Diseñar e implementar la arquitectura de microservicios para la matriculación de los estudiantes en las materias mediante la utilización del framework Spring Boot y utilización de servicios proporcionados por Netflix.

Repositorio en Github

<https://github.com/DavidVique1998/ProyectoMicroservicioEscolastico.git>

(Backend y Frontend - Solucionado)

Introducción

La integración de sistemas heterogéneos se apoya comúnmente en Plataformas de Integración (PI). Estas plataformas consisten en infraestructura especializada, que provee mecanismos para resolver incompatibilidades entre sistemas con el fin de posibilitar su comunicación. En este ámbito, el avance y la expansión de la computación en la nube ha generado nuevos escenarios y requerimientos sobre las PIs en términos de escalabilidad y eficiencia, entre otros. Por otro lado, la arquitectura de microservicios ha surgido recientemente impulsada por la industria, y está ganando creciente popularidad. Esta posee ventajas como el escalamiento independiente y la mantenibilidad que podrían beneficiar a las PIs en distintos escenarios. Por tanto, resulta de interés explorar las ventajas, desafíos y alternativas que introduce la arquitectura de microservicios en estas plataformas (Nebel,2019).

La arquitectura de microservicios permite tener una gran escalabilidad y flexibilidad sin olvidar la robustez que los sistemas de este tipo requieren, mediante la implementación del acceso de usuarios Outh2 se logra tener una capa de seguridad para acceder a cada microservicio los cuales están suscritos a eureka ya que trabajan con puertos aleatorios y se encuentran identificados por su nombre. Para acceder a sus configuraciones como url de enlace a las bases de datos se implementa un servicio que está conectado con github donde se tiene los archivos de configuraciones ya sea de desarrollo o de producción. Zuul, será la puerta de enlace para los demás microservicios junto con balanceadores de carga y Hystrix que se encargará de direccionar a los métodos ante cualquier fallo de las peticiones esta herramienta será la puerta de enlace a los demás servicios y cabe recalcar que cada uno de ellos contará con su propia base de datos implementando la conexión mediante tablas identificadoras.

Estado del arte

Outh 2: OAuth 2.0 es el protocolo estándar de la industria para la autorización. OAuth 2.0 se centra en la simplicidad del desarrollador del cliente al tiempo que proporciona flujos de autorización específicos para aplicaciones (Sucasas, 2019).

Zuul: Es una pasarela de aplicaciones L7 que proporciona capacidades de enrutamiento dinámico, monitorización, resiliencia, seguridad y mucho más. Por favor, consulte la wiki para el uso, la información (Siriwardena, 2020)

Hystrix: Hystrix es una biblioteca de latencia y tolerancia a fallos diseñada para aislar los puntos de acceso a sistemas remotos, servicios y bibliotecas de terceros, detener los fallos en cascada y permitir la resiliencia en sistemas distribuidos complejos en los que los fallos son inevitables.

Cloud Config: Spring Cloud Config proporciona soporte del lado del servidor y del lado del cliente para la configuración externa en un sistema distribuido. Con el Servidor Config (Molchanov, 2018)

Eureka: Eureka es un servidor para el registro y localización de microservicios, balanceo de carga y tolerancia a fallos. La función de Eureka es registrar las diferentes instancias de microservicios existentes, su localización, estado, metadatos (Hoffmann, 1992)

PostgreSQL: PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance (PostgreSQL, 1996).

MySQL: MySQL is the most popular Open Source Relational SQL Database Management System. MySQL is one of the best RDBMS being used for developing various web-based software applications. (MySQL, 2001)

Zipkin: Zipkin es un sistema de trazado distribuido. Ayuda a recopilar los datos de sincronización necesarios para solucionar los problemas de latencia en las arquitecturas de servicios. Sus características incluyen tanto la recopilación como la búsqueda de estos datos (Zipkin, 2015).

RabbitMQ: RabbitMQ es un broker de mensajería de código abierto, distribuido y escalable, que sirve como intermediario para la comunicación eficiente entre productores y consumidores.

