UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Creación de una arquitectura de alta disponibilidad utilizando contenedores y microservicios para procesos concurrentes en una Institución de Educación Superior

CARREÑO TEJADA WILLY ALVARO
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

JARRIN UNUZUNGO JORGE ENRIQUE
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Creación de una arquitectura de alta disponibilidad utilizando contenedores y microservicios para procesos concurrentes en una Institución de Educación Superior

CARREÑO TEJADA WILLY ALVARO
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

JARRIN UNUZUNGO JORGE ENRIQUE
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

PROPUESTAS TECNOLÓGICAS

Creación de una arquitectura de alta disponibilidad utilizando contenedores y microservicios para procesos concurrentes en una Institución de Educación Superior

CARREÑO TEJADA WILLY ALVARO
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

JARRIN UNUZUNGO JORGE ENRIQUE
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

CARTUCHE CALVA JOFFRE JEORWIN

MACHALA
2022

UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

PROPUESTAS TECNOLÓGICAS

Creación de una arquitectura de alta disponibilidad utilizando contenedores y microservicios para procesos concurrentes en una Institución de Educación Superior

CARREÑO TEJADA WILLY ALVARO
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

JARRIN UNUZUNGO JORGE ENRIQUE
INGENIERO EN TECNOLOGIAS DE LA INFORMACION

CARTUCHE CALVA JOFFRE JEORWIN

MACHALA
2022

tesis

INFORME DE ORIGINALIDAD

7%	6%	1%	3%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad Técnica de Machala Trabajo del estudiante	1%
2	es.education-wiki.com Fuente de Internet	<1%
3	www.archivalencia.org Fuente de Internet	<1%
4	www.slideshare.net Fuente de Internet	<1%
5	lorca.uc3m.es Fuente de Internet	<1%
6	ebin.pub Fuente de Internet	<1%
7	www.arpug.com.ar Fuente de Internet	<1%
8	www.anp.com.uy Fuente de Internet	<1%
9	Submitted to UNIV DE LAS AMERICAS Trabajo del estudiante	<1%

CLÁUSULA DE CESIÓN DE DERECHO DE PUBLICACIÓN EN REPOSITORIO DIGITAL INSTITUCIONAL

Los que suscriben, CARREÑO TEJADA WILLY ÁLVARO y JARRÍN UNUZUNGO JORGE ENRIQUE, en calidad de autores del siguiente trabajo escrito titulado Creación de una arquitectura de alta disponibilidad utilizando contenedores y microservicios para procesos concurrentes en una institución de Educación Superior, otorgan a la Universidad Técnica de Machala, de forma gratuita y no exclusiva, los derechos de reproducción, distribución y comunicación pública de la obra, que constituye un trabajo de autoría propia, sobre la cual tienen potestad para otorgar los derechos contenidos en esta licencia.

Los autores declaran que el contenido que se publicará es de carácter académico y se enmarca en las disposiciones definidas por la Universidad Técnica de Machala.

Se autoriza a transformar la obra, únicamente cuando sea necesario, y a realizar las adaptaciones pertinentes para permitir su preservación, distribución y publicación en el Repositorio Digital Institucional de la Universidad Técnica de Machala.

Los autores como garantes de la autoría de la obra y en relación a la misma, declaran que la universidad se encuentra libre de todo tipo de responsabilidad sobre el contenido de la obra y que asumen la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera exclusiva.

Aceptando esta licencia, se cede a la Universidad Técnica de Machala el derecho exclusivo de archivar, reproducir, convertir, comunicar y/o distribuir la obra mundialmente en formato electrónico y digital a través de su Repositorio Digital Institucional, siempre y cuando no se lo haga para obtener beneficio económico.

CARREÑO TEJADA WILLY ÁLVARO
0706655552

JARRÍN UNUZUNGO JORGE ENRIQUE
0707072153

UNIVERSIDAD TÉCNICA DE MACHALA

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

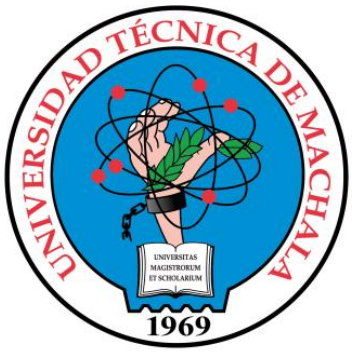
CREACIÓN DE UNA ARQUITECTURA DE ALTA DISPONIBILIDAD UTILIZANDO CONTENEDORES Y MICROSERVICIOS PARA PROCESOS CONCURRENTES EN UNA INSTITUCIÓN DE EDUCACIÓN SUPERIOR

CARREÑO TEJADA WILLY ÁLVARO

JARRÍN UNUZUNGO JORGE ENRIQUE

MACHALA

2023



UNIVERSIDAD TÉCNICA DE MACHALA

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR
PROPUESTAS TECNOLÓGICAS

CREACIÓN DE UNA ARQUITECTURA DE ALTA DISPONIBILIDAD UTILIZANDO CONTENEDORES Y MICROSERVICIOS PARA PROCESOS CONCURRENTES EN UNA INSTITUCIÓN DE EDUCACIÓN SUPERIOR

CARREÑO TEJADA WILLY ÁLVARO

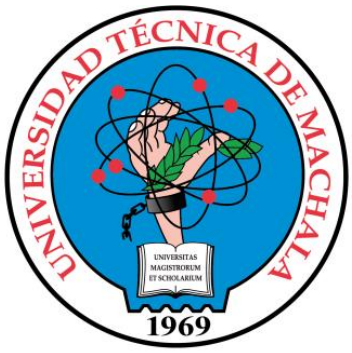
INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

JARRÍN UNUZUNGO JORGE ENRIQUE

INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

ING. CARTUCHE CALVA JOFFRE JEORWIN, MGS.

MACHALA, MARZO DE 2023



UNIVERSIDAD TÉCNICA DE MACHALA

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR
PROPUESTAS TECNOLÓGICAS

CREACIÓN DE UNA ARQUITECTURA DE ALTA DISPONIBILIDAD UTILIZANDO CONTENEDORES Y MICROSERVICIOS PARA PROCESOS CONCURRENTES EN UNA INSTITUCIÓN DE EDUCACIÓN SUPERIOR

CARREÑO TEJADA WILLY ÁLVARO

INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

JARRÍN UNUZUNGO JORGE ENRIQUE

INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

ING. CARTUCHE CALVA JOFFRE JEORWIN, MGS.

MACHALA, MARZO DE 2023

DEDICATORIA

El destino me llevó a conocer docentes excepcionales, hoy en día colegas con una pasión y vocación increíble en su profesión, familia y amigos que siempre me dieron la mano de forma desinteresada, a mis padres, aquellos seres increíbles que me vieron nacer, crecer, fallar y levantarme, y hoy por hoy, triunfar por ellos.

A todos ellos, dedico este trabajo, y a mí, aquel niño que creció desde pequeño con sueños de grandeza, dando siempre lo mejor de sí, buscando ser el orgullo de sus padres, gracias a ti, por nunca darte por vencido.

Carreño Tejada Willy Álvaro

Dedico este trabajo de titulación en primer lugar a mis padres, quienes siempre me han apoyado incondicionalmente para seguir estudiando y trabajar duro por las cosas a las que aspiro lograr.

A mis familiares y amigos que me han apoyado durante todo el proceso de mis estudios y me han ayudado en los momentos que más los necesitaba.

A mis docentes, por brindarme todos sus conocimientos, sus experiencias en el área laboral y así mismo cómo mejorar y ser un excelente profesional.

Jarrín Unuzungo Jorge Enrique

AGRADECIMIENTOS

Sin lugar a dudas me faltan palabras para expresar lo agradecido que me siento con todas las personas que me ayudaron con su apoyo moral, material e intelectual, a conseguir lo que, al momento de escribir estas palabras, es la cúspide de mi carrera académica, que con tanto esfuerzo logré alcanzar.

A mis padres, que me apoyaron siempre con esfuerzo y sacrificio, trabajando honradamente para que jamás faltara nada, a conseguir lo que ellos no pudieron y que nunca dudaron que yo podría conseguir, aquellos que siempre estuvieron orgullosos de mí, y del hombre que el día de hoy les dedica estas palabras.

A ellos, les agradezco por creer en mí, en mis sueños y jamás abandonarme.

Carreño Tejada Willy Álvaro

Quisiera agradecer a todas las personas que hicieron posible la elaboración de este trabajo de titulación, en primer lugar, a Dios por darme la fuerza, paciencia y voluntad de seguir adelante en mis estudios.

A mis padres, por su amor, apoyo emocional y económico y la motivación que me dieron durante todos estos años de mis estudios.

A mis familiares y amigos que siempre me apoyaron y ayudaron en lo que necesitaba.

Finalmente, a todos los docentes de mi carrera, quienes brindaron sus ideas y conocimientos para fortalecerme y mejorar como profesional.

Jarrín Unuzungo Jorge Enrique

RESUMEN

A raíz de la creciente demanda de recursos informáticos que implican la creación de servicios en la nube, para instituciones y empresas, han surgido interesantes propuestas arquitectónicas que nacen a partir de la necesidad de soluciones informáticas con grandes niveles de estabilidad, rendimiento y disponibilidad frente a una cantidad masiva de usuarios. Las Instituciones de Educación Superior se enfrentan a procesos que son demandados por grandes volúmenes de usuarios como el cuerpo docente, personal administrativo y el alumnado. Con especial enfoque en este último, se necesita cubrir las necesidades de estabilidad, rendimiento y disponibilidad, para el funcionamiento correcto de los servicios informáticos, Por lo tanto, el presente proyecto tiene por objetivo crear una arquitectura de alta disponibilidad, mediante el uso de contenedores y microservicios, para la evaluación del rendimiento de un prototipo altamente concurrente del sistema web de matrículas del Centro de Educación Continua de la Universidad Técnica de Machala, frente a un gran número de peticiones simultáneas, aplicando la metodología SEED(S) para el diseño de servicios altamente cohesivos y de bajo acoplamiento, que permitan minimizar los inconvenientes evidentes que posee el sistema actual de cara al usuario, cuando se apertura los periodos de matriculación. Como productos resultantes, se obtuvieron dos arquitecturas de software: la primera, una arquitectura de microservicios capaz de cumplir con los requerimientos previstos para soportar hasta 8000 usuarios concurrentes, y la segunda, una arquitectura monolítica que cubre el mismo dominio de negocio, y que fue utilizada para contrastar el rendimiento frente a grandes niveles de concurrencia simulada, demostrando así la superioridad de la arquitectura propuesta, en cuanto a rendimiento, cantidad de solicitudes atendidas, error porcentual y tiempo medio de respuesta, remarcando como recomendación primordial la utilización tanto de hardware como sistemas operativos adecuados para la realización de pruebas tan exigentes y obtener resultados coherentes y precisos.

