UNIVERSIDAD CATÓLICA DEL MAULE
Facultad de Ciencias de la Ingeniería
Escuela de Ingeniería Civil Informática

Profesor Guía
Dr. Marco Toranzo Céspedes

PROPUESTA DE UN SISTEMA DE INFORMACIÓN INTEGRADO PARA LA FACULTAD DE CIENCIAS DE LA INGENIERÍA DE LA UNIVERSIDAD CATÓLICA DEL MAULE

CLAUDIO IGNACIO DURÁN NÚÑEZ
MARTÍN IGNACIO MANCILLA VILLASECA

Tesis para optar al
Título Profesional de Ingeniero Civil Informático

Talca, Enero 2021

UNIVERSIDAD CATÓLICA DEL MAULE

FACULTAD DE CIENCIAS DE LA INGENIERÍA

ESCUELA DE INGENIERÍA CIVIL INFORMÁTICA

TESIS PARA OPTAR AL

TÍTULO PROFESIONAL DE INGENIERO CIVIL INFORMÁTICO

PROPUESTA DE UN SISTEMA DE INFORMACIÓN INTEGRADO PARA LA FACULTAD DE CIENCIAS DE LA INGENIERÍA DE LA UNIVERSIDAD CATÓLICA DEL MAULE

CLAUDIO IGNACIO DURÁN NÚÑEZ
MARTÍN IGNACIO MANCILLA VILLASECA

COMISIÓN EXAMINADORA	**FIRMA**
PROFESOR GUÍA Dr. Marco Toranzo Céspedes	_____
PROFESOR COMISIÓN Dr. Paulo González Gutiérrez	_____
PROFESOR COMISIÓN Mg. Roberto Ahumada García	_____
NOTA FINAL EXAMEN DE TÍTULO	_____

TALCA, ENERO DE 2021.

Claudio Ignacio Durán Núñez:

A mis abuelos por inculcar en mí las ganas de estudiar una carrera universitaria.

A mis padres, hermanas y perritos por el apoyo y cariño otorgado.

A mis amigos, los cuales fueron un pilar fundamental.

A todos, muchas gracias.

Martín Mancilla Villaseca:

A mis padres y hermanos que fueron fundamentales para mis ganas de salir adelante.

A mis tíos y abuelos que siempre creyeron en mi.

A mis amistades y cercanos, que me alegraron el día a día.

A mis chinchillas, por esperarme.

Sumario

En la actualidad los microservicios son esenciales a la hora de entregar servicios a gran escala, ya que ofrecen una estabilidad y seguridad mayor por sobre los sistemas monolíticos, impidiendo la interrupción de los servicios que componen un aplicativo, así lo han demostrado grandes empresas impulsoras y líderes en utilización de microservicios, como lo son Netflix, Uber y Amazon.

Gracias al trabajo comunitario en el desarrollo de nuevas tecnologías Open Source, es posible realizar proyectos profesionales y aplicables a la actualidad laboral. Esto, acompañado de una documentación completa y gran cantidad de aportes de la comunidad a la mantención de los proyectos, asegurando su seguridad y adaptación a los constantes cambios que ocurren en la industria. Además, se cuenta con la entrega de herramientas necesarias para la comunicación de los integrantes del equipo de trabajo, mas aun dada las condiciones que se han presentado durante el último tiempo a nivel mundial.

La tesis presentada tiene como objetivo, diseñar, desarrollar y aplicar un proyecto base con una arquitectura no monolítica, basada en microservicios. Con esto, se busca ayudar y facilitar la gestión administrativa por parte de los funcionarios de la Facultad de Ciencias de la Ingeniería, haciendo uso de los datos obtenidos a través de un web Scraper, desarrollado en Node.js con la ayuda de la biblioteca Puppeteer, de las plataformas oficiales de la universidad. Además, consta con un frontend, desarrollado en Angular, de carácter modular, permitiendo la integración de nuevas funcionalidades sin interrumpir las ya implementadas. La modularidad igualmente abarca el backend de la aplicación, en donde se implementa una arquitectura basada en microservicios, desarrollada en Java con la ayuda del framework Spring y el módulo de desarrollo para aplicaciones en la nube, llamado Spring Cloud Netflix, que es desarrollado, mantenido y utilizado por Netflix en su servicio de streaming a nivel mundial. El desarrollo modular de las distintas capas generadas para el ecosistema permita agregar nuevas funcionalidades sin generar complicaciones con los módulos ya implementados.

Finalmente, la tesis concluye con el análisis de los resultados obtenidos tras el desarrollo de la misma, obteniendo así un sistema completo, seguro y modular, utilizando una arquitectura de microservicios, arquitectura utilizada por las mayores empresas tecnológicas del mundo. La tesis fue pensada desde un inicio para la integración sencilla y directa de nuevas funcionalidades, sin presentar mayores complicaciones. De este modo, los estudiantes de la carrera de Ingeniería Civil Informática son facilitados para la integración y desarrollo de nuevos módulos siguiendo los lineamientos definidos en la base del proyecto.

Índice General

Índice de Figuras

1 Introducción

1.1 Descripción del Problema

En la actualidad la Universidad Católica del Maule (UCM), y en específico la Facultad de Ciencias de la Ingeniería (FCI), realiza una gran cantidad de procedimientos de manera manual, lo que se refleja en un malgasto de recursos, como el papel, tinta, entre otros, así como la necesidad de transportar la información de manera física, acción que demora el proceso de la resolución final. Así mismo, al no encontrar la información y solicitud de manera digital, es necesario recabar información para cada una de las peticiones, lo que aumenta el tiempo de respuesta. A su vez, los sistemas pertenecientes a la universidad utilizan una arquitectura monolítica que posibilita, en caso de error, que se detenga por completo el sistema. En términos generales, la universidad no cuenta con un sistema que permita realizar tareas que pueden ser automatizadas.

1.2 Solución

Se plantea la creación de un sistema de información basado en microservicios, el cual busca la integración de la información obtenida de los distintos portales de la universidad, con el objetivo de ayudar a la toma de decisiones. El sistema base propuesto permite agregar nuevos módulos con una configuración necesaria mínima, así como generar un ahorro significativo en cuanto a recursos y optimización del tiempo.

1.3 Objetivos

El desarrollo del sistema planteado tiene los siguientes objetivos:

- Objetivo General
  - Diseñar un sistema de información para apoyar la gestión administrativa de la Facultad de Ciencias de la Ingeniería de la Universidad Católica del Maule.
- Objetivos Específicos
  - Aplicar la ingeniería de requerimientos en la recopilación de los requerimientos de información de la facultad.

  - Analizar los diferentes procesos a emplear en el diseño del sistema de información.
  - Analizar las diferentes tecnologías a emplear en el desarrollo del sistema de información.

1.4 Contribuciones del sistema

El sistema a desarrollar pretende entregar las siguientes contribuciones:

- **Automatización de Procesos:** permite la generación de módulos para automatizar procesos establecidos.
- **Base para Desarrollar**: se entrega un ecosistema base para el desarrollo de nuevos proyectos.
- **Base de Datos**: el ecosistema dispone de una base de datos que se carga con cada usuario que utilice el sistema.
- **Mínima Configuración:** permite agregar nuevos módulos utilizando una configuración mínima.
- **Tolerancia a Fallos:** en caso de fallo solo se detiene el módulo con deficiencias y no el sistema completo, a menos que sea un módulo crítico que interrumpa el correcto funcionamiento del ecosistema.

1.5 Estructura de la tesis

Lo que resta del documento se divide en los siguientes capítulos:

- **Capítulo 2:** Muestra del estado actual de los microservicios en el mundo, su impacto en el ámbito empresarial y su aporte en la facilitación y soporte a gran escala de los servicios.
- **Capítulo 3:** Se presentan las distintas tecnologías utilizadas durante el desarrollo y gestión del proyecto.

- **Capítulo 4:** Se detalla el proceso iterativo e incremental utilizado para desarrollar el software.
- **Capítulo 5:** Se detalla el resultado final obtenido.
- **Capítulo 6:** Se presentan las conclusiones, aspectos a mejorar y trabajos futuros.

2 Estado del Arte

2.1 Microservicios

Para el año 2020, se espera que el mercado global en la nube en base a microservicios crezca en alrededor de un 22.5%, con la proyección del mercado norte americano en una crecida estable de un 27,4% [GLO2020], con esto, la tendencia de los desarrolladores de levantar proyectos centrados en un punto de levantamiento decrementará.
En base a lo anterior, se presentan casos de éxito de empresas que se han cambiado desde sus sistemas monolíticos a microservicios, para la entrega de sus servicios.

2.1.1 Amazon

Para el 2001, los atrasos en el desarrollo, dificultades en la programación, e ínter dependencias, no permitieron a Amazon controlar los requerimientos de su creciente de su clientela. Con esto nació la necesidad de realizar un *refactor* a su arquitectura monolítica desde cero, separando sus aplicaciones centralizadas en un sistema, en variados sistemas pequeños, independientes y específicos a una aplicación.
En el mismo año, Amazon decidió realizar el cambio a microservicios, lo cual fue años antes a que su implementación se convirtiese en la norma. Este cambio llevó a Amazon a desarrollar variadas soluciones para apoyar las arquitecturas de microservicios, tales como Amazon AWS. Con el rápido crecimiento y adaptación a microservicios, Amazon se convirtió en la compañía mas valorada mundialmente, valorado con un precio de mercado de $1.433 trillones para el año 2020[YCH2020].

2.1.2 Netflix

Netflix comenzó su sistema de *streaming* en el año 2007, y para el 2008, estaba sufriendo desafíos de escala. Sufrieron una grave corrupción a su base de datos, por lo que por tres días no fue posible realizar envíos de DVDs a sus clientes [10], Este fue el punto de inicio para que Netflix viera la necesidad de cambiar de sus sistemas de un sólo punto de falla (Como lo son las bases de datos relacionales), a sistemas distribuidos escalables y mas estables en la nube. En el año 2009, Netflix comenzó su proceso de *refactor* a sus

sistemas monolíticos a microservicios. Comenzaron migrando sus sistemas no directamente relacionados con sus clientes, como lo son los de codificación de sus películas, para que corrieran en sistemas en la nube como microservicios independientes[H2020]. Este cambio permitió a Netflix sobrellevar sus desafíos de escala y así poder realizar mas de 250 horas de contenido de streaming para mas de 139 millones de usuarios. Netflix abrió sus soluciones a modo de código abierto para la implementación de microservicios, como lo son el Gateway Zuul y el servidor Eureka.