Desarrollo

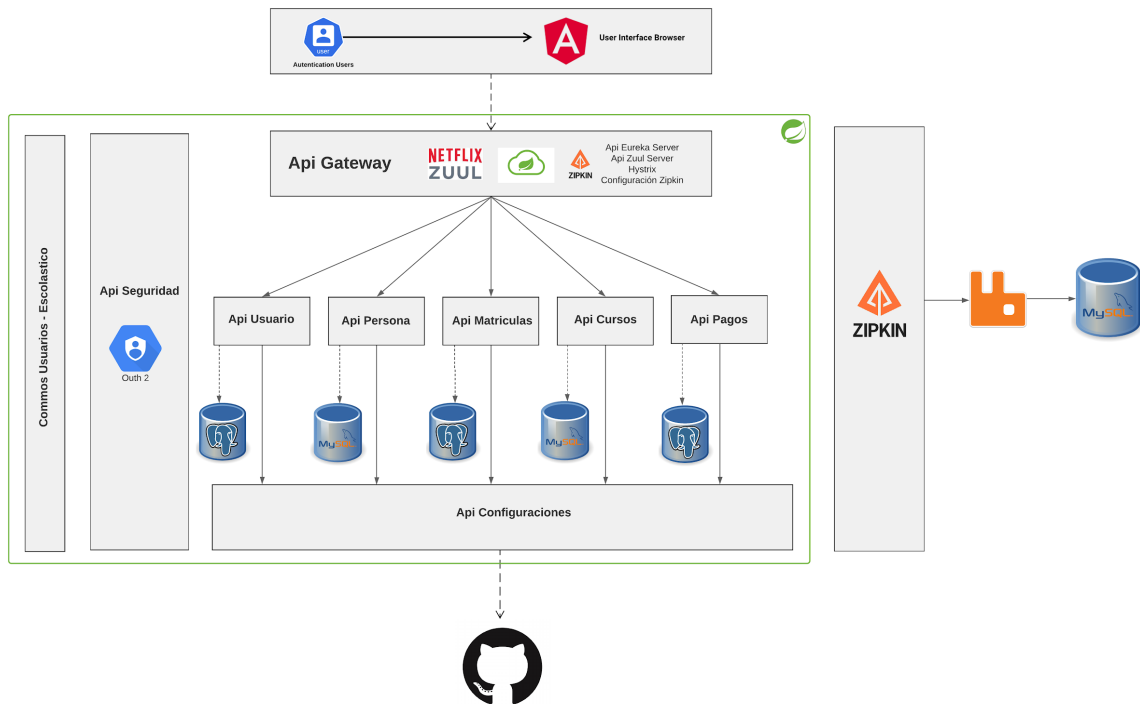


Imagen 1. Arquitectura del Sistema

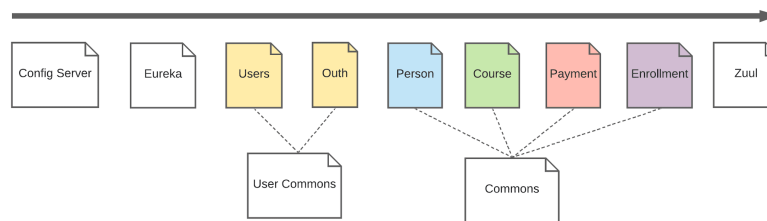
El sistema Matriculate YA!, está conformado por 6 microservicios de la BLL, 3 microservicios de la configuración de la arquitectura, sumando un total de 9 y adicional 2 librerías para los commons, siendo la puerta de enlace la api zuul server, a continuación tenemos una tabla con las configuraciones realizadas.