PALABRAS CLAVE

Arquitectura de alta disponibilidad, microservicios, Docker, disponibilidad, rendimiento.

SUMMARY

As a result of the growing demand for computing resources that involves the creation of cloud services for institutions and companies, interesting architectural proposals have arisen from the need for computing solutions with high levels of stability, performance and availability for a massive number of users. Higher Education Institutions face processes that are demanded by large volumes of users, such as faculty, administrative staff and students. With special focus on the latter, it is necessary to meet the needs of stability, performance and availability, for the proper functioning of computer services. Therefore, this project aims to create a high availability architecture, using containers and microservices, to evaluate the performance of a highly concurrent prototype of the enrollment web system of the Continuing Education Center of the Technical University of Machala, against a large number of simultaneous requests, applying the SEED(S) methodology for the design of highly cohesive services and low coupling, to minimize the obvious drawbacks that the current system has for the user, when the enrollment periods are opened. Two software architectures were obtained as the resulting products: the first, a microservices architecture capable of meeting the requirements foreseen to support up to 8000 concurrent users, and the second, a monolithic architecture covering the same business domain, which was used to contrast performance against high levels of simulated concurrency, thus demonstrating the superiority of the proposed architecture, in terms of performance, number of requests handled, percentage error and average response time, highlighting as a primary recommendation the use of both hardware and operating systems suitable for performing such demanding tests and obtaining consistent and accurate results.

KEYWORDS

High availability architecture, microservices, Docker, availability, performance.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

GLOSARIO

A

Arquitectura de software: Una arquitectura de software de un sistema es la representación gráfica del diseño de la estructura general y el comportamiento de un sistema, donde se muestra tecnologías, conexiones y plataformas que se usan para su desarrollo.

D

Disponibilidad: La disponibilidad de un sistema se refiere al tiempo de continuidad de ejecución de un sistema sin que ocurra fallas durante un período de tiempo específico.

F

Framework: Un framework es una plataforma que proporciona una estructura base para el desarrollo de aplicaciones, es como una plantilla que combina bibliotecas, imágenes, código y referencias en un solo paquete y este paquete puede ser modificado por el desarrollador agregando o remplazando funciones con el propósito de adaptarlo según sea la necesidad.

M

Microservicios: Microservicios o arquitectura de microservicios es una arquitectura de software que se utiliza en la entrega o implementación continua de aplicaciones grandes y complejas.

P

Proceso concurrente: Un proceso concurrente en un sistema informático se refiere a las situaciones en las cuales un gran grupo de usuarios realizan peticiones de forma simultánea en un mismo proceso del sistema.

Prototipo: Un prototipo es un diseño, modelo o versión preliminar de un producto, como un sistema, el cual es diseñado para poner en prueba la funcionalidad un concepto y recopilar comentarios de los usuarios finales antes de invertir en la producción.

S

Servicio web: Un servicio web es un servicio del software que admite la interacción de máquina a máquina con la que se puede interactuar a través de una red.

INTRODUCCIÓN

Día a día las tecnologías de la información ganan cada vez más terreno en todos los ámbitos de la sociedad, la evolución y adaptación de las tecnologías al contexto socioeconómico y comercial, han permitido dar forma a nuevas y mejores formas de abordar problemas, para afrontar retos y dar solución a los problemas de información del siglo XXI. Es más que evidente la digitalización de servicios en las últimas décadas, provocando la modernización de servidores, creación de técnicas y tecnologías apropiadas, todo esto con la finalidad de suplir la creciente demanda de recursos computacionales que implican el proveer servicios informáticos a un gran número de usuarios. Con especial énfasis en los patrones arquitectónicos que han tomado gran relevancia en investigaciones e implementaciones en grandes empresas tecnológicas, los microservicios han surgido como una nueva forma de abordar problemas, debido a las grandes ventajas que ofrece este estilo arquitectónico, consiguió llegar a ser la tendencia actual de la industria en cuanto al desarrollo de software web y servicios en la nube. Tomando como referencia esta tendencia, apoyada por la fuerte aceptación y adopción de dicha arquitectura por las empresas líderes en innovación tecnológica digital, motivó a que el desarrollo de la presente propuesta tecnológica tuviera como foco principal el diseñar una arquitectura de alta disponibilidad, mediante el uso de contenedores y microservicios.

La idea principal de este estudio, recayó en la investigación, creación, desarrollo y pruebas del patrón arquitectónico de microservicios aplicado al contexto del sistema de matrículas del Centro de Educación Continua de la UTMACH, mismo que ha sufrido evidentes inconvenientes en cuanto a rendimiento, lo que ha significado una mala percepción del mismo por su lento desempeño y caídas del servicio frente a un gran número de solicitudes simultáneas y un malestar general en la comunidad universitaria por la incapacidad de utilizar la plataforma. Así pues, los resultados obtenidos permitieron resaltar la clara ventaja en cuanto a rendimiento que ofrece un sistema basado en microservicios y contenedores, sirviendo como precedente para la innovación y mejoramiento de los servicios digitales que pudiera ofrecer la universidad y departamentos relacionados con necesidades de soportar altos niveles de concurrencia.

i. Declaración y formulación del Problema

Declaración del problema

En el mundo, incluso antes de que la catastrófica pandemia del COVID-19 empujara a las empresas e instituciones a digitalizarse, ya se evidenciaba esta tendencia de la expansión comercial en favor de los servicios digitales, por sobre los no digitales. Dado que la demanda de servicios digitales seguirá creciendo rápidamente incluso después de la mencionada emergencia, es trascendental que se posea una infraestructura y se manejen técnicas y tecnologías capaces de soportar la demanda digital actual [1].

En la región sudamericana, se concibe la importancia de la infraestructura tecnológica a partir de los requerimientos de conectividad de la nueva era, por el éxito del streaming, juegos en tiempo real, reuniones virtuales, ofimática en la nube, en dónde: "La disponibilidad de infraestructura constituye un factor indispensable para garantizar el crecimiento de Internet…". Sin embargo, en nuestra región [2], se hace alusión sobre la deficiencia existente en cuanto a CDNs (Content Delivery Network), identificando solo una de estos en América Latina, ubicado en Quilicura, Santiago de Chile.

Con respecto de las Instituciones de Educación Superior de nuestro país, se han experimentado grandes transformaciones digitales con el fin de que la tecnología proporcione nuevas vías de desarrollo intelectual en las universidades, adoptando y regulando el modelo de educación para las carreras a distancia, en línea y semipresencial o de convergencia de medios. Así como lo dictamina el Art. 77 del reglamento de régimen académico vigente señalado que para su ejecución, estas carreras deberán contar con la infraestructura tecnológica qué "… garantizará el funcionamiento de la plataforma informática, protección de la información de los usuarios y contará con mecanismos de control para combatir el fraude y la deshonestidad académica" [3].

No obstante, el impacto de la emergencia sanitaria empujó de manera precipitada los sistemas de Educación Superior de nuestro país hacia la modalidad de estudio en línea, sistemas que aún no estaban preparados para dar el salto tecnológico y dar un servicio de calidad a sus respectivas comunidades universitarias. Esta nueva modalidad, implementada a fin de mitigar la notable repercusión que causó el confinamiento, implicaron retos sin precedentes hacia la infraestructura tecnológica y administración de sistemas informáticos de las Instituciones de Educación Superior.

Tal es el caso, del proceso de matrículas del Centro de Educación Continua de la Universidad Técnica de Machala, que, durante la emergencia presentó una serie de problemas bastante notorios por parte de su alumnado, en concreto un bajo rendimiento de los servicios informáticos, imposibilidad de acceder y efectuar matrículas a niveles de inglés, que a su vez que generaron una mala perspectiva por sobre los sistemas informáticos de la IES en cuestión. Por tal motivo, se creó una arquitectura de alta disponibilidad con microservicios y contenedores capaz de soportar hasta 8000 peticiones concurrentes. En la Figura 1, se presenta el problema, sus causas y efectos identificados.

Figura 1. Declaración del problema.

Formulación del problema

- **Problema principal:**
    - Necesidad de un sistema capaz de satisfacer la alta concurrencia del sistema de matriculación del Centro de Educación Continua de la Universidad Técnica de Machala.
- **Problemas específicos:**
    - ¿Qué beneficios y desventajas implica una arquitectura basada en microservicios?
    - ¿Qué metodologías son idóneas para la creación de la arquitectura basada en microservicios?
    - ¿Qué tecnologías se pueden utilizar para crear una arquitectura de alta disponibilidad?
    - ¿Qué métricas son idóneas para demostrar la alta disponibilidad de la arquitectura?

ii. Objeto de estudio y Campo de acción

Objeto de estudio

- Sistemas de procesos concurrentes en una Institución de Educación Superior.

Campo de acción

- Arquitectura de alta disponibilidad aplicando el patrón de microservicios.

iii. Objetivos

Objetivo General

- Crear una arquitectura de alta disponibilidad, mediante el uso de contenedores y microservicios, para la evaluación del rendimiento de un prototipo altamente concurrente del sistema web de matrículas del Centro de Educación Continua de la Universidad Técnica de Machala.

Objetivos específicos

- Caracterizar el estilo arquitectónico de microservicios, principales beneficios y desventajas de su implementación.
- Diseñar una arquitectura de alta disponibilidad usando contenedores y microservicios.

- Desarrollar un prototipo de aplicación web para matrículas del Centro de Educación Continua de la UTMACH aplicando la arquitectura establecida.
- Evaluar el rendimiento de la arquitectura propuesta frente a la simulación con un gran número de peticiones simultáneas.

iv. Hipótesis y variables o Preguntas de investigación

Hipótesis

La creación de una arquitectura basada en microservicios y contenedores mejora el rendimiento de un sistema web.