2.1.3 Uber

Después de su lanzamiento, la empresa se vió con dificultad a la hora de desarrollar nuevas características, arreglar bugs y la integración rápida de nuevos cambios. Dado esto, Uber decidió realizar el cambio a microservicios, rompiendo la estructura de aplicación a microservicios en la nube. En otras palabras, Uber creó un micro servicio para cada función, tales como el control de pasajeros y de tripulantes. El cambio a microservicios trajo a la empresa variados beneficios, tales como tener una clara idea de la pertenencia de cada servicio, incrementando la velocidad y calidad de su escalada, permitiendo a sus equipo centrarse únicamente en los servicios que requieran escalada, así actualizando servicios virtuales sin interrumpir al resto, consiguiendo una mayor tolerancia al fallo [H2020].

3 Marco Teórico

3.1 Microservicios

Un microservicio es un proyecto con alcances menores a los de un servicio normal, cumpliendo funcionalidades especificas dentro de una aplicación, es independiente y puede comunicarse con otros microservicios, lo que se conoce como arquitectura de microservicios.

Una arquitectura basada en microservicios se utiliza para desarrollar aplicaciones de software y se basa en la generación de un sistema a partir de diversos servicios independientes y autónomos que se comunican entre sí, cada uno de estos servicios es llamado microservicio.

Esta arquitectura al basarse en varios sistemas pequeños permite el desarrollo de manera simultánea, debido a la independencia que estos presentan. A su vez, el desarrollo de un microservicio puede ser llevado por un equipo pequeño, al contar con una cantidad de funcionalidades específicas. En base a la independencia de cada uno de los microservicios, esta arquitectura permite la expansión del sistema, añadiendo nuevos microservicios o actualizando los ya existentes, sin la necesidad de recompilar o detener el sistema por completo.

Como cada uno de los microservicios que se generan son independientes entre sí, pueden ser desarrollados en distintos lenguajes de programación, lo esencial es la utilización de protocolos de comunicación que sean comunes para los sistemas, como puede ser las peticiones HTTP consumiendo las API que se generaron en los microservicios.

Grandes empresas de diversas industrias han optado por la evolución de las aplicaciones monolíticas hacia los microservicios debido a la gran cantidad de beneficios que ofrecen, entre los que se destaca el mantenimiento, la escalabilidad y el manejo de concurrencia, permitiendo levantar distintas instancias de un mismo microservicio si se estima pertinente. Algunas empresas que han optado realizar la migración a microservicios son: Netflix, Amazon, Ebay, entre otros. [HAT2019]

3.2 PostgreSQL

PostgreSQL, también conocido como Postgres, es un gestor de base de datos relacionales gratuito y de código libre. Es mantenido por la comunidad que se denomina PGDG (PostgreSQL Global Development Group). Fue desarrollado en 1996 y está escrito en lenguaje C. Postgres, permite la gestión de la base de datos de manera intuitiva debido al aplicativo web que incluye, llamado pgAdmin, en donde se pueden realizar todas las interacciones posibles en una base de datos, entre los que se incluye la visualización de información, ejecución de scripts, entre otros. Postgres se destaca entre sus competidores por diversos motivos, entre los que se encuentran: gran escalabilidad, estabilidad, confianza, potencia y robustez. PostgreSQL cumple con las características ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad por sus siglas en inglés) lo que garantiza que la información perdure en el sistema en caso de fallos. [POS2020]

3.3 Spring

Spring es un framework de desarrollo web para Java. Creado en el año 2002 bajo licencia de código abierto. Está escrito en Java y se ejecuta bajo la plataforma Java EE (Java Platform, Enterprise Edition). Entre sus características se destaca la creación de código de alto rendimiento, liviano y reutilizable. Spring no necesita una exhaustiva configuración para desarrollar un sistema, es más, la mayoría de los proyectos pueden ser desarrollados y producidos con la configuración inicial por defecto que entrega el framework.

Spring utiliza una arquitectura dividida en módulos, lo que permite agregar los que sean necesarios a la aplicación que está en desarrollo. Cada uno de los módulos tiene como objetivo el cumplimiento de una tarea específica, como puede ser la integración de alguna herramienta de desarrollo o la conexión a un motor de base de datos. Spring permite la integración para desarrollar mediante el patrón MVC (Modelo-vista-controlador), así como la utilización para creación de un backend en donde se generan distintos endpoint para la comunicación, este caso es el principal uso que se le asigna a Spring. [SPR2020b]

3.4 Spring Cloud Netflix

Spring Cloud Netflix es un submódulo perteneciente al módulo Spring Cloud. Spring Cloud fue diseñado para el trabajo con sistemas distribuidos, permitiendo un rápido desarrollo y configuración de los entornos. Spring Cloud Netflix se divide en una serie de proyectos de código abierto creados, desarrollados y mantenidos por Netflix, los cuales utilizan para el funcionamiento de su servicio de streaming, el que está basado en arquitectura de microservicios. Este submódulo entrega una sencilla integración de las herramientas utilizadas por Netflix al ecosistema Spring, todo esto, mediante una serie de configuraciones en base a anotaciones y vinculación de variables de entorno.
La implementación de este submódulo permite generar un ecosistema de microservicios sin la necesidad de una exhaustiva configuración y el respaldo de una de las compañías pioneras en arquitectura de microservicios.[SPR2020a]

3.4.1 Eureka Server

Desarrollado por Netflix, el servidor Eureka es el núcleo principal para la conexión entre microservicios, el cual de por si es un servicio a nivel local que permite el descubrimiento y consulta constante de los microservicios, eliminando la necesidad de ingresar en bruto el hostname y el puerto de los servicios, en este caso, los microservicios.
Cada microservicio al conectarse al ecosistema envía a Eureka su nombre y dirección física (IP y puerto), con esto queda disponible para ser consultado por los demás servicios.
Cada uno de los microservicios que se encuentren en el ecosistema envían cada 30 segundos, por defecto, una señal a Eureka, conocida como heartbeat, lo que permite saber que se encuentran disponibles. En caso que no se reciban 3 señales consecutivas, se da de baja el servicio ya que no se encuentra disponible para realizar consultas. Si la instancia del microservicio se vuelve a levantar, se necesitan 3 señales consecutivas (3 heartbear) para que esté disponible nuevamente.
Cuando se utiliza junto a Zuul, permite el reporte de la ubicación de los servicios registrados para eliminar la necesidad de los aplicativos frontend de conocer la dirección y puerto de los servicios.

3.4.2 Zuul

Zuul es un proyecto código abierto de Netflix, que actúa como router y servidor para el balanceo de carga, basado en JVM. Con esto Zuul sirve como puerta de entrada para todas las solicitudes de dispositivos y sitios web al backend o servicios conectados al servidor, permitiendo el redireccionamiento dinámico, monitoreo, resiliencia y seguridad.
De querer agregar cierta lógica previo o posterior a una solicitud, Zuul permite el uso de filtros para realizar acciones por sobre las peticiones durante su ciclo, ya sean validaciones de roles de usuario, o requerimientos para la cumplimiento de la petición.
En conjunto a Eureka, elimina la necesidad de los aplicativos a saber la ubicación exacta de los servicios con los que Zuul conecte, ya que solo es necesario realizar la petición de forma directa a Zuul, para luego Zuul consultar le estado y la ubicación con Eureka.

3.5 Angular

Angular es un framework de código abierto desarrollado por Google en el año 2010. Actualmente es mantenido por una gran cantidad de personas, entre los que se encuentran principalmente desarrolladores de Google. Angular permite el desarrollo de aplicaciones web y móviles multiplataforma. Está escrito en TypeScript, que es un lenguaje de programación desarrollado y mantenido por Microsoft.
Angular es utilizado en el desarrollo profesional y empresarial debido a su flexibilidad para ser modificado y el cumplimiento de estándares modernos de seguridad. Angular trabaja en base a web components, en donde se generan pequeñas partes de la vista que pueden ser reutilizadas donde se requiera. Cada web component está conformado por 3 elementos; una vista escrita en HTML, lógica del funcionamiento del component escrita en TypeScript (TS) y reglas de estilos escrita en CSS, o en algún preprocesador de CSS como puede ser: Scss, Sass, Less o Stylus.
Un proyecto en Angular, idealmente, se encuentra organizado en módulos, lo que permite separar funcionalidades y reutilizar secciones de código. Cada módulo cumple una función específica y puede ser mantenido por distintos equipos de trabajo. [BOR2018]

3.6 Node.js

Node.js es un manejador de eventos de JavaScript asíncrono y de código abierto que permite la ejecución de código JavaScript en un backend, es decir, fuera del navegador. Fue desarrollado en el año 2009, está escrito en C, C++ y JavaScript y se basa en el motor de JavaScript V8 desarrollado por Google. [NOD2020]

3.6.1 Express.js

Express.js, o también llamado Express, es un framework de desarrollo para NodeJs rápido, robusto y minimalista, creado en 2010 bajo código abierto y escrito en JavaScript, es flexible y provee características para desarrollo de aplicaciones web y móviles.
En conjunto con Node.js se utilizan para generar aplicaciones backend robustas, seguras y rápidas. Ambas tecnologías ofrecen una gran facilidad de desarrollo debido a la utilización de JavaScript como lenguaje de programación. [DOC2020]

3.6.2 Puppeteer

Puppeteer es una librería de Node.js que provee una API de alto nivel para controlar Chrome o Chromium sobre el protocolo DevTools. La librería es de código abierto, fue creada en el año 2017 y está escrita en TypeScript y JavaScript. Puppeteer ejecutar un navegador en modo headless (en segundo plano) por defecto, pero puede ser configurado para que se ejecute de manera non-headless. Además, permite la captura de screenshots, generación de PDFs de las páginas, automatización de envío de formularios, pruebas de UI (user interface), entradas de teclado, entre otros. El principal uso de esta librería es para la automatización de tareas y testeo de funcionalidades, teniendo un rol importante en QA (Quality Assurance). [NPM2020]

3.7 Postman

Postman API Client, como su nombre lo indica, es un cliente de API. Se utiliza para testear API de distintos tipos, como son, REST, SOAP, entre otros. Su utilización permite la verificación y validación de API sin la necesidad de tener el cliente creado para consumir la funcionalidad. Es ampliamente utilizado en el mundo del desarrollo de software debido

a que ayuda a optimizar el tiempo de desarrollo por la facilidad que entrega de poder trabajar funcionalidades de manera asíncrona, sin depender de la funcionalidad completa del sistema, ya que permite el testeo de manera individual. [POS2019]

3.8 Git

Git es sistema de control de versiones open source diseñado para almacenar proyectos de todas proporciones con rapidez y eficiencia. Desarrollado en el año 2005 por Linus Torvalds, creador de linux, para el desarrollo del kernel del mismo. Actualmente es mantenido por Junio Hamano y otros desarrolladores de software. Escrito en C, Shell, Perl, Tcl, Python, entre otros lenguajes de programación.
Git permite la gestión de los cambios que se realizan en los archivos y configuraciones de un proyecto, además, permite la coordinación del equipo de trabajo. [GIT2019]