Api	Puertos	Accesos y registros	Base de datos
service-zuul-server	8090	service-eureka-server	-
service-eureka-server	8761	-	-
service-config-server	8888	Repositorio GitHub llamado service-escolastico-co nfig	-
service-oauth	9101	service-eureka-server service-users (FeignClient)	-
service-users	Aleatorio	service-eureka-server	PostgreSQL

		service-config-server	db_springboot_cloud_security
service-persons	Aleatorio	service-eureka-server service-config-server	MySQL db_springboot_cloud_person
service-enrollments	Aleatorio	service-eureka-server service-config-server (FeignClient)	PostgreSQL db_springboot_cloud_enrollm ent
service-payments	Aleatorio	service-eureka-server service-config-server	PostgreSQL db_springboot_cloud_payment
service-courses	Aleatorio	service-eureka-server service-config-server	Mysql db_springboot_cloud_course

Tabla 1. Configuración de los microservicios

A continuación se muestra el orden de arranque para los microservicios ya que como propuesta de trabajo futuro se plantea desplegar mediante el uso de contenedores.



Secuencia de arranque de los microservicios

Pasos para la creación de la arquitectura:

Crear repositorio de github con las configuraciones de los microservicios, que se identificarán a través del nombre y del flujo de trabajo en el que se encuentre siendo este “dev”. Cada archivo contiene las conexiones de los microservicios a las bases de datos además de información adicional para la autenticación

The screenshot shows a GitHub repository interface. At the top, there's a header with 'master' branch, '1 branch', and '0 tags'. Below this, the repository name 'lachaiza6 update person' is displayed with a commit hash 'a3906e8' and '2 minutes ago', along with '16 commits'. A table lists recent commits:

File	Commit Message	Time
application.properties	Add config to payment and enrollment	17 hours ago
service-courses-dev.properties	block generate table	2 hours ago
service-enrollments-dev.properties	Add config to payment and enrollment	17 hours ago
service-enrollments-prod.properties	Start configurations	4 days ago
service-enrollments.properties	Start configurations	4 days ago
service-payments-dev.properties	Add configuration rename	17 hours ago
service-persons-dev.properties	update person	2 minutes ago
service-users-dev.properties	Configuration Postgress Dialect refactor all	3 days ago

At the bottom, there's a prompt to 'Add a README'.

A continuación nos creamos los servicios con los nombres especificados en el repositorio, dichos servicios tendran un archivo llamado bootstrap.properties que se encargaran de apuntar al microservicios de configuraciones y obtener esta propiedades del repositorio github

```
1 spring.application.name= service-enrollments
2 spring.profiles.active = dev
3 spring.cloud.config.uri= http://localhost:8888
```

Antes de configurar los servicios y controladores correspondientes a cada microservicio, debemos crear los commons, que es realmente una librería que contiene todas las especificaciones de las clases para que el JPA de spring boot realice persistencia hacia la base de datos, para este caso tenemos dos commos uno para usuarios que contiene las entidades de User, Rol y UserRole. En la otra contamos con todas las clases para cada microservicio separadas por paquetes.