Declaración y categorización de variables

La declaración de variables identificadas y su respectivo concepto Tabla 1.

Tabla 1. Variables identificadas.

Variables	Conceptos
Variable Independiente: Arquitectura de software basada en microservicios y contenedores	Desarrollo de una arquitectura de software de alta disponibilidad.
Variable dependiente: Sistema web de matrículas (prototipo) concurrente del Centro de Educación Continua de la UTMACH.	El software resultante de la implementación de la arquitectura es capaz de satisfacer las necesidades de concurrencia de sistemas informáticos en Instituciones de Educación Superior

La categorización de la variables y dimensionamiento se puede observar en la

Tabla 2.

Tabla 2. Variables y dimensionamiento.

Variables	Categorías	Indicadores	Técnicas
Variable Independiente: Arquitectura de software basada en	• Arquitectura de software con contenedores y microservicios.	• Modelo conceptual de la arquitectura.	• Recopilación de información para la y creación de la arquitectura.

Variables	Categorías	Indicadores	Técnicas
microservicios y contenedores	• Programación Web con microservicios.	• Desarrollo de la aplicación web de matrículas.	• Selección de la tecnología de contenedores. • Comunicación a nivel de sistemas. • Selección de tecnologías de desarrollo web. • Virtualización de contenedores.
Variable dependiente: Sistema web de matrículas (prototipo) concurrente del Centro de Educación Continua de la UTMACH.	• Pruebas de rendimiento del sistema monolítico. • Pruebas de rendimiento de los microservicios.	• Tiempo medio de respuesta. • Error porcentual. • Rendimiento. • Cantidad de solicitudes atendidas.	• Pruebas de carga.

v. Justificación

En la actualidad con respecto de la ingeniería de software, las organizaciones requieren de soluciones robustas, escalables y mantenibles que se ajusten acorde a las necesidades del nuevo mundo digital. Por consiguiente, es importante considerar las nuevas técnicas, métodos, tecnologías emergentes, que surgen precisamente para suplir estas necesidades de la era moderna [4], [5]. Además, los desarrolladores de software han diseñado e implementado diferentes tipos de arquitecturas a lo largo del tiempo que ayudan a que los productos de software sean eficientes en recursos y se compongan de requisitos funcionales.

La arquitectura monolítica ofrece algunas ventajas y desventajas que dan lugar a confianza o inconvenientes tecnológicos. Es necesario conocer los riesgos que pueden derivar en problemas a corto o largo plazo. Entre las ventajas más significativas se encuentran los sistemas estables, el

sistema de gestión completa, utilizado por las grandes empresas del mundo como IBM, Sun Microsystems y BMC. Los servicios profesionales de estas empresas tienen un alto nivel de conocimiento sobre sus productos. Entre algunas desventajas: son sistemas rígidos y difíciles de adaptar a las nuevas necesidades [4].

En el transcurso del tiempo, las soluciones arquitectónicas han evolucionado con la finalidad de desarrollar nuevos y mejores sistemas basados en la demanda permanente de servicios informáticos que apoyan todo tipo de mecanismos computarizados. No obstante, la adopción de estas propuestas arquitectónicas implica esfuerzo, dinero y tiempo. Esta última variable se ve muy resaltada en los sistemas de información pertenecientes a ciertas organizaciones gubernamentales y privadas, Las cuales no tienen como una de sus prioridades el mantenimiento o renovación de productos y soluciones digitales, obviando por completo el valor que brinda la innovación tecnológica.

La bien conocida arquitectura monolítica, es un enfoque tradicional ampliamente adoptado en la industria, donde una aplicación se construye con una sola base de código que incluye múltiples servicios. De ahí que, en primera instancia este enfoque fuera implementado en el sistema web de matrículas del Centro de Educación Continua de la Universidad Técnica de Machala.

Dicho estilo arquitectónico es incapaz de escalar, separar responsabilidades, y requieren de servidores únicos que soporten toda la aplicación [6]. Claramente no satisfacen las exigencias que requieren las organizaciones con un gran volumen de consumidores, donde; se enfatizan factores como la escalabilidad (para soportar cada vez más usuarios) y modificabilidad (para agregar nuevas funcionalidades y adaptar las ya existentes), provocando malestares en los usuarios y terminando en la obsolescencia del software. En consecuencia, el sistema mencionado ha manifestado varios de los síntomas descritos previamente, particularmente en periodos de inscripción, afectando este servicio tan vital para la comunidad universitaria.

Por tales motivos, es trascendental acoger nuevas técnicas y tendencias que permitan solucionar estos inconvenientes, es allí donde emerge como propuesta el diseño orientado a microservicios, que plantea un nuevo enfoque que descompone el dominio del negocio en contextos pequeños, débilmente acoplados y desplegables de forma independiente [6], [7].

El patrón arquitectónico de microservicios se caracteriza por su escalabilidad, modificabilidad, alto grado de resiliencia y compresibilidad de los sistemas [8]–[10], por lo cual enfatiza el diseño y desarrollo de software estable, escalable y altamente mantenible.

Este estilo de arquitectura ha surgido como un paradigma para desarrollar aplicaciones mediante la composición de servicios, donde que cada servicio realiza un proceso delimitado brindando atributos de calidad de software [11]. Estos servicios mencionados suelen distribuirse mediante contenedores independientes que, si llegaran a fallar, pueden sustituirse por otro de las mismas características [12], además de poder agregar servicios bajo demanda para suplir necesidades solo cuando sea necesario, ahorrando poder de cómputo.

Por lo tanto, esta investigación aspiró a la creación de una arquitectura con estas características, para que, con ayuda de otra arquitectura monolítica, someter ambas arquitecturas a duras pruebas de grandes niveles de concurrencia, contrastar los resultados y evidenciar con un sustento cuantitativo fiable, la clara ventaja en cuanto a rendimiento del sistema orientado a microservicios por sobre el diseño monolítico.

vi. Organización del documento

El presente trabajo de titulación se encuentra organizado en una introducción y tres capítulos, los cuales serán detallados a continuación:

Introducción: En el apartado de introducción se realizó la declaración y formulación del problema, la explicación del objeto de estudio y campo de acción, elaboración de los objetivos, hipótesis y variables o preguntas de investigación, justificación y la distribución del trabajo de integración curricular.

Capítulo I: En el primer capítulo se muestran los antecedentes de la investigación que constan de aplicación de la metodología de revisión sistemática de literatura la cual permitió la elaboración de los antecedentes históricos, antecedentes teóricos y antecedentes contextuales.

Capítulo II: En el segundo capítulo se explica el desarrollo del prototipó, para la cual se realizó la definición del prototipo, explicación de la metodología de desarrollo del prototipo, enfoque, alcance y diseño de investigación. Además, se explicó las unidades de análisis, Técnicas e instrumentos de recopilación de datos, técnicas de procesamiento de datos para la obtención de resultados, metodología o métodos específicos, herramientas y/o materiales y por último se realizó el desarrollo y ejecución del prototipo.

Capítulo III: En el tercer capituló se encuentra la evaluación del prototipo en dónde se elaboró un plan de evaluación y se explica los resultados de la evaluación y, por último, la formulación de las respectivas conclusiones y recomendaciones.

CAPÍTULO I. MARCO TEÓRICO

1.1. Antecedentes de la Investigación

Para la revisión bibliográfica de la investigación se utilizó la metodología de Revisión Sistemática de la Literatura (SRL: Systematic Review of the Literature), además se usó la herramienta online Parsifal, que ayudó en la revisión sistemática de las diferentes investigaciones obtenidas.

El propósito de la metodología SLR es recopilar toda la evidencia, información, publicaciones y documentos relevantes sobre el tema de investigación con los criterios de elegibilidad predefinidos para responder a las preguntas planteadas de la investigación actual [13]. Está metodología utiliza procedimientos seguros y sistemáticos para minimizar la aparición de sesgos durante la búsqueda, identificación, evaluación, síntesis, análisis y resumen de estudios.

a) Preguntas de investigación

Las preguntas de investigación que se plantearon para realizar la búsqueda de información sobre la creación de una arquitectura de alta disponibilidad utilizando contenedores y microservicios, se detallan en la Tabla 3.

Tabla 3. Preguntas de investigación.

Pregunta	Descripción y motivación
¿Qué es y cómo funciona una arquitectura de microservicios?	Esta pregunta pretende buscar caracterizar el funcionamiento de la arquitectura de microservicios.
¿Qué técnicas son aplicables para el diseño de la arquitectura de microservicios?	Esta pregunta pretende buscar modelos, técnicas para el diseño de la arquitectura basada en microservicios y contenedores.
¿Cuáles son las tecnologías usadas para la implementación de una arquitectura de microservicios?	Esta pregunta pretende buscar herramientas, tecnologías, proyectos, lenguajes, software de terceros que se utilizan para la implementación de la arquitectura.
¿Cuáles son los beneficios y desventajas que se consiguen con la implementación de arquitecturas con microservicios?	Esta pregunta pretende buscar beneficios, resultados, experiencias y desventajas que

Pregunta	Descripción y motivación
	se consiguen en la implementación de la arquitectura de microservicios.
¿Qué métricas son idóneas para demostrar la alta disponibilidad de la arquitectura?	Esta pregunta pretende buscar métricas para medir, testear, verificar la alta disponibilidad de la arquitectura basada en microservicios y contenedores.
¿El sistema que implementa una arquitectura basada en microservicios mejora el rendimiento frente a altos niveles de concurrencia?	Esta pregunta aspira demostrar la superioridad de la arquitectura propuesta, usando las métricas obtenidas en las pruebas de rendimiento del sistema de microservicios y monolítico.

b) Palabras claves y Cadena(s) de búsqueda

La estrategia de búsqueda de información en las diferentes bases de datos bibliográficas se utilizaron palabras claves y cadenas de búsqueda utilizando los términos principales de la investigación, con el propósito de encontrar publicaciones en palabras claves, títulos, resumen, etc.

