Certified Tech Developer
The Ultimate Degree

Especialización en Back End I

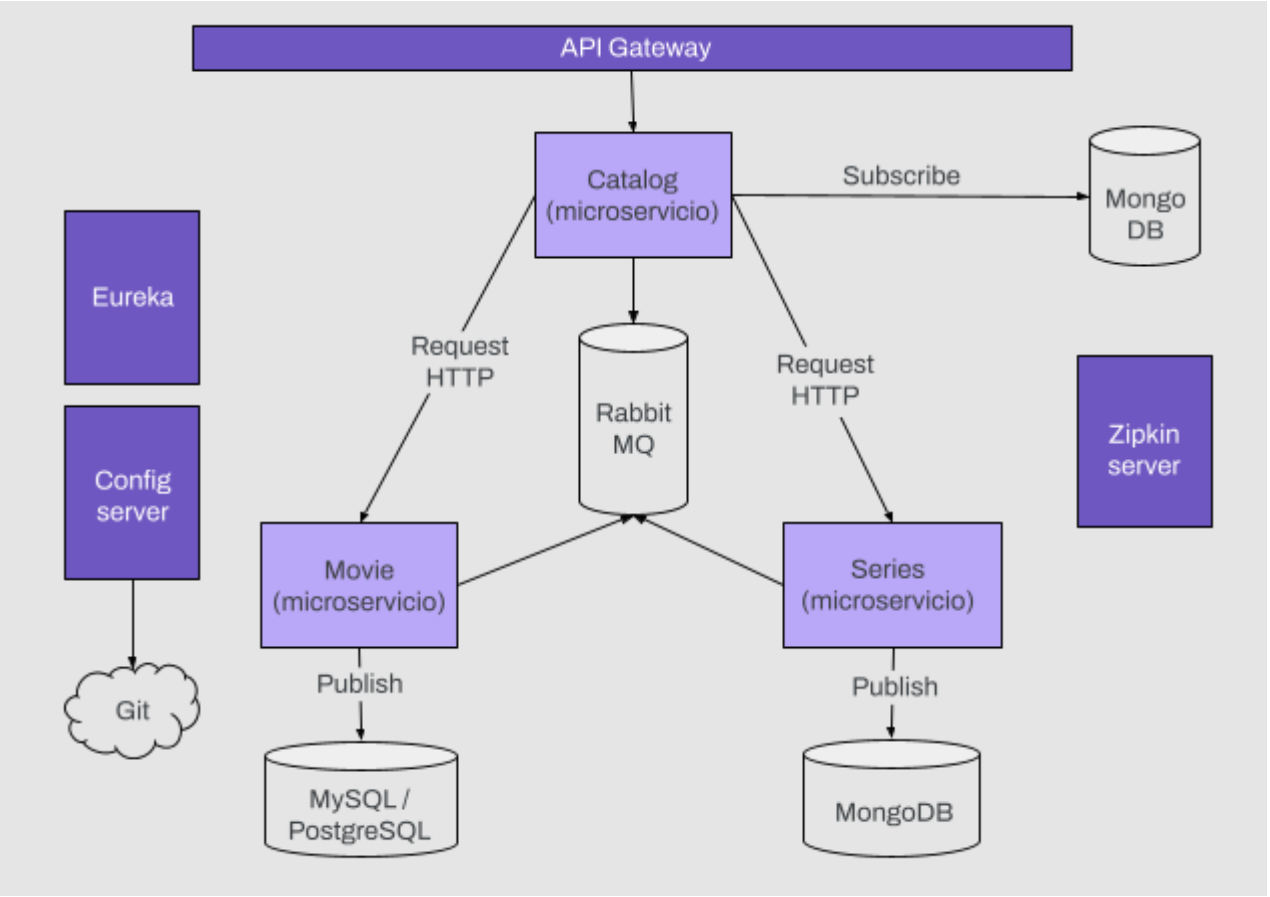
Examen final

Esta evaluación será distinta a la anterior, ya que la construiremos durante las clases siguientes. Esto no quiere decir que lo resolvamos durante los encuentros en vivo, pero con los temas que iremos abordando podrán completar los requisitos. ¡Éxitos!