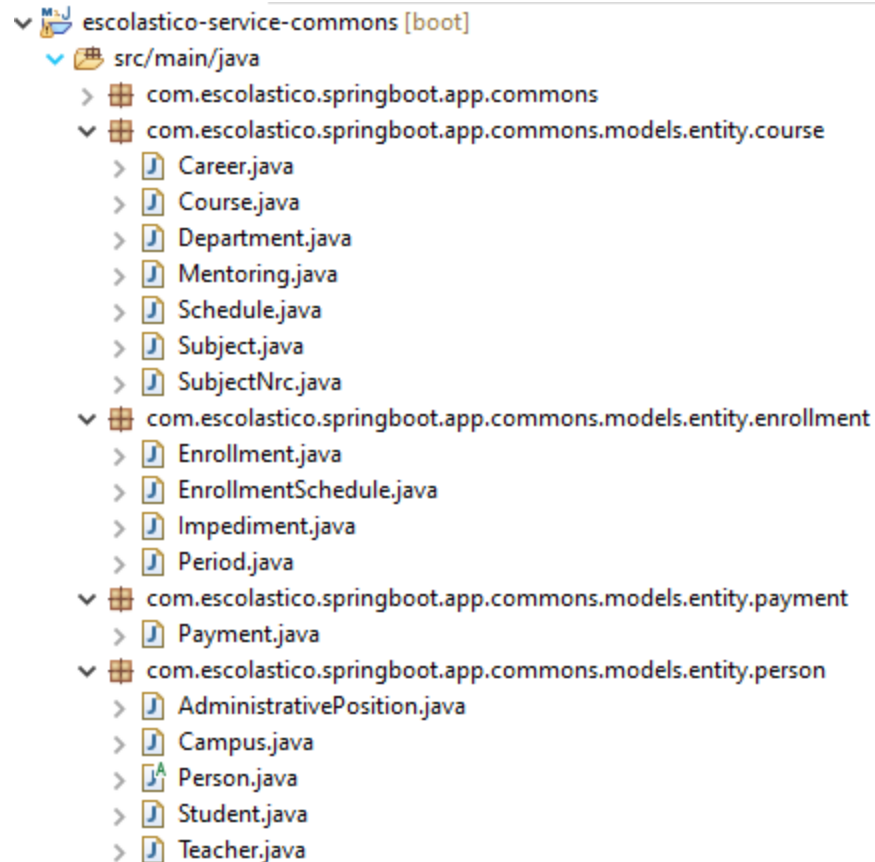


Imagen 2. Commons

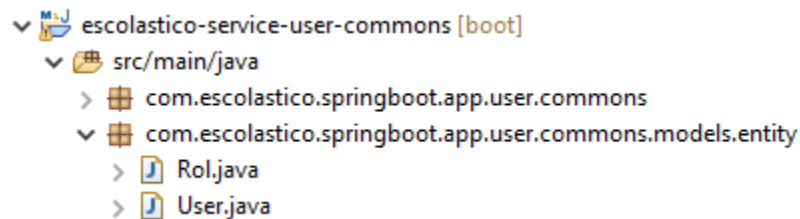


Imagen 3. User Commons

Para utilizar estos commons es necesario pensar en un ambiente de producción para lo cual accedemos por consola en cada uno y lo transformamos en una librería mediante el comando `mvnw clean install`, teniendo acceso al `HOME.JAVA` desde las variables de entorno.

Ya que tenemos la librería construida accedemos a ella mediante el `pow`

```
<dependency>
  <groupId>com.escolastico.springboot.app.user.commons</groupId>
  <artifactId>escolastico-service-user-commons</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Imagen 4. Dependencia para acceder a los commons de usuario

```

<dependency>
  <groupId>com.escolastico.springboot.app.commons</groupId>
  <artifactId>escolastico-service-commons</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>

```

Imagen 5. Dependencia para acceder a los commons

Al momento de declarar la variable de estas entidades direcciona directo al common correspondiente pero adicional a esto debemos escanear la ubicación específica de la clase y declararlo como configuración inicial con el Entity Scan.

```

package com.escolastico.springboot.app.persons;

import org.springframework.boot.SpringApplication;

@EnableEurekaClient
@SpringBootApplication
@EntityScan({"com.escolastico.springboot.app.commons.models.entity.person"})
public class EscolasticoServicePersonsApplication {

    public static void main(String[] args) {
        SpringApplication.run(EscolasticoServicePersonsApplication.class, args);
    }

}

```

Imagen 6. EntityScan para el microservicio personas

Utilizamos el Repository Rest Resource que se encarga de crear los servicios y controladores declarando el Repositorio en el Dao

```

package com.escolastico.springboot.app.enrollments.models.dao;












import org.springframework.data.repository.PagingAndSortingRepository;

@RepositoryRestResource(path="enrollments-repository")
public interface EnrollmentDao extends PagingAndSortingRepository<Enrollment, Long>{

}

```

Después de esto creamos las entidades de acceso para separar las relaciones dadas por el BLL, de las que identificamos cinco relaciones por resolver.