Cadena de búsqueda en español:

- (Microservicios) AND (Disponibilidad OR Resiliencia) AND (Software)
- (Microservicios) AND (Contenedores OR (Docker OR Kubernetes))
- (Microservicios) AND (Conceptos OR Fundamentos OR Características OR (Ventajas AND Desventajas)
- (Microservicios) AND (Rendimiento OR Pruebas OR Métricas)
- (Microservicios) AND ((Sistemas OR Aplicaciones) AND Tecnologías OR (Framework OR Herramientas OR Base de Datos))

Cadena de búsqueda en inglés:

- (Microservices) AND (Availability OR Resilience) AND (Software)
- (Microservices) AND (Containers OR (Docker OR Kubernetes))
- Microservices) AND (Concepts OR Fundamentals OR Characteristics OR (Advantages AND Disadvantages))

- (Microservices) AND (Performance OR Tests OR Metrics)
- (Microservices) AND ((Systems OR Applications) AND Technologies OR (Framework OR Tools OR Database))

c) Proceso y resultados de la búsqueda

Proceso de búsqueda.

El proceso de búsqueda se realizó mediante el uso de las palabras claves y cadenas de búsquedas en las diferentes bases de datos bibliográficas, las cuales son:

- IEEE Xplore.
- Science Direct.
- Scopus.
- Web of Science.

A continuación, en la Figura 2 se detalla los pasos del proceso de búsqueda de las investigaciones que fueron obtenidas a través del uso de las palabras claves y cadenas de búsqueda.

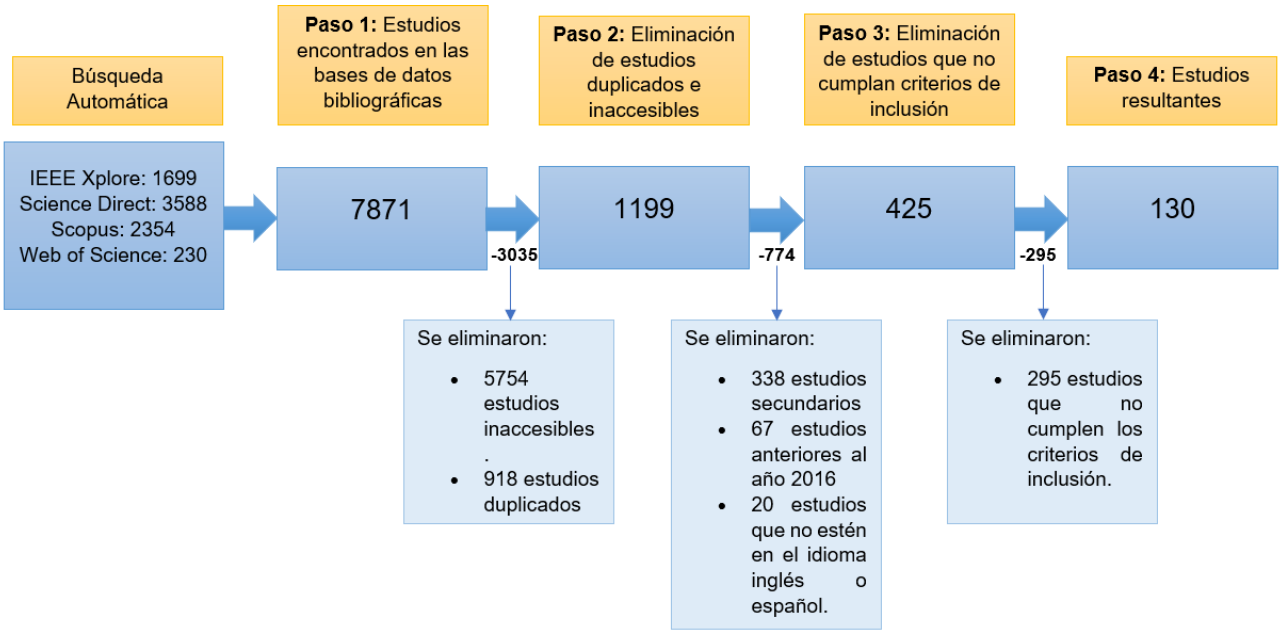


Figura 2. Proceso de búsqueda.

Resultados finales de búsqueda

Los resultados de búsqueda se clasificaron de acuerdo al año, en la Figura 3 se visualiza el número de estudios por año, revelando que en el año 2021 se dio un crecimiento notable de trabajos de investigación referentes a la temática de microservicios.

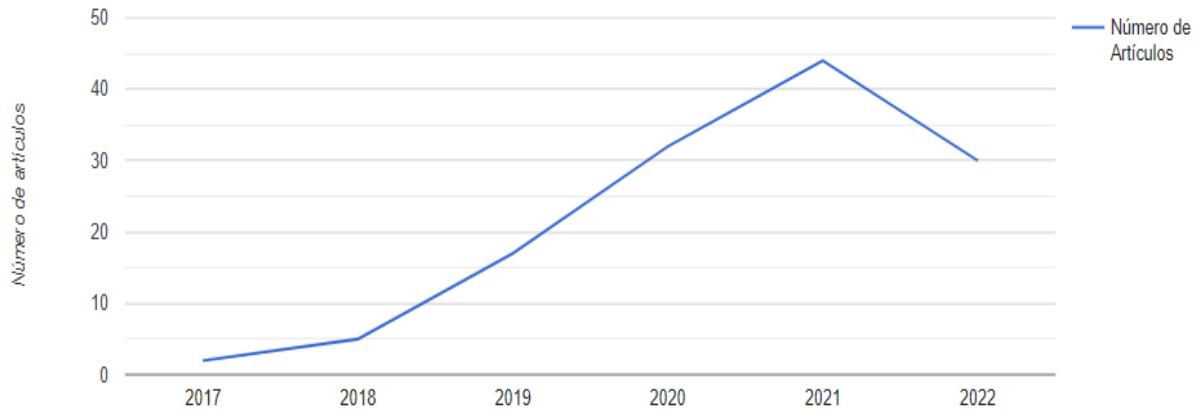


Figura 3. Diagrama de resultado de búsqueda de cantidad de estudios por año.

En la Figura 4 se visualiza un diagrama del resultado de búsqueda por el área de estudio, en donde se enfatiza que el 45,6% de trabajos de investigación enfocados al tema de investigación son realizados en el área de ciencias computacionales, seguido del área de ingeniería con 18,1%.

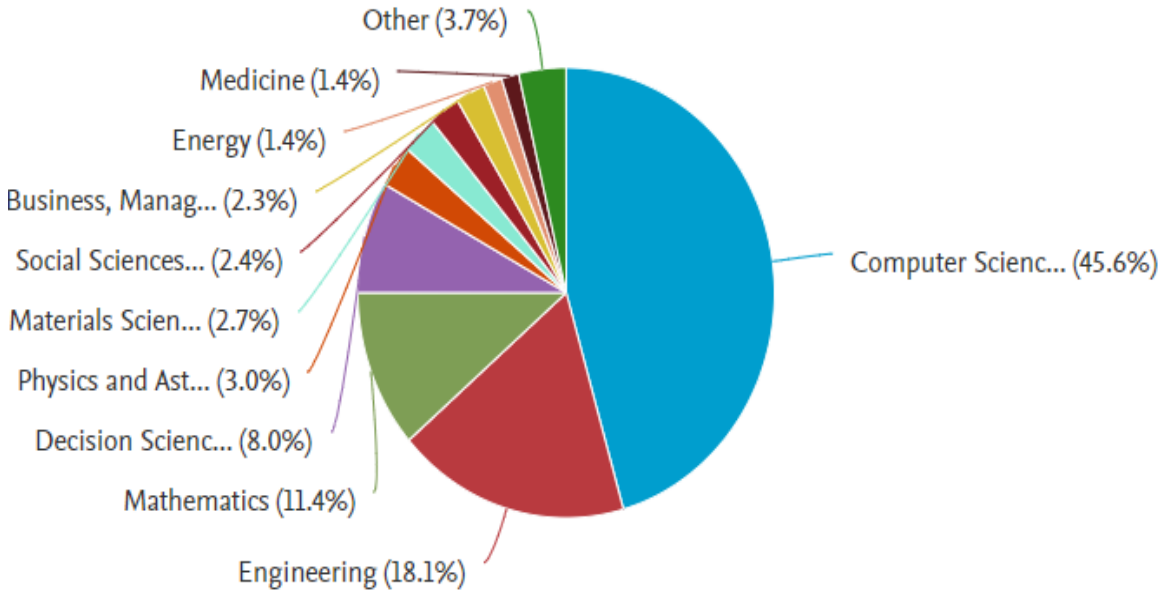


Figura 4. Diagrama de resultado de búsqueda por área de estudio.

Se utilizó la herramienta VOS-Viewer para realizar el gráfico de concurrencia y relación entre las palabras claves, el gráfico que se observa en la Figura 5, donde la palabra clave Microservicios, junto con arquitectura de software, arquitectura de microservicios y Docker son los términos que guardan estrechas relaciones en los trabajos de investigación.