3.9 Scrum

Scrum es un framework que entrega lineamientos para que un equipo logre convertirse en ágil, es decir, permite a los equipos adaptarse a problemas complejos a la vez que son productivos, permitiendo la entrega de productos con el mayor valor posible. Scrum es simple, se basa en la utilización del método científico del empirismo, en donde se aprende haciendo, en base a la experiencia. Como pilares del framework se encuentra la transparencia, inspección y adaptabilidad, y como bases la autoorganización del equipo y el respeto de sus miembros, con el fin de poder abordar la complejidad e impresionabilidad de los proyectos.
Dentro de los lineamientos definidos en Scrum se encuentra el equipo Scrum, que cuenta con 3 roles definidos, que son: Scrum Master, Product Owner y Developer Team. En lo que respecta al flujo de trabajo, se definen: Sprint, Sprint planning, Daily Scrum, Sprint Review y Sprint Retrospective. Finalmente, se definen los artefactos, que son: Product Backlog, Sprint Backlog, Increment. [SUT2020]

3.10 Jira

Jira es un producto para el seguimiento de problemas y la gestión ágil de proyectos. Fue desarrollada por Atlassian Corporation Plc, data desde el año 2002 y está escrita en Java. Cuenta con una interfaz amigable con la que permite la creación y control de tareas para la distribución del trabajo dentro del equipo, donde a través de tarjetas, se puede distinguir una tarea de otra, mostrando el trabajo a realizar en la misma.[ATL2020b]

3.11 Confluence

Confluence es una wiki empresarial basado en la web. Permite la generación de publicaciones basadas en Markdown y XHTML de forma interna por parte del equipo. Con esto, Confluence otorga las herramientas necesarias para la generación de historias y épicas, que permiten la definición de requisitos y metas a cumplir y desarrollar para el proyecto. Pertenece a la suite de productos de Atlassian, fue creado en el año 2004 y está escrito en lenguaje Java. [ATL2020a]

3.12 Justificación de Arquitectura del Software

Es necesario seleccionar una arquitectura de software que cumpla con los estándares requeridos para el software. Algunos requerimientos clave para la selección de la arquitectura a emplear son los siguientes:

- Capacidad para soportar alta cantidad de tráfico
- Capacidad para agregar módulos sin necesidad de detener el sistema
- Posibilidad de generar distintos proyectos e interconectarlos
- Validar accesos y peticiones de forma automatizada

En base a los requerimientos mencionados existe una arquitectura que logra satisfacer en su totalidad lo solicitado, que es, la arquitectura de microservicios. Algunas de las ventajas que conlleva la utilización de la arquitectura mencionada, son:

- **Modularidad:** permite generar distintos servicios independientes que pueden ser separados por funcionalidad.

- **Escalabilidad:** se pueden agregar nuevos servicios, a medida que sean necesarios, sin impactar en los ya creados.
- **Instancias:** es posible generar nuevas instancias de un microservicio en caso de que se estime necesario.
- **Independencia:** cada microservicio trabaja de forma independiente, por lo tanto, es posible realizar un mantenimiento sin afectar al sistema completo.
- **Equipos de trabajos pequeños:** permite el desarrollo por parte de equipos pequeños debido a la modularidad de los microservicios.
- **Seguridad:** permite establecer una validación compartida entre los microservicios y aplicativos de frontend.

3.13 Justificación de Tecnologías

Previo a la etapa de desarrollo del sistema es necesario realizar una evaluación sobre las tecnologías disponibles para el desarrollo del sistema. En esta sección, se presentan las tecnologías seleccionadas para el desarrollo del sistema con su respectiva justificación.

- **Angular:** en su versión 10 es el seleccionado para el frontend del sistema. La selección de este framework recae en que trabaja en base a web components, lo que permite generar una aplicación escalable y reutilizable. A su vez, permite la separación en módulos, los cuales se pueden corresponder con cada uno de los microservicios generados. Además, permite la utilización de los web componentes generados para el desarrollo de una aplicación móvil, pensando en una expansión futura. Otros puntos a favor son:
  - Open Source[1]
  - Utiliza TypeScript
  - Alto rendimiento
  - Modular

[1] `https://github.com/angular/angular`

Cabe destacar su rápida adaptación a estándares de ciberseguridad, como lo son las restricciones CORS, validación de formularios, e integración y control de *jwt* (Java Web Token), que a través de interceptores, permite la validación de autoridad y usuario a todas las peticiones (a menos que se declare lo contrario) que se realicen en la aplicación.

- **Spring:** Spring es el framework seleccionado para la realización del ecosistema conformado por microservicios. La principal razón es que cuenta con un proyecto especializado en despliegue en la nube, llamado Spring Cloud, el cual a su vez define un módulo llamado Spring Cloud Framework. Este módulo, desarrollado por Netflix, permite una rápida configuración del sistema de microservicios mediante la utilización de los patrones utilizados por los desarrolladores. Algunas otras razones son:
  - Open Source[2]
  - Robusto
  - Utilizado por empresas líderes en sus áreas
  - Gran comunidad

  Adicionalmente, gracias a su facilidad de integración y configuración de propiedades a través de *Beans*, permite la configuración avanzada pero modular en cuanto a temas de seguridad, como lo son las validaciones JWT y de CORS.

- **PostgreSQL:** El gestor de base de datos relacional seleccionado es PostgreSQL, debido a:
  - Open Source[3]
  - Escalabilidad
  - Estable y Confiable
  - Integración Nativa con Spring
  - Diseñado para servicios Web

[2] `https://github.com/spring-projects`
[3] `https://git.postgresql.org/gitweb/?p=pgrpms.git`

- **Node.js:** La elección de Node como servidor para realizar scraping es debido a la versatilidad que entrega mediante la utilización de JavaScript como lenguaje de programación junto a la librería Puppeteer[4], la que permite abrir un navegador en modo headless (segundo plano). Mediante las herramientas mencionadas, se puede acceder directamente a DOM (Document Object Model) para obtener la información presente en la página web. Otras ventajas de Node como servidor son:
  - Open Source[5]
  - Robusto
  - Escalable
  - Gran Comunidad
- **Github:** Github es el proveedor mas grande para servicios git, permitiendo mantener repositorios git, tanto privados o públicos, asegurando la estabilidad del servicio y proporcionando herramientas de trabajo y gestión propias de git, facilitando asi su uso, siendo esto una de las principales características que definieron su selección como mantenedor de los repositorios. Gracias a Github y a git, se permite el trabajo de forma remota sin necesidad de interrumpir el flujo de trabajo debido a las distancias existentes entre los integrantes del equipo, y mantener un orden entre los cambios realizados a los proyectos, los cuales se encuentran en repositorios independientes gracias a la facilidad de poder levantar repositorios de forma ilimitada. Otras razones incluyen:
  - Estable (99,9% uptime)
  - Utilizado por grandes proyectos conocidos
  - Facilitación de repositorios privados para equipos pequeños

3.14 Justificación de Herramientas de Gestión

Durante el desarrollo de los proyectos es esencial mantener una buena comunicación y claridad de los requerimientos a desarrollar, más aun en la situación actual de pandemia

[4] `https://github.com/puppeteer/puppeteer`
[5] `https://github.com/nodejs/node`

en la que fue desarrollada esta tesis, por esto, se describen a continuación las herramientas seleccionadas para la gestión del proyecto.

- **Scrum:** Respecto a una metodología para utilizar durante el desarrollo de software, Scrum fue la elegida, debido a las características ágiles que presenta. Scrum permite estar al tanto de lo que se trabaja en el proyecto mediante la daily meeting, la cual, por temas de pandemia, se realiza mediante un chat. También permite organizar los requerimientos que se conocen y los que se agregan o se eliminan a medida que se avanza en el desarrollo, así como la entrega de incrementos (partes funcionales del sistema) que en este caso, se entregan cada un mes aproximadamente, lo que implica la duración de un sprint. Algunas otras ventajas que presenta Scrum por sobre otras metodologías son:

  - División del proyecto en Sprint
  - Revisión de código durante Sprint Review
  - Genera un ambiente colectivo e innovador

- **Jira y Confluence:** La utilización de estas herramientas se debió a las características ya descritas, así como la naturaleza del proyecto, ya que se encontraba dividido según las capas o propósitos existentes, permitiendo la correcta distribución de tareas manteniendo y conservando el orden.

  Ambas herramientas fueron seleccionadas gracias a su facilidad de uso a la hora de definir y distribuir el trabajo dentro del equipo, además de su utilidad para demostrar y recibir guía por parte del profesor guía. Al ser el equipo integrado por menos de 10 integrantes, Atlassian ofrece una licencia gratuita para el uso completo de su suite. Dichas herramientas fueron esenciales durante el desarrollo del proyecto para la correcta ejecución de tareas, disminuyendo los posibles conflictos en la modificación del código de los proyectos, además de la definición exacta de la extensión del proyecto.

4 Desarrollo de Iteraciones utilizando Scrum

4.1 Análisis

La realización del proyecto se realiza bajo el marco de trabajo Scrum, la cual utiliza sprint con el objetivo de hacer entrega de incrementos en cada una de las iteraciones realizadas, así como la validación de las funcionalidades, por parte del cliente, en cada una de las entregas.

4.2 Organización del Sistema Según Capas

La realización del software considera la creación de distintos proyectos, los cuales son divididos en 4 capas de acorde a la funcionalidad correspondida por cada uno de ellos. El funcionamiento del proyecto depende de la comunicación continua y segura entre todas las partes presentes, así como de la conexión a internet para la extracción de información. En la Figura 4.1 se presentan cada una de las capas mencionadas con la respectiva tecnología a utilizar.