```
>  escolastico-service-commons [boot]
>  escolastico-service-config-server [boot] [devtools]
>  escolastico-service-courses [boot] [devtools]
>  escolastico-service-enrollments [boot] [devtools]
>  escolastico-service-eureka-server [boot] [devtools]
>  escolastico-service-oauth [boot] [devtools]
>  escolastico-service-payments [boot] [devtools]
>  > escolastico-service-persons [boot] [devtools] [EscolasticoMicroservicios feature_armando ↓1]
>  escolastico-service-user-commons [boot]
>  escolastico-service-users [boot] [devtools]
>  escolastico-service-zuul-server [boot] [devtools]
```

El servicio commons tiene librerías para las clases, servicio y controladores que se implementan de manera dinámica a través de interfaces.

El servicios config server se comunica directamente al repositorio github y funciona en el puerto 8888 para que los demás servicios accedan a sus configuraciones necesitan de bootstrap properties.

El servicio courses contiene la lógica para acceder a los cursos materias y materias nrc.

El servicio enrollments contiene la lógica para realizar el módulo de matriculación y accede a los usuarios, cursos y pagos,

El servicio eureka-server como su nombre lo indica contiene el eureka activado como servidor a donde apuntan todos los demás microservicios a excepción de los commons.

El servicio outh tiene configurado el sistema de ingreso por outh 2, crea y vincula los tokens con la información de los usuarios a trabajar como puerta de seguridad.

El servicio paymments contiene la lógica para acceder a los pagos.

EL servicio persons contiene toda la lógica para acceder a los estudiantes y profesores.


El servicio user commons tiene librerías para las clases de usuario que se implementa en el servicio de users

El servicio user contiene la lógica para acceder a los usuarios y roles.


EL servicio zuul-server como su nombre lo indica contiene el zuul server activado, además accede a los microservicios a través de su tag, es la puerta de redireccionamiento que gestiona de manera balanceada.

Interfaz

000-099-252-0223
Ingresar


Matriculate-Aquisito!
MATRICULACION EN LINEA

Matriculación
Horario



Ingreso o registro
Nombre de usuario*

Contraseña*

 ☐ Recuerdame
Perdiste tu contraseña?
No tiene una cuenta?Regístrate ahora

Main Menu
Home
Our Services

Knowledge Base
Delivery
Returns

Useful Links
Site Map
Search

Contact Us
25 Astor Pl, NY 10003, USA
+1 212-982-4589

Contiene el inicio de sesión que accede al servicio outh a través de zuul para lo cual se requiere una clave de acceso.

```

/**
 * signIn
 */
public signIn(loginUser: LoginUser): Observable<JwtDto> {
    let body = new URLSearchParams();
    body.set('username', loginUser.userName);
    body.set('password', loginUser.password);
    body.set('grant_type', 'password')
    console.log(body);
    console.log(loginUser);
    // { 'username': loginUser.userName, 'password': loginUser.password, 'grant_type': 'password' }
    return this.httpClient.post<JwtDto>(this.authUrl + '/oauth/token', body.toString(), this.httpOptions);
}

```

```

httpOptions = {
    headers: new HttpHeaders({
        'Content-Type': 'application/x-www-form-urlencoded',
        Accept: 'application/json',
        'Authorization': 'Basic ' + btoa(`${environment.username}:${environment.password}`)
    })
};

```

000-099-262-0223
Cuenta
Salir

Matriculate-Aquisito!
MATRICULACIÓN EN LÍNEA

Pagos

Matriculación
Horario

BUSCAR MATERIA POR SU NRC

NRC	Cupos	Materia	Docente	Matricularme
Next »				

DATOS

ID	Nivel	Tipo	Nombre	Celular	Cédula
----	-------	------	--------	---------	--------

HORARIO

Es la pantalla principal que permite buscar materias por sus nrc, en la parte inferior se podrá ver el horario y los datos del estudiante