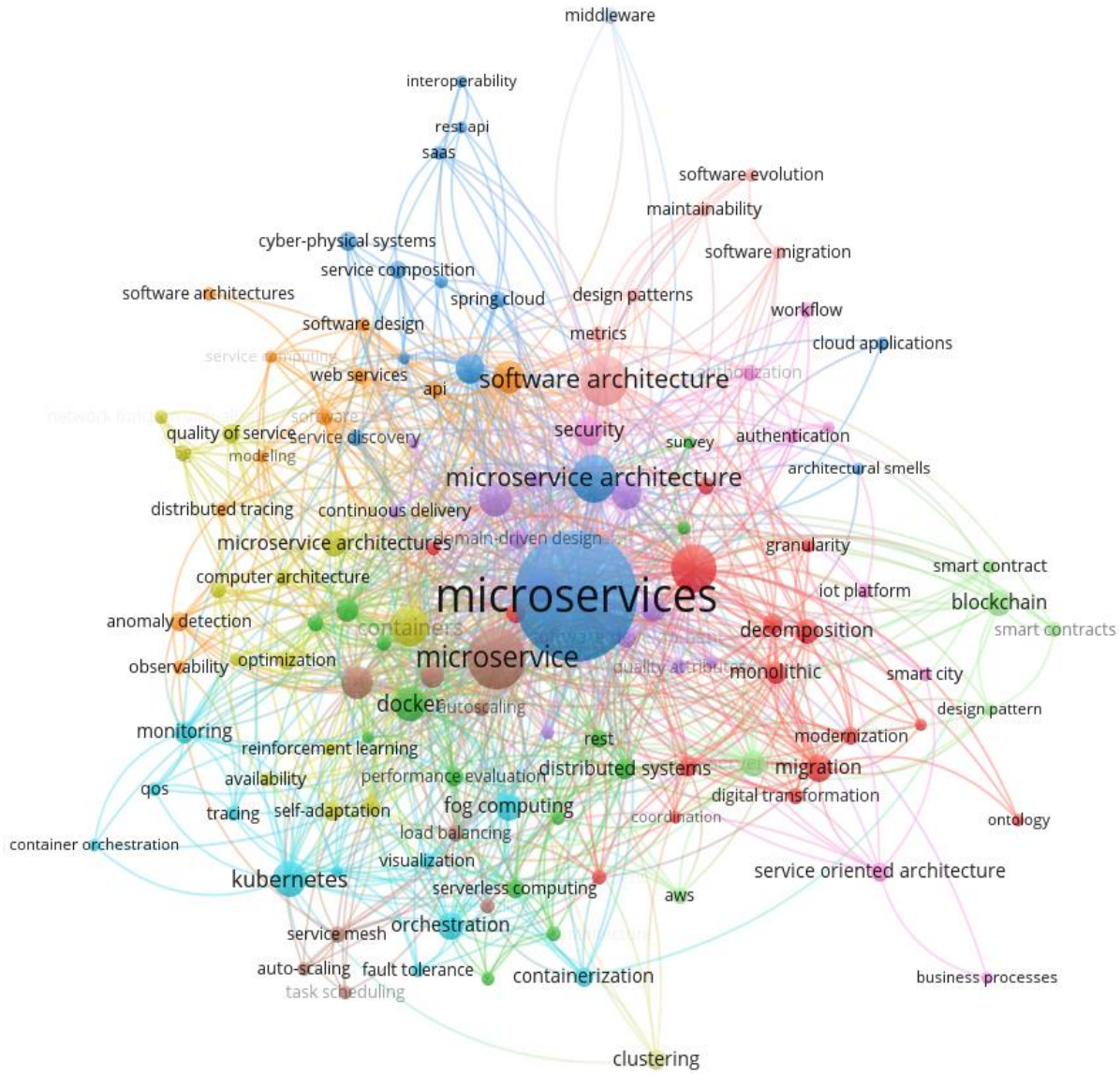


Figura 5. Gráfico de resultado de búsqueda de concurrencia y relación entre palabras claves.

1.2. Antecedentes Históricos

Las arquitecturas de alta disponibilidad con microservicios y contenedores es una tecnología que está en tendencia en el área de desarrollo de software en los últimos años. Esta tecnología está emergiendo como un nuevo paradigma informático para los nuevos y futuros sistemas y la computación en la nube. En la Figura 6 se resume la línea de tiempo de la evaluación de los microservicios y contenedores.

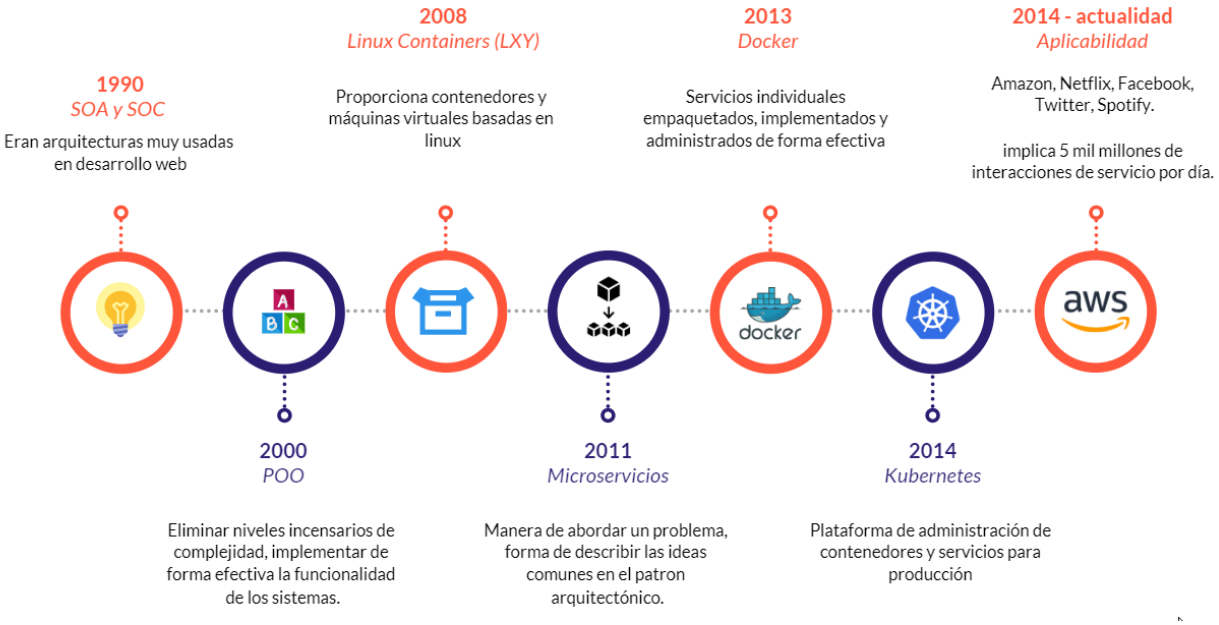


Figura 6. Línea de tiempo de la evolución de microservicios y contenedores.

En la década de 1990, la arquitectura orientada a servicios (SOA), computación orienta a servicios (SOC) y los servicios web eran tecnologías muy usadas logrando reemplazar a la arquitectura monolítica.

La arquitectura SOA fue propuesta como tecnología innovadora para trabajar aplicaciones del lado del servidor, conocido como backend y así mejorar la reutilización de componentes permitiendo el dinamismo, modularidad, reutilización de los servicios, desarrollo distribuido e integración de sistemas heterogéneos y heredados [14], [15].Las organizaciones usaban SOA y SOC para obtener el éxito clave de sus proyectos, pero con el paso de tiempo y la gran demanda de requisitos por parte de los usuarios hicieron que estas tecnologías fracasaran y en la actualidad no sean muy usadas. Las organizaciones usaban SOA y SOC para obtener el éxito clave de sus proyectos, pero con el paso de tiempo y la gran demanda de requisitos por parte de los usuarios hicieron que estas tecnologías fracasaran y en la actualidad no sean muy usadas.

En la década del 2000, surge la segunda generación de servicios donde la arquitectura SOA y SOC se la aplica junto con la programación orientada a objetos (POO) con el propósito de eliminar niveles incensarios de complejidad para mejorar la programación de servicios y estos se puedan implementar de forma efectiva la funcionalidad de los sistemas [11].

En el año 2008 aparece Linux Containers (LXC), proyecto que proporciona contenedores y máquinas virtuales que ejecuten sistemas Linux [16], permitiendo la distribución y un entorno

neutral para el desarrollo de tecnologías de contenedores Linux y permitiendo su uso para el desarrollo con microservicios.

En el año 2011, el término de microservicios aparece en un grupo de arquitectura de software [16], como una forma de describir las ideas comunes de los participantes en los patrones arquitectónicos.

En el año 2013, aparece Docker como una tecnología de contenedorización [16], permitiendo a los servicios individuales se empaqueten, implementen y administren de manera más efectiva en tiempo de ejecución.

En el año 2014, James Lewis y Martin Fowler definieron por primera vez el estilo arquitectónico basado en microservicios, desde entonces la popularidad de los microservicios ha ido aumentado de forma constante [17].

Kubernetes aparece en el año 2014, plataforma de administración de contenedores y servicios para producción con el propósito de automatizar las tareas de administración y asignación de contenedores, esencialmente excluyendo la infraestructura física o virtual subyacente de los desarrolladores de servicios [16].

Debido a la creciente popularidad de la arquitectura de microservicios en el año 2014 aparece Amazon Web Services y en el año 2016 Azure de Microsoft y Google Cloud, plataformas desarrolladas por las grandes compañías, las cuales son muy populares por todos los servicios de computación en la nube que ofrecen [16]. Estás plataformas permitieron que los usuarios desarrollen, implementen y entreguen en producción funcionalidades sin la necesidad de utilizar y administrar recursos de infraestructura físicos.

En el año 2020, con la propagación del virus COVID-19, toda la población a nivel mundial entro en confinamiento, por tal razón hubo un consumo concurrente de las plataformas o aplicaciones que ofrecen servicios digitales, además en ese mismo año los microservicios comenzaron a ganar más popularidad y de forma rápida las empresas como Amazon, Netflix, Facebook, Twitter, Spotify, etc eligieron la arquitectura de microservicios para ofrecer sus servicios digitales y mejorar sus plataformas [16], debido a que está tecnología les permitió la facilidad de escalar y hacer más resistentes los sistemas, a las fallas que otros estilos arquitectónicos puedan tener cuando se implementan en producción.

El sistema de servicios en línea de Netflix utiliza más de 500 microservicios, lo que implica 5 mil millones de interacciones de servicio por día. Cada página de Amazon involucra de 100 a 150 peticiones de microservicio [14], [15].

Actualmente, la mayoría de los problemas que se presentaban en el rendimiento de los servidores de los sistemas han sido superados, pero aun así aparecen nuevos desafíos. Los microservicios son una práctica recomendada junto con la tecnología de contenedores [14], por lo cual los programadores siguen desarrollando aplicaciones con la arquitectura de microservicios y contenedores, haciendo los sistemas más flexibles, escalables, modulares y eficaces entregando un ambiente de forma dinámica, ligera, rápida, sencilla y confiable, dando soporte a las demandas de escala global.