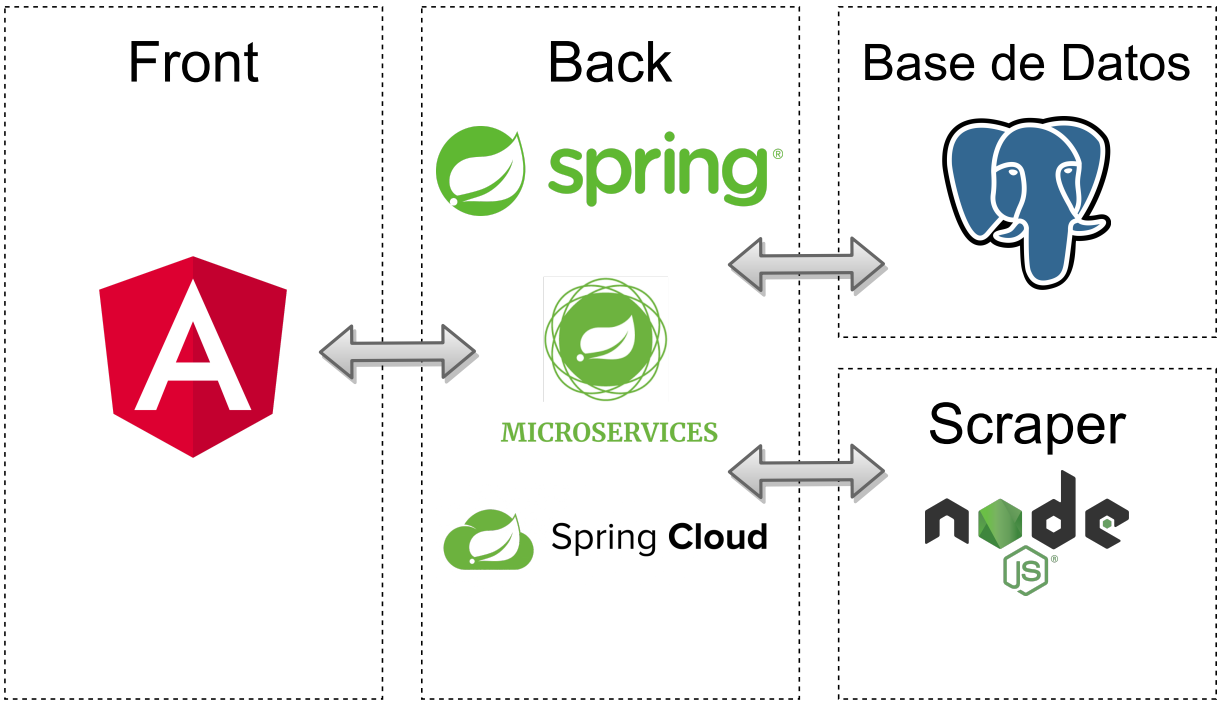

Figura 4.1: Tecnologías Utilizadas En Cada Una de Las Capas

A continuación, se procede a detallar cada unas de las partes previamente definidas.

- **Scraper:** es el encargado de obtener la información de los distintos portales de la universidad, tanto de empleados como alumnos.
- **Backend:** se encarga de almacenar la información obtenida por el Scraper y disponibilizar la información almacenada en la base de datos.
- **Frontend:** parte visual con la que interactúa el usuario.
- **Base de Datos:** almacenamiento de la información obtenida por parte del Scraper y estandarizada por el backend.

4.3 Desarrollo de Iteraciones del Sistema

Con las capas necesarias definidas, se procede a desarrollar el sistema buscando priorizar los módulos base, que son necesarios para el funcionamiento de los demás. Es por esto, que el desarrollo del sistema se divide en dos iteraciones, o sea, dos sprint pertenecientes a Scrum, los que se detallan a continuación:

4.3.1 Sprint 1

En el primer sprint, se busca obtener un ecosistema de microservicios mínimo funcional. Así, se busca extraer la información desde los distintos portales de la universidad, almacenarla en la base de datos, luego de ser limpiada y estandarizada, para posteriormente disponibilizar endpoints y poder ser consultada mediante un frontend o programas que permitan el consumo de API como Postman. Es por esto, en este sprint se generan los avances que se detallan a continuación.

- **Scraper:** Es el encargado de obtener la información desde la plataforma de estudiantes, así como de la plataforma de los académicos de la universidad. Esto se realiza a través del ingreso de credenciales validas existentes en dichas plataformas, por lo que la información obtenida corresponde exclusivamente al usuario que esta haciendo uso de ella. El scraper consta de un navegador headless (Sin ventana gráfica) que ejecuta las ordenes definidas para la obtención de la información. Una vez obtenida

la información se comunica con el backend mediante una llamada API REST.
En el proyecto se generan cuatro métodos con sus respectivos endpoints, que son:

- **Verificación de Credenciales de Alumnos:** validación de las credenciales ingresadas por el alumno, éstas se validan en el portal de alumnos de la universidad.
- **Verificación de Credenciales de Funcionarios:** validación de las credenciales de los funcionarios de la universidad en el sistema de intranet de ésta.
- **Extracción de Información de Alumnos:** Extracción de información personal y académica del portal del alumno una vez que se hayan ingresado credenciales válidas.
- **Extracción de Información de Funcionarios:** Extracción de información de intranet y simbad referente a información personal y académica respectivamente.

- **Backend:** En este sprint se constó de 3 proyectos bases necesarios para el levantamiento de los demás sistemas, que son:

  - **Eureka:** Como se ha descrito en secciones anteriores, Eureka es el encargado de registrar el estado de cada uno de los microservicios y almacenar las direcciones donde se encuentran alojados. Para lograr su cometido, como se hizo uso de los proyectos generados por Netflix para Spring, basta con realizar una configuración de dos puntos, que son:

    * **Activar Eureka Server:** se activa el proyecto generado como servidor de Eureka con una anotación en la clase con el método *main*.
    * **Asignar nombre y puerto:** En el archivo *application.properties* se define el nombre con el que se conocerá el servidor de Eureka y el puerto en el que estará alojado. Este puerto es importante ya que a ese puerto es donde envía la señal de estado de cada uno de los microservicios.

    Quedando así habilitado el servicio de descubrimiento para los microservicios que se comuniquen con Eureka.

  - **Zuul:** En esta instancia se hizo uso de un routing simple ya que aun no se implementan mayores niveles de seguridad, por lo que trabaja a modo de *reverse proxy* para redireccionar las peticiones que lleguen al ecosistema hacia los microservicios correspondientes. Junto a Eureka, cumple la función de interconectar los microservicios y facilitar la comunicación con los aplicativos, en este caso, al recibir una petición se encarga de consultar el estado del microservicio de destino a Eureka, en caso de ser una respuesta exitosa, envía la petición, para obtener una respuesta y devolver al solicitante.
  - **Microservicio Scraping:** es el encargado de comunicarse mediante llamadas a la API REST generada en el Scraper, definido anteriormente, y así obtener la información, dar formato y realizar el almacenamiento en la base de datos para que esté disponible a los demás microservicios. Es el único microservicio que se comunica con un servicio que se encuentra externo al ecosistema como tal, ya que el Scraper no es accesible mediante Zuul.

  Para realizar las pruebas de cada uno de los endpoints y funcionalidades creadas hasta este punto, debido a la ausencia de un front que permita obtener peticiones y respuestas de manera visual, se hace uso de la herramienta Postman, definida anteriormente, en donde se realizan los test de cada uno de los endpoints creados. Así mismo, cabe destacar que cada uno de los proyectos generados es almacenado en un repositorio git, bajo el proveedor Github, en donde se hace uso de control de versiones para mantener una estructura de desarrollo ordenada y con respaldos en caso de presentar fallas.

- **Base de Datos:** fue desarrollada con la ayuda de librería Spring Data JPA, perteneciente a Spring, almacena la información obtenida desde los distintos portales de la universidad mediante los distintos procesos que involucran al scraper que posteriormente es estandarizada y almacenada por parte del backend. Para el proyecto se define una única base de datos para todos los microservicios, y cada uno de ellos acceden directamente a esta. En un futuro, si se estima pertinente, no hay imposibilidad para definir una nueva base de datos a la cual acceden solo los microservicios que se definan.

4.3.2 Sprint 2

Luego de obtener una base sólida sobre la cual trabajar, obtenida en el primer sprint, se da inicio al segundo sprint. En esta iteración, se busca realizar el frontend que permita la utilización de las funcionalidades definidas en el backend. También, se pretende definir la configuración de la seguridad para la aplicación, así como la asignación y eliminación de permisos para acceder a distintos servicios presentes en el sistema. Todas las funcionalidades desarrolladas buscan ser genéricas, con el objetivo de permitir la integración de nuevos módulos en un futuro. En base a lo anterior, los distintos avances obtenidos en este sprint se detallan a continuación.

- **Frontend:** El aplicativo frontend obtenido en este sprint es un sitio web, modular y seguro. Creado en base a Angular 10, el sitio se conforma de componentes modulares para la generación de las vistas, como lo es la barra de navegación, por ejemplo, que se encuentra presente en todas las vistas (a excepción de la pantalla de ingreso).
  Para lidiar con la modularidad necesaria para poder integrar nuevas vistas al proyecto, este se divide en base a la siguiente estructura:

  - **Core:** Es dónde se almacenan todos los archivos no directamente relacionados a las vistas del proyecto, como lo son las configuraciones, guards, interceptores, modelos, interfaces y servicios.
  - **Modules:** Es donde se almacenan, según funcionalidad, los archivos relacionados a las vistas a conformar, al igual que los archivos Typescript para la lógica presente en cada componente. Dentro de Modules, se encuentran dos secciones importantes, *base*, que es para los componentes base de las vistas, como lo son las vistas de usuario, director, etc. Además se encuentran las vistas de autenticación, lo cual sirve a modo de ejemplo de un módulo mayor implementado sin influir al resto, existente dentro de la carpeta *authentication*.

  Otro factor importante es la seguridad, en donde se utilizaron los guards e interceptores, conceptos propios de Angular, y se detallan a continuación:

  - **Guard:** El guard o guardia, como lo dice el nombre, es un guardia que se ejecuta cada vez que se realiza un flujo en el proyecto, para esto existen dos

guardias:

* **Auth Guard:** Es quien se encarga de validar de la existencia del token en la sesión, de existir, valida que sea un token válido, de no ser válido o de no existir el token, se realiza la eliminación de este de la sesión y se realiza el *deslogeo*.
* **Access Guard:** Es quien extrae los permisos del usuario desde el token de la sesión, con esto se valida el acceso a alguna vista o función dentro del proyecto. Esto se realiza de igual manera al realizar un flujo dentro del aplicativo.

– **Interceptor:** Es quien se encarga de introducir el token en el header de autorización de las peticiones dentro de la petición, de esta manera el proceso se realiza de forma automática para todas aquellas peticiones que requieran validación JWT en el Backend, a menos que se indique lo contrario. Para no incluir el JWT, se debe agregar la ruta al archivo `path.config.ts`, de este modo, se pueden agregar módulos nuevos al proyecto de forma sencilla sin la necesidad de agregar manualmente el JWT en cada petición.

Al finalizar el segundo sprint, en lo que respecta a front, se pueden generar las siguientes interacciones con el ecosistema de microservicios:

– **Login:** permite el acceso al sistema mediante las credenciales provistas por la universidad, para ello, al mandar la solicitud, se genera un token en caso de contar con credenciales válidas.

– **Asignación de Permisos:** el acceso a las funcionalidades del sistema se rigen mediante permisos, los cuales pueden ser revocados o asignados según:

* **Usuario:** permite la asignación o eliminación de permisos de acceso al sistema a un usuario en específico.
* **Rol:** permite la asignación o eliminación de permisos a un rol definido en el sistema.

– **Actualización de Información:** El sistema permite definir una actualización de información por parte del scraper, en caso que se estime conveniente, la cual

se realiza en el siguiente inicio de sesión correcto por parte del usuario, y puede ser asignado según:

  * **Usuario:** permite la actualización de un usuario en específico o una lista de ellos.
  * **Todos:** permite la actualización de todos los usuarios registrados en el sistema.