```

public getEnrollmentsByIdStudent(id: number): Observable<Enrollment[]> {
  return this.httpClient.get<Enrollment[]>(this.authUrl + '/enrollment-repository/enrollment/student/search/idStudent/' + id, this.httpOptions);
}

public deleteByIdEnrollmentAndIdStudent(idEnrollment: number): Observable<any> {
  return this.httpClient.delete<any>(this.authUrl + '/enrollment-repository/enrollment/student/delete/' + idEnrollment, this.httpOptions);
}

public createByIdEnrollmentAndIdStudent(idEnrollment: number, idStudent: number): Observable<Enrollment> {
  return this.httpClient.get<Enrollment>(this.authUrl + '/enrollment-repository/enrollment/student/create/' + idEnrollment + '/' + idStudent, this.httpOptions);
}

```

```

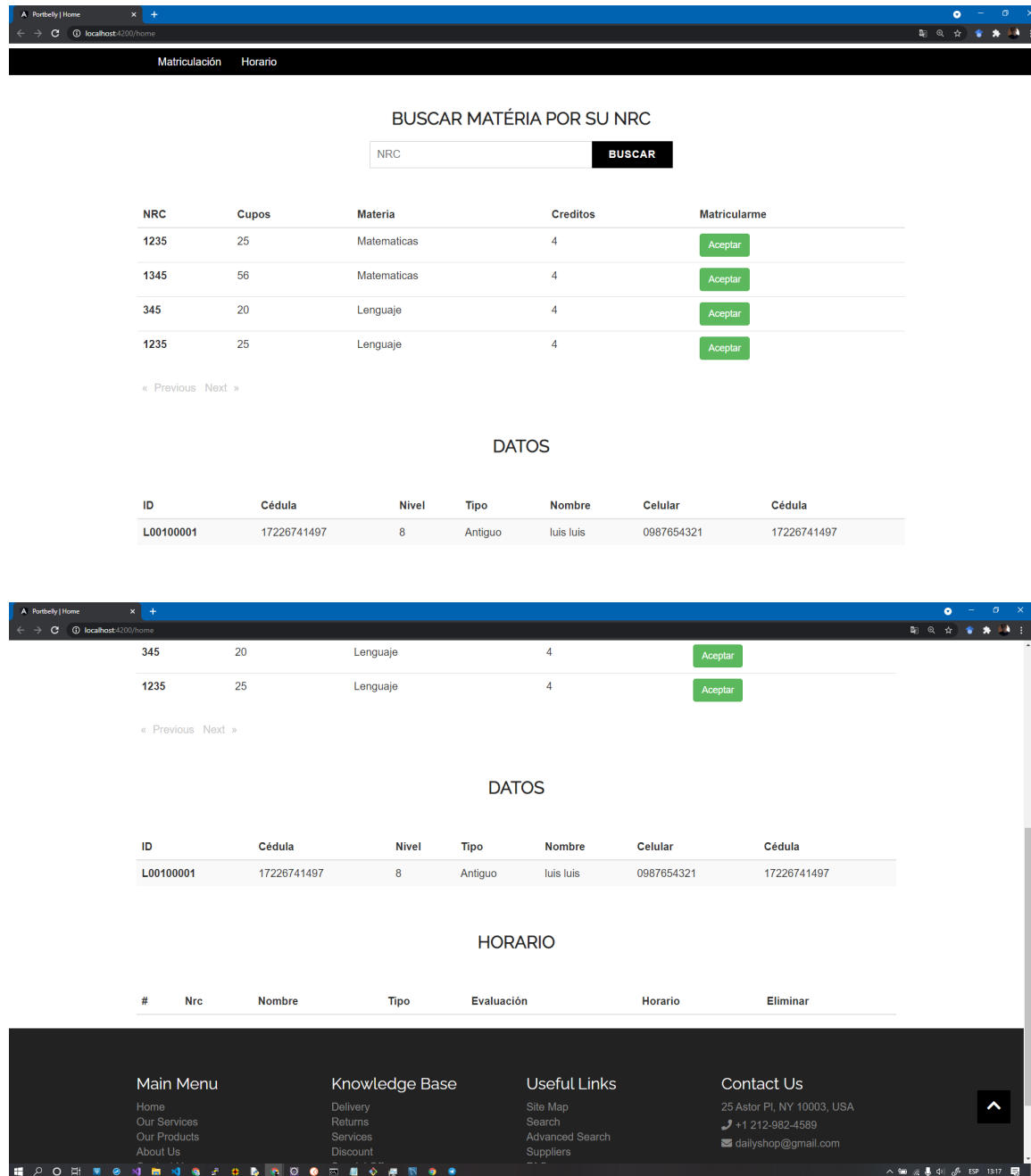
public getStudentByIdUser(id: number): Observable<Student> {
  return this.httpClient.get<Student>(this.authUrl + '/enrollment-repository/student/search/idUser/' + id, this.httpOptions);
}

```