1.3. Antecedentes Teóricos

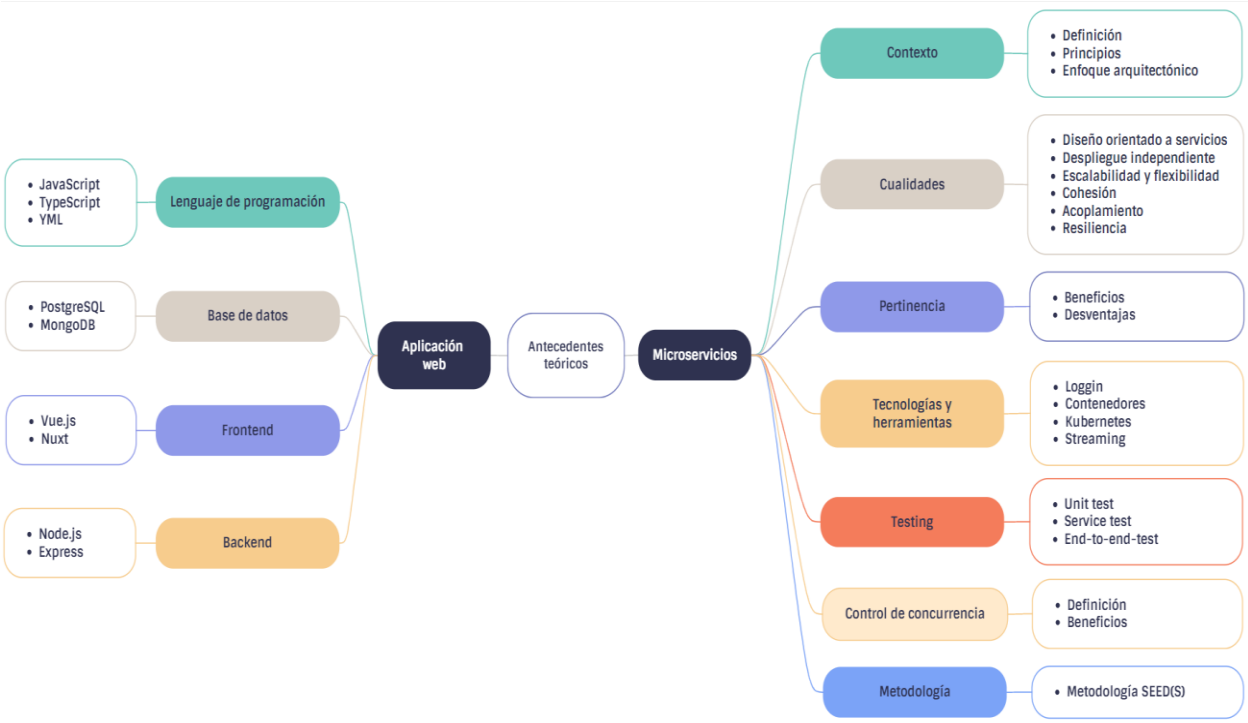


Figura 7. Antecedentes teóricos.

1.3.1. Microservicios

Definición

Los microservicios es una tecnología que se está convirtiendo en una alternativa en el desarrollo de software de aplicaciones complejas y distribuidas. Está tecnología surgió como un patrón de diseño arquitectónico que tiene como objetivo abordar la escalabilidad y facilitar el mantenimiento

de servicios en línea. Además, la seguridad ha aumentado en la disponibilidad, la integridad y la confidencialidad de los sistemas basados en microservicios [18]. Esta tecnología se ha vuelto cada vez más popular durante los últimos años debido que son servicios pequeños, autónomos y desplegados de forma independiente, con un único y definido propósito.

El despliegue independiente ofrece muchas ventajas, como: Desarrollo en diferentes lenguajes de programación, escalamiento independiente de otros servicios e implementación en el hardware que mejor se adapte a sus necesidades. Además, son más fáciles de mantener y más tolerantes a fallas. Cada servicio puede ser desarrollado, implementado y probado de forma independiente por un equipo de desarrollo diferente utilizando variedad de tecnologías [19].

Los microservicios se caracterizan por su descentralización, característica que permite la división de servicios, en donde, cada servicio tiene una ejecución independiente y esto permiten el uso de interfaces de programación de aplicaciones (API), que es una parte del sistema que maneja la comunicación del servicio para acceder a los datos o llamar a alguna funcionalidad, además es una de las partes importantes en el arquitectura de microservicios [20].

Los patrones en los microservicios son elementos esenciales para la construcción de software desde la inclusión de un patrón en la arquitectura proporciona una solución confiable a un problema, ayuda a satisfacer uno o más atributos de calidad, e incluso mejora la comunicación a través del equipo de desarrollo. Sin embargo, una desventaja del uso de esta tecnología es la identificación de patrones disponibles en la tecnología de microservicios, debido a que es una tarea difícil ya que la información es escasa [21].

Es una unidad desplegable independiente, con un alcance bien definido alrededor del negocio, que opera bajo la comunicación basada en mensajes con otros servicios [22]. Los microservicios hacen que la entrega de aplicaciones sea más simple y eficiente.

El hecho menos enfocado es que esto se hace en el costo de aumentar la complejidad de la capa de infraestructura. La arquitectura de microservicios es una arquitectura de software que se orienta al desarrollo de software altamente automatizado, evolutivo y alineados a los requerimientos del negocio [23].

Principios

Los principios fundamentales de esta arquitectura son [6]:

- Responsabilidad única por servicio.
- Los microservicios son autónomos y de despliegue independiente.
- Los servicios son ciudadanos de primera clase: exponen el servicio con base en a interfaces que ocultan los detalles de implementación.

Enfoque arquitectónico

La orientación de la arquitectura de microservicios se encuentra orientada en tres aspectos claves [22]:

- ***Ideales para grandes sistemas***

El diseño bajo microservicios está pensado para abordar grandes sistemas, tomando ventaja sobre el diseño modular de los servicios y dedicar cada elemento del sistema (servicio), a realizar una y solo una tarea de la mejor manera [17]. Desde el exterior, un solo microservicio se trata como una caja negra (bajo el principio de ocultación de información), con funcionalidades limitadas e interfaces de comunicación muy bien definidas [22], [24].

- ***Arquitectura orientada a los objetivos***

La arquitectura de microservicios trata de resolver un objetivo particular en un contexto definido, utilizando un enfoque en particular de comunicación distribuida entre sistemas.

A menudo se lo vincula con ciertas técnicas y practicas comunes, que son ideales para el desarrollo de sistemas bajo este patrón, más sin embargo el núcleo de este estilo de desarrollo de software es el de lidiar con la magnitud de los sistemas bajo la estrategia "divide y vencerás" [22]. El sistema de software, la organización y la forma de trabajar se optimizan holísticamente para lograr un objetivo [25].

- ***Arquitectura altamente reemplazable***

Uno de los puntos fuertes de los microservicios es la idea de impulsar el reemplazo de componentes por sobre su mantenibilidad, gracias a que puede escalar de forma horizontal, los microservicios pueden mantener el sistema a flote [22], aunque una de sus partes llegara a quedar inaccesible, siempre y cuando exista otro microservicio con las mismas características, que supla las necesidades del anterior.

Características claves

Con ayuda de la arquitectura de microservicios en las grandes aplicaciones, según [26], pueden descomponerse en un conjunto de aplicaciones pequeñas, que pueden desarrollarse, implementarse, expandirse, administrarse y monitorearse de forma independiente. Características como la resiliencia, escalabilidad, entrega continua y el uso eficiente de recursos, son las necesidades que tiene por objetivo suplir este patrón de arquitectura.

Diseño orientado al negocio

El principio por el cual se basa el diseño basado en el dominio (Domain-Driven Design) es un concepto recurrente usado en el contexto de microservicios, para definir los límites de cada unidad (microservicio) y modelarlo entorno al negocio [27]. Con este modelo de implementación se consigue un alto grado de tolerancia al cambio, al agregar o mantener funcionalidades de un sistema pequeño y delimitado por pequeños grupos de desarrollo.

Contextos acotados

Los contextos acotados, identifican los límites, incluidas las entradas, salidas, eventos, requisitos, procesos y modelos de datos; permiten la implementación y la ejecución en tiempo de ejecución de diferentes partes del sistema sin corromper los modelos de dominio independientes presentes en ese sistema [24], [25].

La capa de comunicación de datos ayuda a que las implementaciones de microservicios sean lo suficientemente flexibles para adaptarse a cambios dentro del dominio comercial y contextos acotados que aún no han sido definidos [27].

El enfoque orientado a los dominios de aplicación ganó relevancia y popularidad con la aparición de los microservicios. Esto debido a que su conjunto de prácticas que permite modelar sistemas complejos siguiendo el concepto de contextos acotados, asegurando que los microservicios diseñados están débilmente acoplados [28].

Si los límites de nuestro servicio se alinean con los contextos acotados en nuestro dominio, y nuestros microservicios representan esos contextos acotados [22], estamos en un excelente comienzo para garantizar que nuestros microservicios estén débilmente acoplados y fuertemente cohesivos.

Despliegue independiente

Los microservicios están diseñados para ser independientes, la idea de esto es apuntar a la gran confiabilidad del software, donde, cambiar alguna funcionalidad interna de un microservicio e implementarlo en un entorno productivo, sin necesidad de actualizar cualquier otro microservicio.

Con este estilo arquitectónico en caso de que ocurra un problema, puede ser rápidamente aislado a un servicio individual, lo que hace que la reversión rápida sea fácil de lograr. También significa que podemos ofrecer nuevas funcionalidades a los clientes más rápidamente [24]. Esta característica se obtiene gracias al bajo nivel de acoplamiento que exige el patrón.

Esta es una gran ventaja modular por sobre otros patrones arquitectónicos, sin embargo, la heterogeneidad de las técnicas o tecnologías que se usen en un microservicio supone un arma de doble filo, debido a la compleja ejecución que conlleva mantener técnicas, tecnologías y metodologías diferentes [29].

Escalabilidad y flexibilidad

Los microservicios son concebidos como entidades separadas que trabajan juntas para realizar las mismas tareas, enfatizando características de flexibilidad y escalabilidad del software [30].

En términos de flexibilidad, conforme con [22], la creación de servicios bajo este patrón, son realizados en periodos muy cortos, en comparación con servicios convencionales, proveyendo la capacidad de reducir el costo de diseño, implementación y despliegue. La reducción de costos puede aumentar su agilidad porque hace que sea más probable que se experimente con nuevas ideas y estrategias de negocio.

Del mismo modo, la escalabilidad es una característica del software que sobresale en este tipo de arquitectura, en [24] se expone que, con un monolito grande, cuando existen necesidades de escalamiento, es necesario escalar todo junto debido al acoplamiento inherente de esta arquitectura. Por otro lado, con ayuda de pequeños servicios, se pueden escalar independientemente según las necesidades del dominio.