- **Backend:** En este sprint, se agregaron nuevos proyectos y se generaron nuevas funcionalidades en Zuul. Lo anterior, se detalla a continuación.

  – **Zuul:** se mantuvieron las funcionalidades de routing generadas en el sprint anterior, pero debido a la necesidad de segurizar la aplicación y expansión de los microservicios se agregaron dos importantes cualidades, que son:

    * **Filtros:** se definieron filtros dentro de zuul, los cuales se ejecutan dependiendo de las condiciones definidas en cada uno de ellos, como pueden ser, solicitudes mediante cierta url o algún tipo de petición. Los filtros generados limitan el acceso de los usuarios que no cuenten con permisos a los microservicios que se definen, como es el caso del microservicio director, donde solamente pueden acceder usuarios que cuenten con tal permiso. En el caso que un filtro arroje que un usuario no cumple los requisitos para realizar la solicitud, se retorna un error 403 forbidden, que denota la prohibición de acceso a ese servicio.
    * **Seguridad:** se añaden nuevas configuraciones de seguridad, las cuales constan de implementar validaciones de JWT para todas las peticiones realizadas, a excepción del login, para así permitir el acceso solamente a usuarios que cuenten con credenciales validadas por los portales de la universidad. A su vez, se incluye la definición de CORS, para limitar los tipos de llamadas, como lo son los headers o limitar las fuentes de las cuales puede recibir peticiones.

- **Microservicio Usuarios:** este servicio se encarga de la administración de los usuarios nuevos y los ya registrados en el sistema, tanto de estudiantes como de funcionar-

ios. En este microservicio se generó la lógica para las llamadas al login y es el único microservicio que permite el acceso sin contar con el JWT, ya que para el login no es necesario, debido a que este es quien asigna un token válido en caso de obtener un inicio de sesión correcto. Este microservicio se comunica directamente con el microservicio de scraping, desarrollado en el sprint anterior, llamando a los métodos de verificación de credenciales o de extracción de información de los distintos portales, en caso de ser el primer inicio de sesión o si es requerido.

- **Microservicio Director:** este microservicio cuenta con un acceso exclusivo para personas con el permiso de director, en él se definen las funcionalidades que son realizadas exclusivamente por el director de carrera. Adicionalmente, están las funcionalidades referente a la asignación de permisos por usuario y por rol, ya que el director es el administrador del sistema. Así mismo, el director igualmente es el encargado de definir cuando es necesario realizar la extracción de información de los portales nuevamente, para actualizar la almacenada en el sistema, lo cual se puede definir a un usuario en específico o a todos los usuarios, este proceso se ejecuta cuando el usuario realiza el siguiente inicio de sesión, siempre y cuando haya ingresado con credenciales válidas.

- **Librerías:** Debido a la modularidad de los proyectos y con el objetivo de estandarizar ciertas funcionalidades, para este punto se generaron librerías que definen ciertas características que utilizan los microservicios. Debido a lo anterior, se generaron dos librerías, que son:

  - **Entidades**: Define las clases con las que se generan las tablas en la base de datos. Es utilizada por todos los microservicios que necesitan acceso a esta información, la cual se obtiene mediante las mismas clases con las que fueron definidas las tablas.

  - **Útil**: Proyecto definido para generar funcionalidades que son transversales a todos los proyectos, por ejemplo, los métodos asociados al rut, como validación, generación de DV (dígito verificador), entre otros.

5 Ecosistema Basado en Microservicios

El sistema resultante en base a la implementación mencionada a lo largo de este documento cuenta con distintos proyectos que trabajan de manera independiente y se comunican entre sí. En la Figura 5.1 se presenta un diagrama del sistema completo, explicado anteriormente, con las distintas partes que lo componen y las interacciones que se generan entre ellas.

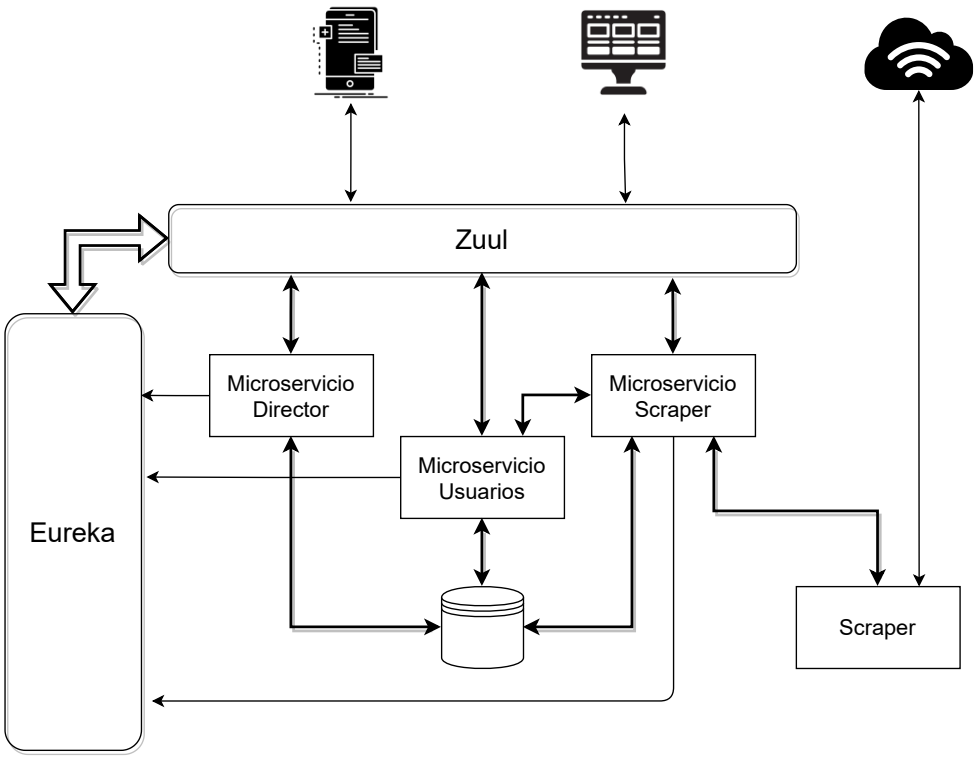

Figura 5.1: Diagrama del Sistema

Los módulos pertenecientes a Spring Cloud Netflix, Eureka y Zuul, se comunican entre ellos, además de comunicarse con los microservicios propios del ecosistema, en el caso de Eureka, solamente espera respuestas de los microservicios, los llamados heartbeat, para conocer su estado. Los microservicios desarrollados se comunican directamente con la base de datos compartida. Además, existe una interacción asociada al login y extracción de información por parte del Scraper, entre el microservicio de usuarios y el microservicio de scraping. En lo que respecta al frontend, las peticiones son enviadas directamente a Zuul, quien es el encargado de enrutar la solicitud al microservicio correspondiente.

5.1 Base de Datos

Luego de la realización de los proyectos definidos en la sección anterior durante las distintas iteraciones, la base de datos resultante se presenta en la Figura 5.2 y es la que se utiliza a modo de guía para futuros proyectos que requieran hacer uso de ella.

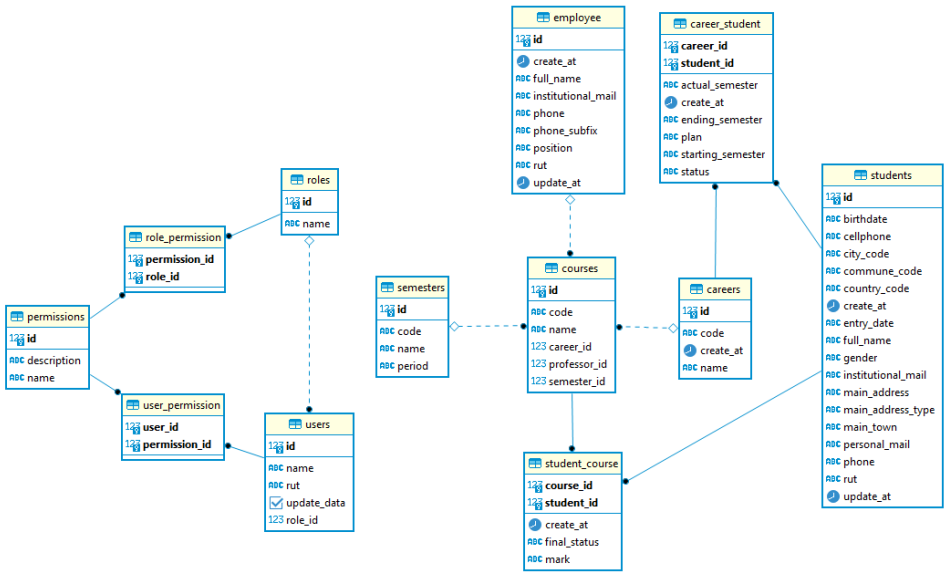

Figura 5.2: Base de Datos

En la figura anterior, se muestra el esquema de base de datos dividido en dos modelos, uno encargado del login de usuarios y la obtención de permisos y roles, y el otro donde se almacena la información personal de alumnos, funcionarios y otras informaciones referentes a cursos, carreras, etc.

Se realizó la estructuración de la base de datos de esta manera debido a la posibilidad futura de migrar el sistema relacionado a usuarios, permisos y roles a una base de datos privada, donde solamente se accede cuando se necesita realizar un inicio de sesión o consultar los permisos asociados a un usuario o rol. Lo anterior, buscando dificultar el acceso a la información privada almacenada en la base de datos de alumnos y funcionarios y así proteger la integridad de esta en caso de algún ataque malicioso.

5.2 Sistema

A continuación se muestra un flujo realizado en el sistema finalizado, demostrando la interconexión entre las distintas capas existentes. El flujo a demostrar es como sigue:

- Como se muestra en la Figura 5.3, sólo los servicios de Eureka y Zuul se levantan en puertos fijos, mientras que el resto de microservicios se levantan en puertos aleatorios, demostrando la necesidad de implementar Eureka con Zuul para la comunicación entre capas.