```

public getAllSubjectsNrc(): Observable<SubjectNrc[]> {
  return this.httpClient.get<SubjectNrc[]>(this.authUrl + '/enrollment-repository/subjectnrc/all/', this.httpOptions);
}

```



Conclusión

Los microservicios ayudan a la escalabilidad y flexibilidad de los sistemas, ya que al estar cada uno instanciado de manera independiente reduce el riesgo de que si uno falla los demás pueden seguir trabajando, además pueden contener bases de datos independientes vinculadas por tablas intermedias, Spring boot ayuda con la creación de estos microservicios ya que contiene todo lo necesario para establecer comunicaciones entre ellos con configuración desacoplada. Adicionales esto la utilización de commons ayuda a tener todas las clases más organizadas, estas clases también implementan servicios y controladores a través de interfaces resulta en beneficio al momento de completar los servicios Rest. Como trabajo adicional se incorporó el servicio Zipkin

para la gestión de la trazabilidad donde aunque no se detalló de manera exhaustiva si sirvió para dejar el granito de curiosidad y utilizarlo para proyectos posteriores, esta herramienta es el receptor y los microservicios son los emisores. Para capturar estas trazas se incorporó el servicio RabbitMQ que es un intermediario por donde pasarán todos los mensajes y los guardará en una base de datos seleccionada en este caso es MySQL.

Recomendaciones