Los microservicios ofrecen diferentes ejes de escalado, que deben ser cuidadosamente seleccionados según su aplicación, los cuáles se detallan en la Tabla 4.

Tabla 4. Eje de escalabilidad.

Ejes de escalabilidad	
Escalado vertical	Incrementar la capacidad (recursos) computacional.
Duplicación horizontal	Tener diferentes implementaciones que ejecuten las mismas funciones.
Particionamiento de datos	Dividir el trabajo en función de algún atributo de los datos
Descomposición funcional	Separación del trabajo según el tipo, por ejemplo, descomposición de servicios.

El segundo eje en particular, permite automatizar el proceso de escalamiento, herramientas de computación en la nube como AWS (Amazon Web Services), Azure y Google Cloud Platform, permiten encender o apagar instancias duplicadas en un proceso denominado autoscaling; que, según la demanda establecida en horarios donde el negocio podría experimentar aumentos de carga en solicitudes [24]. Acorde a [31], también existen soluciones de autoscaling modernas aplicadas en aplicaciones en la nube que emplean umbrales de carga, para auto escalar aplicaciones según esta métrica.

Cohesión

Podría decirse que los microservicios priorizan la alta cohesión en términos de funcionalidad empresarial, por sobre la cohesión en la funcionalidad técnica [16].

Se define la cohesión como: "el código que cambia junto, permanece junto" [25], [29]. Alineando esta definición en el contexto de microservicios, se busca principalmente la facilidad de realizar cambios en la funcionalidad del software, por lo que se busca que la funcionalidad se agrupe de tal manera que podamos realizar cambios en la menor cantidad de lugares posible, esto implica que las modificaciones se realicen en la menor cantidad de servicios posibles, para mantener el bajo costo del cambio.

Acoplamiento

El acoplamiento está relacionado con el principio de ocultación de información, explicado en [29], cuantas más cosas estén "acopladas", más tendrán que cambiar juntas. Una arquitectura de microservicios, que tiene la ventaja de un acoplamiento flexible, se puede dividir en varias partes independientes y cada parte funciona de forma independiente [32]. Consecuentemente, en [25],

menciona que se debe buscar la "independencia" de los microservicios, de modo que una modificación de código en uno de ellos no tenga efectos dominó en los demás.

Los servicios débilmente acoplados, no deben implicar modificaciones en servicios conectados, Un servicio débilmente acoplado sabe tan poco como necesita sobre los servicios con los que colabora. Las conexiones entre contextos acotados deben acoplarse débilmente, ya que los cambios realizados dentro de un contexto acotado deben minimizar o eliminar el impacto en los contextos vecinos [27].

Resiliencia

El termino resiliencia, habla sobre el manejo y soporte de errores, se deben manejar de forma elegante y evitar el bloque de todo el sistema, incluso cuando ocurren errores [22], [33].

El objetivo de lograr una arquitectura resistente a errores mediante microservicios se basa en:

- Replicación. Evitando así puntos únicos de falla.
- Contención y aislamiento. Evitar la propagación de errores en cascada, el consumidor de un servicio es responsable de sus propios errores.

Los 4 aspectos claves relacionados con los sistemas resilientes [24], se exponen en la Tabla 5.

Tabla 5. Aspectos resilientes.

Aspectos resilientes	
Robustez	La capacidad de responder frente a excepciones esperadas.
Rebote	La capacidad de recuperarse después de un evento catastrófico.
Extensibilidad elegante	Qué tan bien lidiamos con una situación o evento inesperado.
Adaptabilidad sostenida	La capacidad de adaptarse continuamente a entornos, partes interesadas y demandas cambiantes.

En el trabajo abordado en [34], establece que la creación de un sistema resiliente se facilita al reducir la complejidad y el tamaño, que es exactamente uno de los principios que rigen el patrón de microservicios, al instaurar pequeñas piezas de software, en contextos delimitados. En colaboración con tecnologías actuales, en [32], se aborda que una arquitectura basada en

microservicios, en combinación con contenedores, hace posible "empaquetar" aplicaciones y contextos en componentes pequeños y desechables, proporcionando un gran potencial para la mejora de la resiliencia de los sistemas de software.

Pertinencia de la arquitectura de microservicios

Las organizaciones privadas fueron pioneras en implementar nuevas soluciones para sus negocios, sin embargo, tomar decisiones arquitectónicas y de infraestructura suelen ser sensibles.

Implementar y mantener microservicios implica desafíos tecnológicos y organizacionales [22], esto hace cuestionarse a las empresas si una colección de servicios pequeños, débilmente acoplados y de implementación independiente son realmente rentable para la empresa.

Beneficios

Las empresas analizan cuidadosamente, los beneficios que ofrecen los microservicios, según [22], los diversos directivos tecnológicos de empresas reconocidas como Amazon, Gilt, Hootsuite, Disney, SoundCloud, coinciden al incidir en que los microservicios:

- Acelera el proceso de desarrollo e implementación de código.
- Permite la heterogeneidad de las tecnologías, lenguajes, frameworks basada en las necesidades.
- Promueve la innovación tecnológica mediante código desechable.
- Promueve la atomicidad de servicios.
- Aumenta la flexibilidad del software frente a los cambios del negocio y sus necesidades.
- Crea software y servicios más eficientes y comprobables.

Los microservicios apuntan a la colaboración entre equipos en grandes grupos de desarrollo [6], [24], en donde pequeñas startups podrían encontrar problemas para mantener servicios diversos.

El carácter independiente de la implementación abre nuevos modelos para mejorar la escala y la solidez de los sistemas [29]. En aplicaciones como servicio, son una buena opción de implementación, ya que requieren de una alta disponibilidad en la mayoría de las aplicaciones de este tipo, además de que poseen varias vías de escalamiento según las necesidades. Además, se hace énfasis en la interoperabilidad de los sistemas, con la creciente computación en la nube, los sistemas que contengan procesos más críticos pueden ser migrados a servicios con un poder de cómputo mayor, y ubicar servicios más triviales en servidores locales.

Desventajas

Los microservicios pueden parecer una gran solución e innovación tecnológica a primera vista, sin embargo, existen contextos en los que pueden ser mucho más contraproducentes y en su gran mayoría relacionadas con su naturaleza distribuida, tal cual se mencionan en [6], [24], [29]:

- Agrega complejidad en la comunicación de sistemas.
- Aumenta complejidad y reduce la manejabilidad de los datos.
- La latencia y disponibilidad necesitan ser observadas para asegurar la calidad de los servicios.
- La heterogeneidad de tecnologías, lenguajes y frameworks puede ser contraproducente para la comunicación entre equipos.
- Complejidad de la trazabilidad alrededor de los servicios.
- Requiere de personal capacitado en el uso de tecnologías afines (pod, Kubernetes, Docker).
- Complejidad de pruebas de todo el sistema (End-to-End).

Los microservicios traen consigo un grado significativo de complejidad, explicando en [24], ya que implementarlos conlleva un riesgo ya que esta complejidad debe ser justificada. Solo cuando los conceptos básicos de los microservicios se entienden correctamente, puestos en pie e implementados, pueden ayudar a crear arquitecturas robustas y eficientes que pueden ayudar a que los sistemas se conviertan en más que la suma de sus partes.

Tecnologías y herramientas

Logging

Logging o archivo de registro son un tipo de archivos conocidos como logs los cuales permiten capturar los eventos que son proporcionados por el sistema en tiempo de ejecución con el propósito de realizar:

- Un análisis posterior para detectar errores en el sistema, violaciones de seguridad, etc.
- Los archivos de registros también se los utiliza para detectar vulnerabilidades existentes en los mecanismos de seguridad preventivos y mejorar esos mecanismos.
- El uso de archivos de registro es de gran ayuda en sistemas concurrentes con microservicios ya que como que se puede tener varios microservicios ejecutándose y comunicando entre

sí y en caso de que uno o varios microservicios falle será muy complicado detectar el error, por lo cual los archivos de registro permitirán detectar el error de una forma muy eficaz.

Contenedores y Kubernetes

La virtualización en contenedores es una tecnología que proporciona un entorno de tiempo de ejecución virtual aislado y ligero. En [35] se explica qué, está tecnología permite que múltiples espacios de contenedores se ejecuten en una máquina física mediante la virtualización del kernel del sistema operativo en lugar del hardware físico.

Además, diferentes contenedores pueden compartir los mismos recursos físicos, pero desde el punto de vista de una aplicación alojada, cada contenedor tiene su propio sistema operativo autónomo que se ejecuta de forma independiente.

Docker es una tecnología de virtualización de contenedores más populares utilizadas en la práctica del desarrollo de software y permite empaquetar componentes de software en imágenes de Docker, que luego se exportan como plantillas de solo lectura para crear y ejecutar contenedores, permitiendo un mejor rendimiento de la aplicación en tiempo de producción [36].

La orquestación de contenedores se utiliza para automatizar la implementación y la administración de aplicaciones en contenedores a gran escala.

Docker Swarm y Kubernetes, [36], son actualmente las tecnologías más populares utilizadas para el orquestamiento y creación de contenedores, ya que proveen de las herramientas necesarias para escalar, cargar, equilibrar e interconectar.

Con Docker, los microservicios se implementan en un entorno contenedorizado, donde cada microservicio se encapsula en un contenedor [37], mientras que Kubernetes es una herramienta para orquestar contenedores y administrar la implementación de aplicaciones.

Streaming

La arquitectura de microservicios para sistemas o plataformas de streaming como de películas, música, transmisión en vivo, transmisión de datos, predicciones contextuales en tiempo real, etc, tienen la capacidad de procesar una gran cantidad de datos heterogéneos de forma rápida en tiempo real [38].

Como tecnología de streaming para microservicios se tiene a Apache Kafka, según [39], es una es una plataforma middleware de transmisión y procesamiento de eventos distribuidos tolerante a fallas, que permite la canalización de datos de alto rendimiento, análisis de transmisión de datos e implementación de aplicaciones de tiempo real.

La función de Kafka es procesar y organizar los datos que se le pasan, los datos se almacenan en temas que representan nombres de categorías donde se publican y consumen los mensajes, donde cada tema se puede dividir en una o más particiones , y cada partición es una secuencia inmutable y editada a la que se agregan constantemente nuevos registros [39], [40].