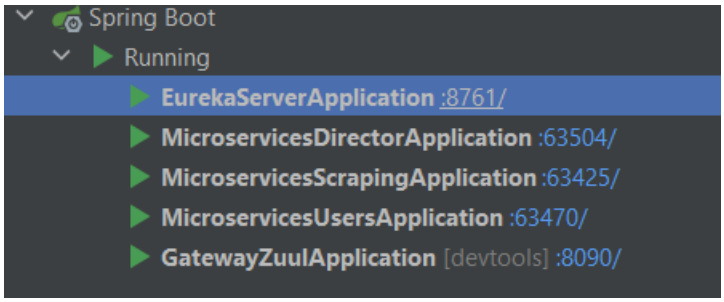

Figura 5.3: Microservicios Levantados

- Para el caso del Scraper, el puerto utilizado es siempre el puerto 3000, como se ve en la Figura 5.4:

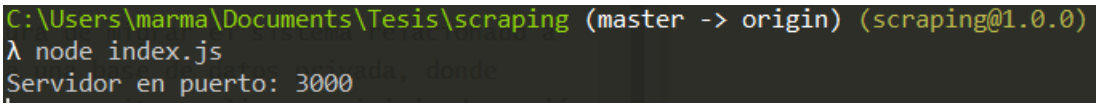

Figura 5.4: Scraper en Terminal

- Una vez levantado todos los microservicios, es posible ver los registros de estos como clientes en Eureka, como se muestra en la Figura 5.5:

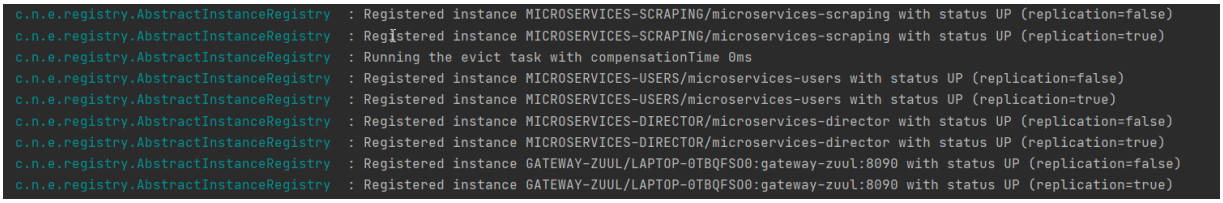

Figura 5.5: Registro Clientes en Eureka

- De igual modo se muestra en la Figura 5.6 como el servicio de director se registra en Eureka como cliente:

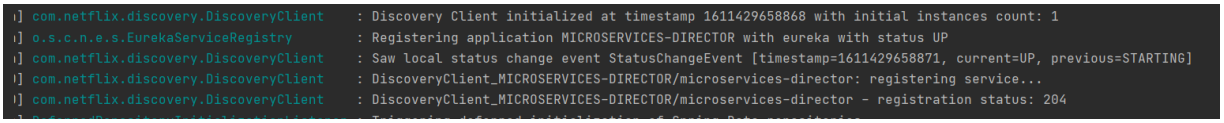

Figura 5.6: Registro Microservicio Director en Eureka

- Con esto, es posible levantar el aplicativo frontend, tal como se muestra en la Figura 5.7:

Figura 5.7: Aplicativo Front

- Una vez realizado el *login*, es posible visualizar en la Figura 5.8 el llamado al microservicio de usuarios para la obtención de datos del usuario ingresado, realizándose directamente a Zuul y no al microservicio como tal, cabe destacar que la información es inicialmente obtenida por el web scraper si es que el usuario no se encontraba inicialmente presente en la base de datos.

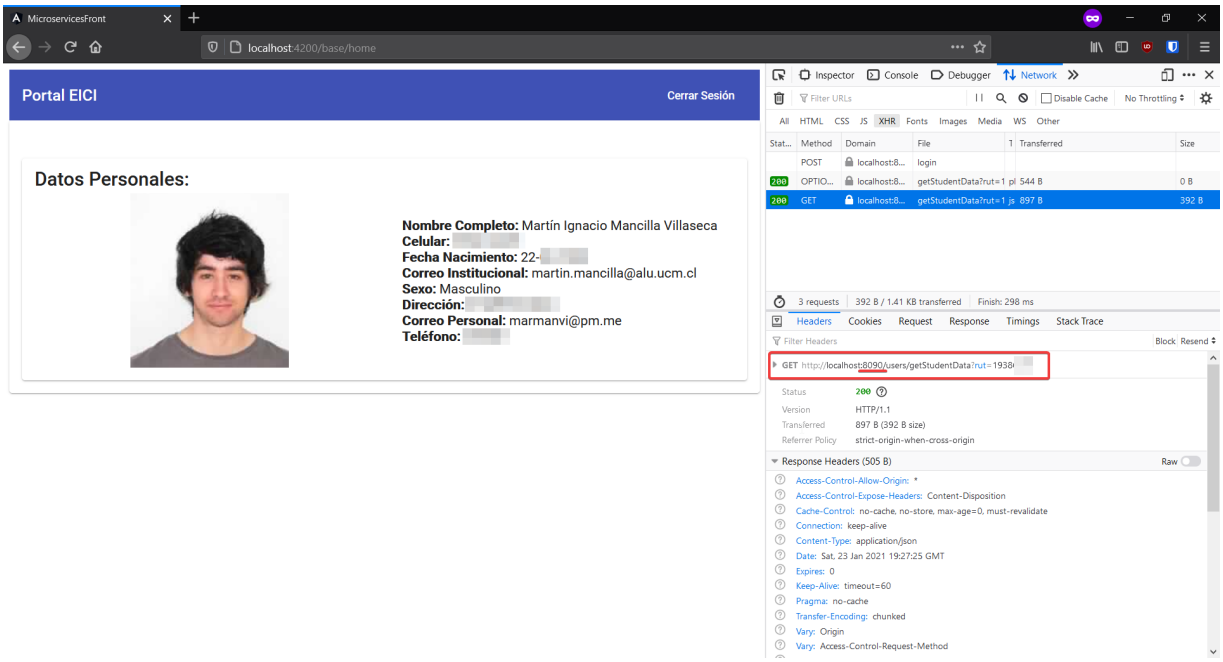

Figura 5.8: Aplicativo Front Home

- Es posible ver el redireccionamiento realizado por Zuul en la consola del mismo, como se ver en la Figura 5.9:

```
: Flipping property: microservices-users.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLi
: Shutdown hook installed for: NFLoadBalancer-PingTimer-microservices-users
: Client: microservices-users instantiated a LoadBalancer: DynamicServerListLoadBalancer:{NFLoadBalancer:name=microservices-users,current list of Servers=[]
: Using serverListUpdater PollingServerListUpdater
: Flipping property: microservices-users.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLi
: DynamicServerListLoadBalancer for client microservices-users initialized: DynamicServerListLoadBalancer:{NFLoadBalancer:name=microservices-users,current l
connection failure:0;    Total blackout seconds:0;   Last connection made:Wed Dec 31 21:00:00 CLST 1969; First connection made: Wed Dec 31 21:00:00 CLST 196

: Flipping property: microservices-users.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLi
```

Figura 5.9: Redireccionamiento en Zuul

- Para la validación de la petición, se muestra en la Figura 5.10 como en el *Header* se incluye token de autorización en la petición, y en la Figura 5.11 , se muestra como la información relacionada a permisos y tipo de usuario se incluyen en el token.

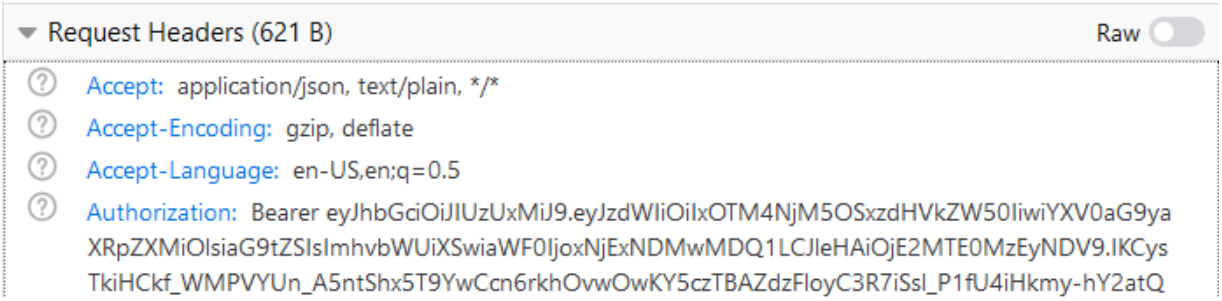

Figura 5.10: Token en Header

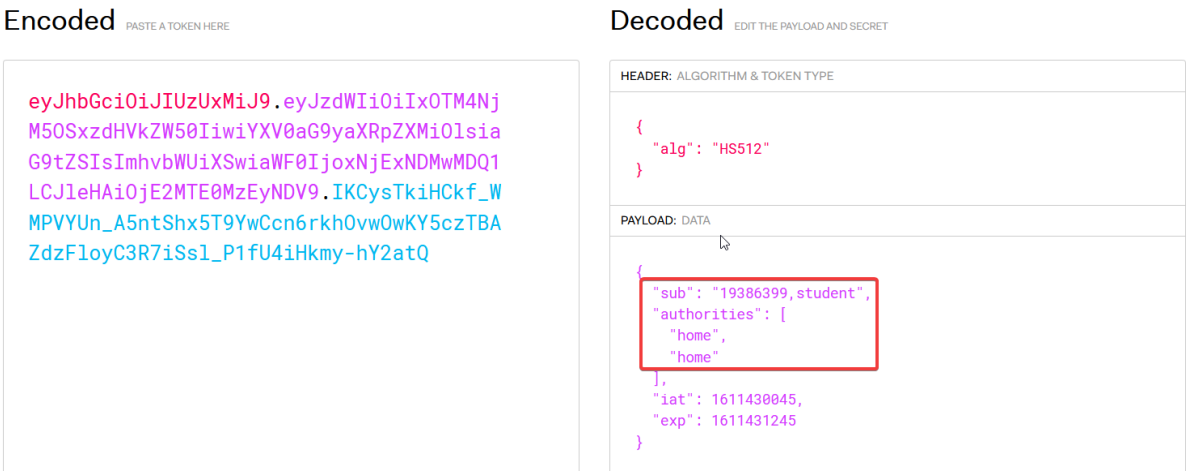

Figura 5.11: Información en Token

- Para demostrar que es posible consultar a distintos microservicios, se asigna el rol *Director* al usuario de muestra, para luego solicitar el listado de alumnos presentes en el sistema, esto a través del microservicio de director, como se muestra en la Figura 5.12, y el redireccionamiento realizado por Zuul en la Figura 5.13:

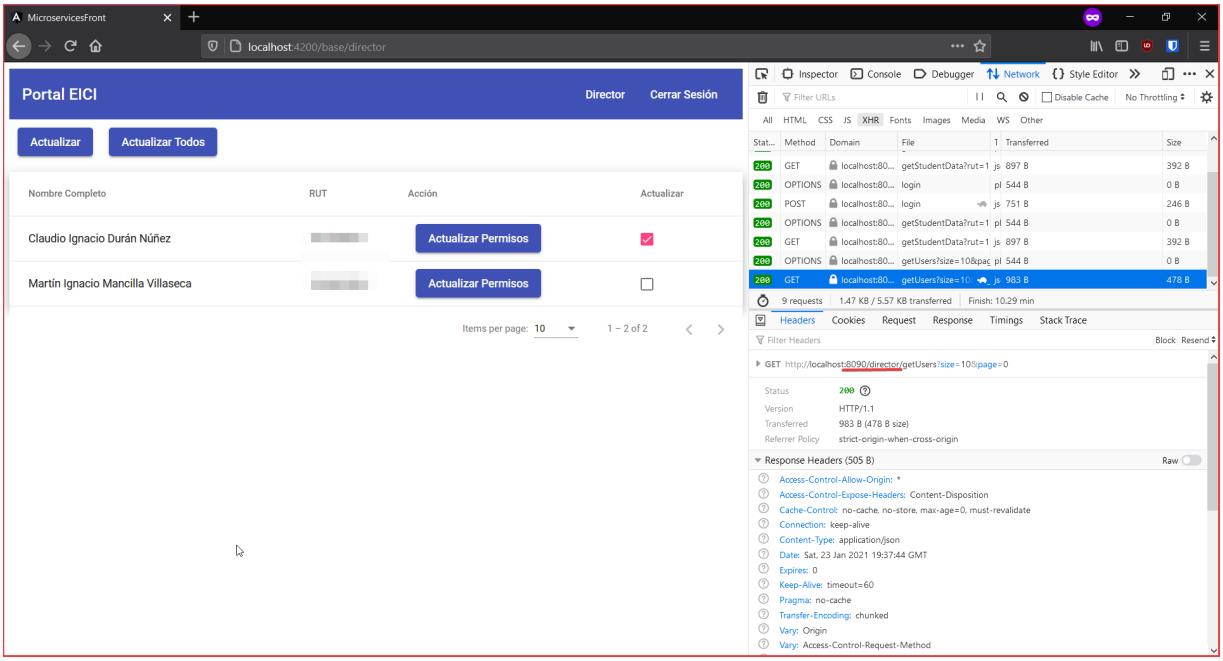

Figura 5.12: Vista Director

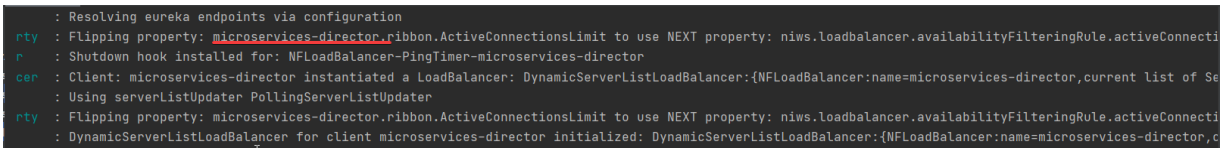

Figura 5.13: Redireccionamiento a Microservicio Director

Con lo que se demuestra el correcto funcionamiento del ecosistema completo, así como la interacción por parte de todos los proyectos generados.

5.3 Validación del Sistema

Para realizar una validación del ecosistema creado se contactó a un grupo de alumnos que amablemente aceptaron desarrollar un módulo para luego vincularlo al ecosistema de microservicios. Al grupo interesado se le concedió acceso a la librería de entidades, descrita anteriormente, y a un script con datos maquetados para que pudieran realizar las pruebas necesarias.

El módulo a desarrollar consiste en un sistema que permita realizar evaluaciones sobre distintas instancias que se presenten en las aulas, como por ejemplo, evaluar la presentación de compañeros de clases, como oyentes. Los logros obtenidos para cada una de las capas del ecosistema son los siguientes:

- **Frontend:** por temas de conocimientos y tiempo del grupo no logró realizar el desarrollo en frontend.
- **Backend:** se realizó correctamente el backend del proyecto dejando disponible distintos endpoints para ser consumidos.
- **Base de Datos:** se integraron nuevas tablas a la base de datos ya desarrollada.
- **Scraper:** no hubo incidencia en esta capa.

Posterior al desarrollo realizado por el equipo mencionado, por nuestra parte corresponde la integración del nuevo microservicio al ecosistema ya desarrollado. Para esto, se realiza una serie de sencillos pasos descrita en el manual adjunto a esta tesis, en la sección Backend. Luego de realizar la integración al ecosistema, se realizan las siguientes pruebas:

- **Eureka:** en la Figura 5.14 se aprecia los distintos microservicios descubiertos por Eureka, entre los que se encuentra el microservicio de evaluaciones, que fue el desarrollado por el equipo.

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY-ZUUL	n/a (1)	(1)	UP (1) - 192.168.1.8:gateway-zuul:8090
MICROSERVICES-DIRECTOR	n/a (1)	(1)	UP (1) - microservices-director
MICROSERVICES-EVALUATIONS	n/a (1)	(1)	UP (1) - microservices-evaluations
MICROSERVICES-SCRAPING	n/a (1)	(1)	UP (1) - microservices-scraping
MICROSERVICES-USERS	n/a (1)	(1)	UP (1) - microservices-users

Figura 5.14: Microservicios Descubiertos por Eureka

- **Zuul:** Para probar la integración de Zuul con el microservicio agregado se realizan una serie de pruebas, debido a la carencia de un frontend se realizan mediante Postman. En primer lugar, como se muestra en la Figura 5.15, se realiza una petición

GET al microservicio recientemente agregado sin incluir el token necesario, obteniendo como respuesta un error 403, que impide el acceso.

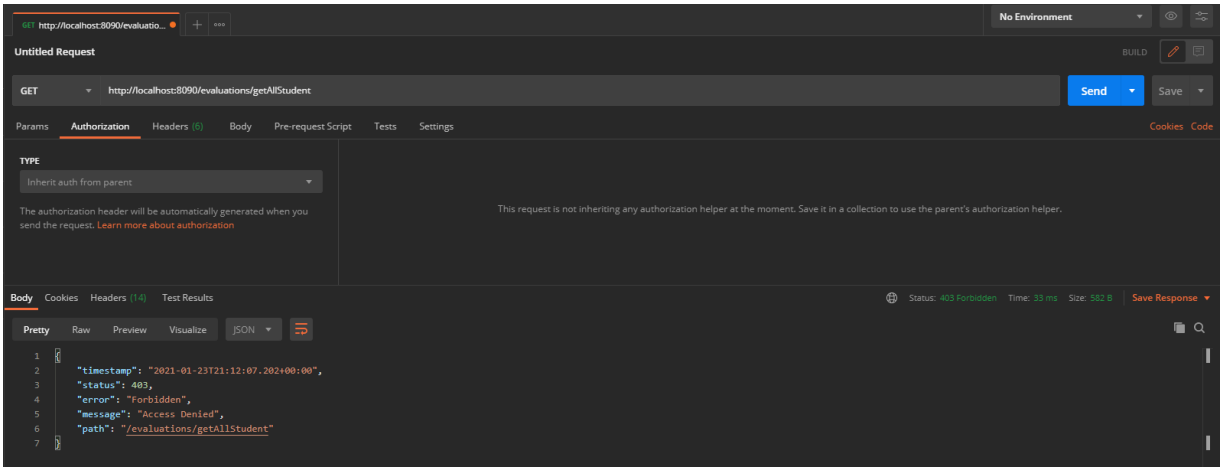

Figura 5.15: Petición sin Token

Posteriormente, se realiza una solicitud de login par obtener un token válido y poder realizar las peticiones de prueba, como se muestra en la Figura 5.16 se realiza de manera correcta, lo que demuestra que los demás microservicios se encuentran trabajando de manera correcta aun con la integración del nuevo microservicio.

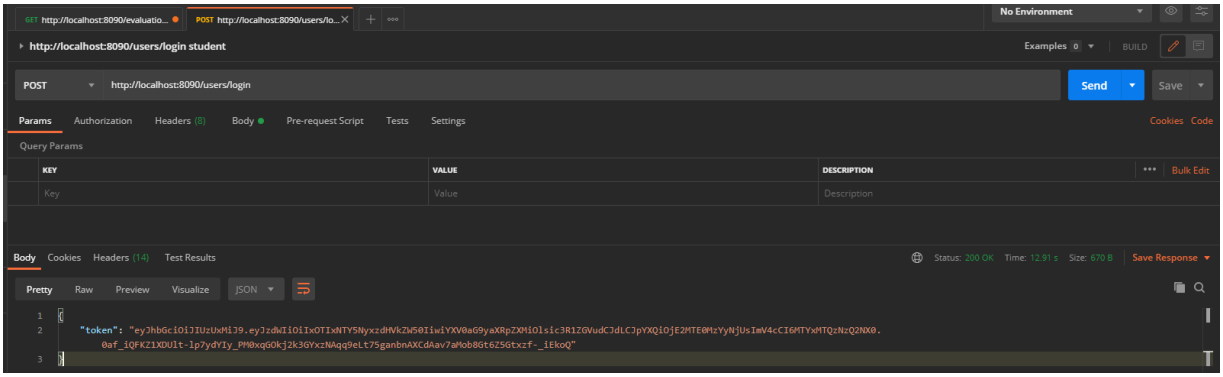

Figura 5.16: Generación de Token mediante Login Válido

Luego de contar con un token válido, se añade como autorización a la petición generada en la Figura 5.15, el que se puede observar de manera gráfica en la Figura

5.17, el cual realiza la petición de manera correcta debido a que el microservicio no solicita ningún permiso adicional, solamente contar con un token válido.

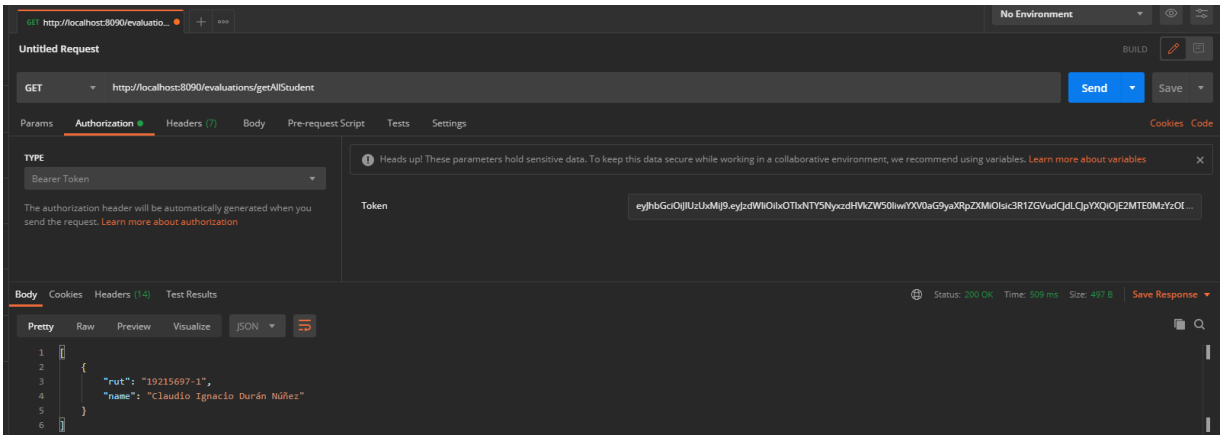

Figura 5.17: Consumo de Endpoint del Microservicio de Evaluaciones

Finalizando, el último caso de prueba es realizar la misma petición de la figura anterior, pero en este caso, con el token inválido, se puede ver gráficamente en la Figura 5.18, en donde se cortó el token para invalidarlo, obteniendo una imposibilidad de acceso al no contar con un token válido.