Para una mejor gestión de los microservicios que contiene distintas bases de datos es necesario diseñará las relaciones de manera correcta es decir entre entidades separadas compartirán una tabla intermedia de identificadores que aunque no está ligada concretamente se utiliza para referenciar a las demás clases, en cuestión de las configuraciones de la arquitectura se plantea la utilización del diseño de maquetación o diseño de arquitectura donde se visualizan lo componentes a un alto nivel y sus conexiones que serán tu utilidad el el servicio de eureka, manejar puertos dinámicos y latencia de carga.

Bibliografía

Nebel, A. (2019). Arquitectura de microservicios para plataformas de integración.

Sucasas, V., Mantas, G., Radwan, A., & Rodriguez, J. (2016, May). An OAuth2-based protocol with strong user privacy preservation for smart city mobile e-Health apps. In 2016 IEEE International Conference on Communications (ICC) (pp. 1-6). IEEE.

Siriwardena, P. (2020). Edge Security with an API Gateway. In Advanced API Security (pp. 103-127). Apress, Berkeley, CA.

Molchanov, H., & Zhmaiev, A. (2018). Circuit breaker in systems based on microservices architecture. Advanced Information Systems, 2(4), 74-77.

Hoffmann, T. R. (1992). EUREKA: A hybrid system for assembly line balancing. Management Science, 38(1), 39-47.

PostgreSQL, B. (1996). PostgreSQL. Web resource: [http://www. PostgreSQL. org/about](http://www.PostgreSQL.org/about).

MySQL, A. B. (2001). MySQL.

Zipkin (2015) from <https://zipkin.io/>

Anexos

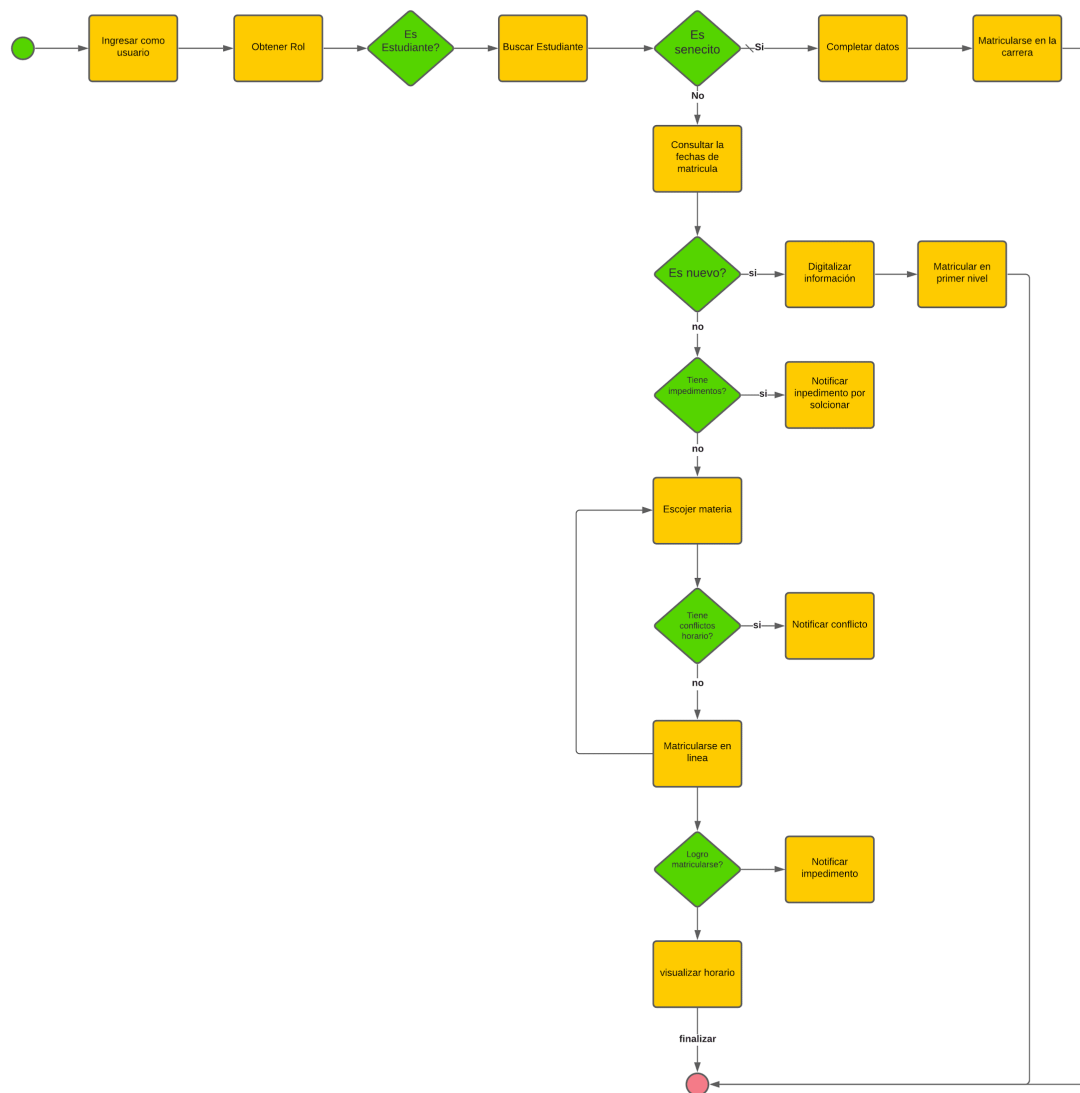


Imagen. Diagrama de Secuencia