Contextualización

El proyecto consiste de tres microservicios: **Movie**, **Series** y **Catalog**. **Catalog** es un microservicio que lee información de **Movie** y **Series** con el objetivo de enviar un catálogo al cliente. **Catalog** recibe un mensaje cada vez que una película o una serie es dada de alta y las persiste en una base de datos no relacional de MongoDB. Cuando le llega una petición del cliente, busca en la base de datos y responde.

Veamos un diagrama base de los microservicios:



A continuación, veremos la información detallada de los microservicios.

movie-service

El microservicio gestiona las operaciones sobre las películas. Cada película tiene como atributo:

- id
- name
- genre
- urlStream

serie-service

El microservicio gestiona las operaciones sobre las series. Cada serie tiene como atributos:

- id
- name
- genre

- seasons
    - id
    - seasonNumber
    - chapters
        - id
        - name
        - number
        - urlStream

catalog-service

El microservicio tiene como objetivo invocar a los microservicios **Movies** y **Series**. Estos microservicios deben ser invocados cada vez que se carga una nueva película o serie y se debe persistir la información que proporcionan ambos microservicios en una base de datos no relacional de MongoDB con la siguiente estructura:

- genre
    - movies
        - id
        - name
        - genre
        - urlStream
    - series
        - id
        - name
        - genre
        - seasons
            - id
            - seasonNumber
            - chapters
                - id
                - name
                - number
                - urlStream

Consigna

serie-service

- Crear microservicio serie.
- Configurar Eureka para el nuevo servicio y utilizar el nombre: **serie-service**.
- Configurar el ruteo en el gateway para el nuevo servicio y agregar seguridad con OAuth.
- Configurar Server config para obtener la configuración desde un repositorio Git.
- Crear API que nos permita:
    - Obtener un listado de series por género. Endpoint: **/series/{genre}** [GET]
    - Agregar una nueva serie. Endpoint: **/series** [POST]
- Persistencia: agregar la dependencia e implementar MongoRepository para persistir las series.
- Agregar RabbitMQ y enviar un mensaje en el momento que se agregue una nueva serie.

movie-service

- Agregar persistencia: utilizar MySQL para persistir las películas.
- Configurar el ruteo en el gateway para agregar seguridad con OAuth.
- Agregar RabbitMQ y enviar un mensaje en el momento que se agregue una nueva película.

catalog-service

- Actualizar el catálogo utilizando **Feign** de manera de agregar a este servicio la búsqueda de las series por género (serie-service) y agregarlas a la respuesta del endpoint **/catalog/{genre}**.
- Agregar persistencia: luego de obtener las películas y las series según género, persistir las mismas en MongoDB.

- Agregar RabbitMQ y escuchar los mensajes que envían movie-service y serie-service. En caso de recibir un mensaje de algún servicio, actualizar el listado correspondiente, ya sea las películas o las series.

Spring Cloud: traceo utilizando Zipkin

- Crear proyecto y configurar **Zipkin server** para recibir los mensajes de los microservicios. Agregar **Zipkin UI** para visualizar las trazas.
- Configurar Zipkin en cada microservicio.
- Visualizar las comunicaciones entre los microservicios desde la interfaz que nos da **Zipkin UI**.
- Deployment: todos los microservicios deberán deployarse en dockers.

Resiliencia - Resilence4J

- Del proyecto anterior, se debe seleccionar uno de los servicios (preferentemente el que consideres que será más utilizado) y adaptarlo para que el mismo sea tolerante a fallos.
- Para lo anterior deberás:
    - Definir esquema de resiliencia. Por ejemplo: doble redundancia, retry and fallback, balanceo de carga, tiempos de warm up, reglas de circuito.
    - Modificar el código de tu proyecto —aplicando alguna de las tres tecnologías mencionadas— para que dentro del servicio seleccionado se aplique el esquema definido.
- Como mínimo, el servicio deberá contar con:
    - Doble redundancia.
    - Reglas del circuito (se puede crear un servicio que devuelva activo/inactivo en función de la memoria disponible, uso del procesador, exceptions).
    - Descripción de la solución de redundancia, justificación (un comentario en el código).