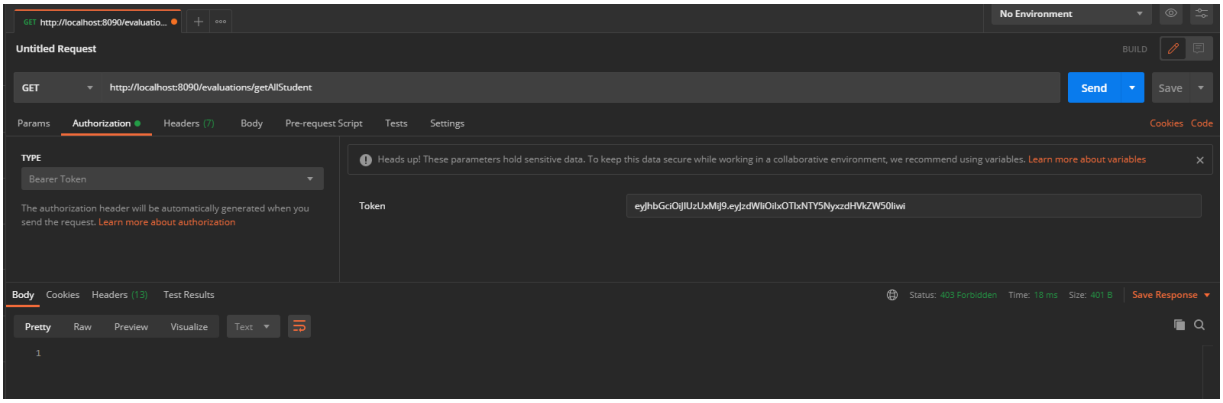

Figura 5.18: Consumo de Endpoint del Microservicio de Evaluaciones Token Inválido

- **Base de datos:** Debido a que se agregaron tablas mediante la creación del nuevo microservicio, estas se añaden a la base de datos compartida del ecosistema, quedando el esquema final como se muestra en la Figura 5.19.

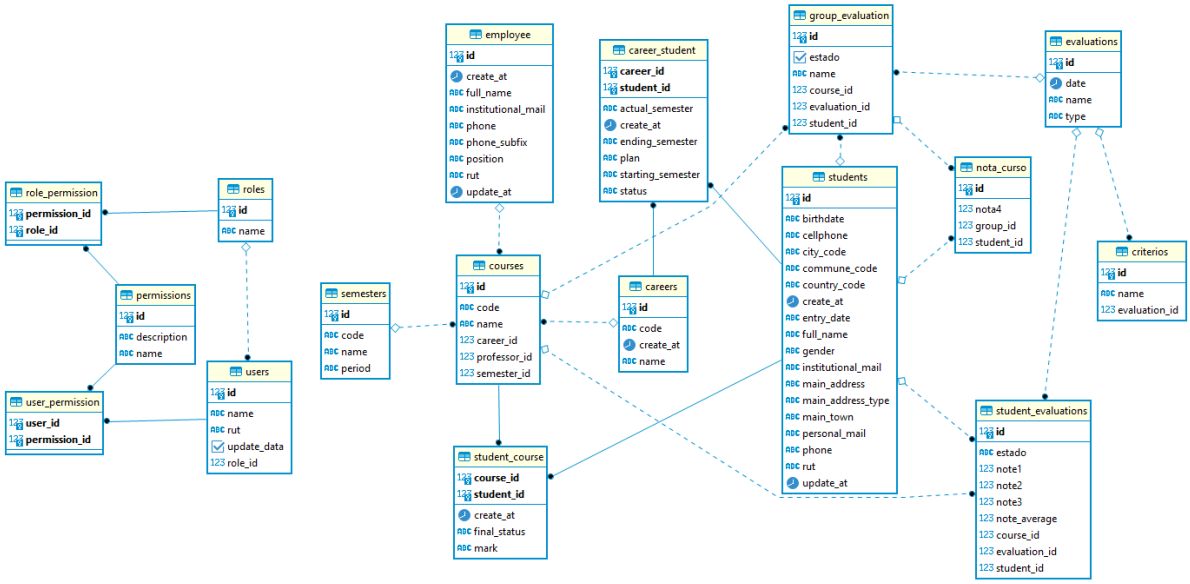

Figura 5.19: Actualización del Diagrama de Base de Datos

En base a las pruebas demostradas anteriormente, se concluye que el nuevo microservicio incluido al ecosistema está funcionando de la manera esperada. En donde se prohíbe el acceso en caso de no contar con un token o contar con uno inválido y se permite el acceso si el token es válido. En lo que respecta a la base de datos, se agregan las nuevas tablas, al modelo existente, sin ningún impedimento. Por lo que se concluye que el caso de prueba es exitoso.

6 Conclusiones y Trabajos Futuros

6.1 Conclusiones

La información es uno de los bienes más preciados por las empresas e instituciones en el mundo actual. Por esto, contar con un sistema que facilite la recopilación de ésta es importante para lograr visualizar la información y permitir una toma de decisiones enfocada en los objetivos de la empresa.

Como se evidenció en lo expuesto en la tesis, el sistema propuesto y desarrollado genera una base sólida sobre la cual se pueden desarrollar más proyectos y ser vinculados al ecosistema de microservicios. Todo esto, sin la necesidad de invertir tiempo en configuraciones excesivas de seguridad ni en un método para poblar la base de datos con información real, ya que se encuentra configurado el Scraper para poder realizar esta acción.

La utilización de microservicios para el ecosistema se desarrolló con el objetivo que alumnos de la universidad que se encuentran cursando ramos de especialidad puedan utilizar esta base y generar sus propios sistemas que serán incluidos en el ecosistema. Esto logra generar un acercamiento real a soluciones de problemas de la vida diaria de un profesional, lo que se permite trabajar en un ambiente enfocado a la puesta en práctica de la teoría vista en las aulas.

A medida que avanzaba el proyecto de desarrollo se evidenciaba la necesidad del uso de herramientas para la gestión del proyecto, como fue, en este caso Jira y Confluence. Con está tecnologías se pueden mantener en orden las tareas y requerimientos asociados a cada uno de los desarrolladores, ítem que es fundamental en el desarrollo de software y más aun en la situación actual que se encuentra el país donde se dificulta la interacción humana. Así mismo, la utilización de una tecnología de control de versiones es fundamental para cualquier equipo de desarrollo, independiente del tamaño y proyecto. Git, tecnología utilizada en este proyecto, facilita la integración de los cambios realizados en el código desarrollado por distintos integrantes, cosa que es prácticamente imposible de

realizar a mano.

Otro punto destacable en la realización de esta tesis es la gran importancia que tiene el software Open Source. El software utilizado para el desarrollo y gestión, prácticamente en su totalidad, es Open Source, lo que permite la utilización de herramientas desarrolladas por equipos de profesionales y entusiastas del software de manera gratuita. Un claro ejemplo es la utilización de *Spring Cloud Netflix*, que es desarrollado, mantenido y utilizado por Netflix actualmente para su sistema de streaming a nivel mundial.

A nivel personal y profesional, nos enfrentamos a algunas tecnologías nuevas para nosotros, sin embargo, el haber realizado el proyecto nos permitió darnos cuenta del potencial que hemos desarrollado durante nuestro paso por la universidad. Esto, debido a que sin tener nociones de algunas tecnologías nos enfrentamos al proceso de entenderlas y estudiarlas para posteriormente desarrollar con ellas. Todo el proceso nos permite valorar y validar habilidades que hemos adquirido en estos años de estudio, así como la capacidad de autoaprendizaje que se ha ido desarrollando con el paso del tiempo y que es tan valorada en la industria de la tecnología.

6.2 Trabajos Futuros

Respecto a Trabajos futuros, algunas implementaciones que se pueden realizar son:

- **Proceso de QA:** Realizar una validación completa del sistema por alguien especializado para buscar posibles falencias y corregirlas antes de su lanzamiento.
- **Implementación de un Logger:** implementar un logger que permita llevar una trazabilidad de las operaciones realizadas en el sistema.

Adicionalmente, está abierta la posibilidad de desarrollar más microservicios y nuevas funcionalidades que se pueden agregar al ecosistema. Así, se puede generar una herramienta que permita solventar algunas falencias que se encuentran presentes en los sistemas de información de la universidad y fuerza a los estudiantes a salir de la zona de confort al trabajar en base a un proyecto ya en marcha y desarrollar siguiendo los lineamientos y estándares ya definidos.

Bibliografía

[ATL2020a] Atlassian. *Confluence basics*. 2020. URL: https://www.atlassian.com/software/confluence/guides/get-started/confluence-overview#about-confluence.

[ATL2020b] Atlassian. *What is Jira used for?* 2020. URL: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for.

[BOR2018] Mátyás Lancelot Bors. *An overview of Angular*. Mar. 2018. URL: https://medium.com/@mlbors/an-overview-of-angular-3ccd2950648e.

[DOC2020] MDN Web Docs. *Express/Node introduction*. Dec. 2020. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

[GIT2019] Git. *About*. 2019. URL: https://git-scm.com/about.

[GLO2020] Charter Global. *Five Microservices Trends in 2020*. Oct. 2020. URL: https://www.charterglobal.com/five-microservices-trends-in-2020/.

[H2020] Jeremy H. *4 Microservices Examples: Amazon, Netflix, Uber, and Etsy*. 2020. URL: https://blog.dreamfactory.com/microservices-examples/.

[HAT2019] Red Hat. *What are microservices?* 2019. URL: https://www.redhat.com/en/topics/microservices.

[NOD2020] Nodejs. *About Node.js*. ene 2020. URL: https://nodejs.org/en/about/.

[NPM2020] npm. *puppeteer*. Nov. 2020. URL: https://www.npmjs.com/package/puppeteer.

[POS2019] Postman. *The Postman API Platform*. 2019. URL: https://www.postman.com/api-platform/.

[POS2020] PostgreSQL. *About*. 2020. URL: https://www.postgresql.org/about/.

[SPR2020a] Spring. *Spring Cloud Netflix*. 2020. URL: https://spring.io/projects/spring-cloud-netflix.

[SPR2020b] Spring. *Spring Framework Overview*. Dec. 2020. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html.

[SUT2020] Ken Schwaber & Jeff Sutherland. *The Scrum Guide*. 2020. URL: https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[YCH2020] YCHARTS. *Amazon.com Market Cap*. 2020. URL: https://ycharts.com/companies/AMZN/market_cap.