La arquitectura de Apache Kafka consta de un clúster de Kafka, un servidor Zookeeper, además de un productor y un consumidor [40]. El productor de Kafka y el consumidor de Kafka son aplicaciones en las que una aplicación transmite mensajes (productor) y la otra recibe mensajes del emisor (consumidor).

Performance

El rendimiento de los microservicios es un punto crítico en las aplicaciones basadas en microservicios. Un sistema con microservicios donde el tráfico de los datos aumenta de forma constante y donde cada microservicio debe poder escalar con todo el sistema sin sufrir problemas de rendimiento.

Para garantizar el rendimiento de la arquitectura de microservicios se deben aplicar ciertas métricas que permitirán medir el rendimiento de la arquitectura [26]:

- El número de veces que se llama a un microservicio.
- El número de veces que un microservicio no responde o supera el límite de tiempo.
- El tiempo de ejecución del servicio invocado.
- El número de llamadas de métodos entre todos los pares de microservicios extraídos. Cuanto más pequeño es el número de interacción mejor es la calidad de los microservicios candidatos, ya que un número de interacción bajo, refleja un acoplamiento débil.
- El número de solicitudes de prueba enviadas al sistema o microservicios.
- El tiempo máximo para una solicitud (salida) realizada de un microservicio a otro.
- El tiempo máximo de respuesta es el de una llamada (entrada) o solicitud al sistema o microservicio. Es el momento de procesar una respuesta a otro microservicio.
- El número de paquetes enviados al sistema o microservicio.

Testing con microservicios

Unit Test

Las pruebas unitarias son útiles para medir la funcionalidad de los microservicios, donde se testea una pequeña parte de la funcionalidad del microservicio [41]. Las pruebas unitarias invocan al código del microservicio que está siendo testeado y, por lo general, involucran solo una parte del microservicio, esto se realiza de forma aislada e independiente y compara lo real con los resultados esperados para garantizar un funcionamiento adecuado y detectar fallas tempranas en el código que pueden ser más difíciles de encontrar en etapas posteriores.

Este tipo de prueba se las puede aplicar en diferentes patrones arquitectónicos [41], como, por ejemplo, en una arquitectura monolítica, donde se aíslan los componentes utilizables más pequeños de una aplicación y se prueban uno por uno para garantizar que todo funcione por sí solo antes de integrarse con la aplicación como un todo. Sin embargo, con los microservicios, es un poco más fácil de hacer [42].

Service Tests

Las pruebas de servicio están diseñadas para testear los microservicios de forma directa evitando el uso de la interfaz de usuario.

Para un sistema que comprende varios microservicios, una prueba de servicio probaría las capacidades de un microservicio individual [24]. Al ejecutar las pruebas en un solo microservicio de esta manera, se tiene una mayor confianza en el comportamiento del servicio, pero aún se mantiene el alcance de la prueba aislada.

Algunas de estas pruebas pueden ser tan rápidas como las pruebas unitarias de pequeño alcance, pero si decide realizar la prueba en una base de datos real, los tiempos de la prueba pueden aumentar [24]. Estas pruebas también cubren más alcance que una simple prueba de unidad, por lo que cuando fallan, puede ser más difícil detectar el error que con una prueba de unidad.

End-to-End Tests

Las pruebas de extremo a extremo son pruebas que se realizan con todo el sistema, donde se utiliza la interfaz gráfica del sistema mediante un navegador web verificando que funcione como se

esperaba de principio a fin. Este tipo de prueba toma mucho tiempo en realizarse y es dificultoso realizarse y mantenerse en un entorno de microservicios.

Estas pruebas cubren una gran cantidad de código de producción, y si la prueba sale correcta se tienes un alto grado de confianza [24], asegurando que el código o parte funcional del sistema que se está testeando funcione en producción.

Control de concurrencia

El control de concurrencia es uno de los aspectos principales que ayuda a superar los problemas de rendimiento de un sistema. Con el crecimiento de los usuarios en las aplicaciones, la gestión de los conflictos en un entorno multiusuario exige el empleo de control de concurrencia eficaz [43]. Los sistemas modernos han tenido un gran impacto en el control de la concurrencia, debido a que deben sincronizarse con el almacenamiento y procesamiento de datos.

El control de concurrencia permite evitar los conflictos de los procesos concurrentes en los aplicativos modernos, ya que estos sistemas son dinámicos y se ejecutan en tiempo real, además mejora los procesos importantes de los sistemas.

Metodología SEED(S)

SEED(s) es una metodología repetible y confiable para diseñar interfaces de servicios con el propósito de aumentar la flexibilidad y usabilidad de estas interfaces a través de un buen diseño que pueda tener un profundo impacto en la robustez de la arquitectura del sistema y en la productividad del desarrollador [25]. Esta metodología es útil al momento de diseñar microservicios y puede ser eficaz en el diseño de cualquier tipo de servicios, como APIs RESTful o GraphQL, que son desarrollados para interfaces de usuario (IU).

Los pasos de esta metodología son las siguientes [25]:

Paso 1. Identificación de los actores

El primer paso de modelado de la metodología SEED(S) se debe comenzar con el diseño de servicios identificando actores clave del dominio, para así lograr un alcance centrado en el cliente de las capacidades representadas por los servicios. Los actores son las entidades de la arquitectura de microservicios, como ejemplo: Persona, usuario, cliente, vendedor, etc.

Paso 2. Identificar las tareas que deben realizar los actores

Una vez identificado los actores, se deben identificar las acciones, tareas que estos realizan. Para identificar las tareas, se recomienda usar el formato de historias de trabajo (JTBD), donde para cada uno de los actores que se identifican, se necesita descubrir los mejores JTBD para ese actor y así asegurar que los datos clave estén bien documentados.

La metodología SEED(S) utiliza historias de trabajo, el cual tiene el siguiente formato:

Cuando **<una circunstancia>,** quiero **<motivación>**, para poder **<objetivo>**

A continuación, se muestra un ejemplo del uso de JTBD:

"Cuando los planes de Emma cambian y no puede viajar en un vuelo previamente reservado, quiere reprogramar fácilmente su vuelo, para poder obtener un vuelo que funcione para sus nuevos planes".

De esta historia de trabajo se obtiene la tarea de "reprogramar vuelo".

Paso 3. Descubrir los patrones de interacción usando diagramas de secuencia

La metodología SEED(S) recomienda emplear diagramas de secuencia de lenguaje de modelado unificado (UML) para el tercer paso.

En el diagrama de secuencia se debe diagramar los autores y las tareas que realizan, haciéndolo en forma de flujo la secuencia que realiza el actor con dicha actividad. A continuación, en la Figura 8 se muestra un ejemplo de diagrama de secuencia del flujo.

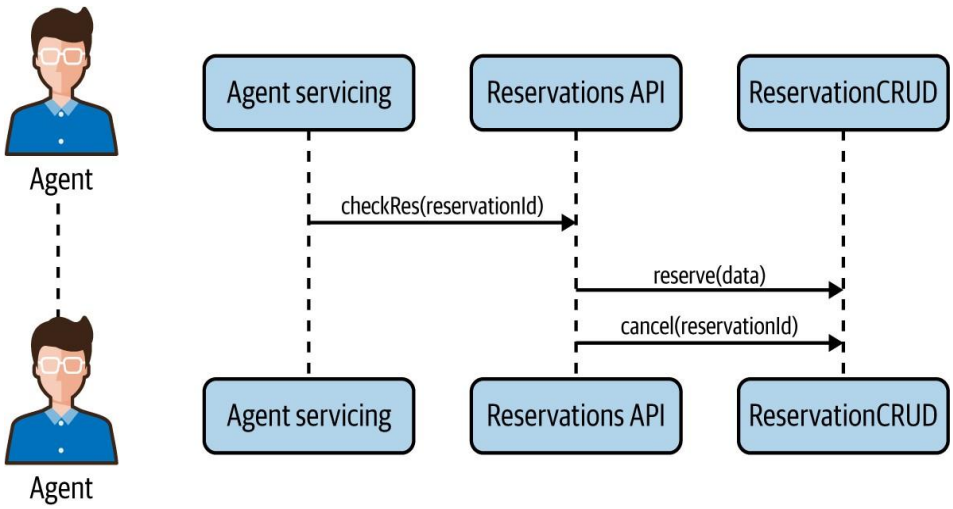


Figura 8. Ejemplo de diagrama de secuencia.[25].

El “Agent” es el actor, el “Agent Servicing” es la interfaz de usuario, como puede ser la página web o aplicación móvil, Reservations API es una API REST que la aplicación invoca y reservationCRUD es una de las los microservicios que alimentan dicha API y por último checkRest(reservationId), reserve(data), cancel(reservationId) son las actividades, tareas que realiza el agente.

Una vez que se tiene los diagramas de secuencia las interacciones, se puede capturar los requisitos técnicos para el microservicio o una API, en forma de un conjunto de acciones y consultas descritas utilizando una sintaxis estándar.

Paso 4. Derivar acciones y consultas de alto nivel basadas en los trabajos a realizar (JTBD) y los patrones de interacción

En este paso se de utilizar el principio command query separation (CQS) que significa comandos y consultas para modelar el lado de acción de los servicios por separado del lado de la consulta y documentar cada uno con su propio formato estándar.

Las consultas son búsquedas con entradas y salidas definidas, las acciones son solicitudes que causan algún tipo de modificación del estado. Al igual que en las historias de trabajo, se recomienda utilizar un formato estándar para capturar consultas y acciones, donde el formato de las consultas es el siguiente:

Esta es una descripción de una consulta.

- Entrada: Lista de variables de entrada.
- Salida: Lista de elementos de datos de salida.

Del mismo modo, el formato de las acciones es el siguiente:

Esta es una descripción de una acción.

- Entrada: Lista de variables de entrada.
- Salida esperada: Descripción del efecto secundario inducido.
- Salida (opcional): Lista de elementos de datos de la salida (si es que hubiera).

A continuación, se muestra un ejemplo de una historia de trabajo en donde se extrae las consultas.

Riley que quiere encontrar un vuelo que coincida con los requisitos de comodidad de viaje de su familia, indicando detalles preferencias tales como: número de asientos adyacentes, número máximo de conexiones, etc. Para satisfacer las necesidades de dicho trabajo, necesitamos un contrato de consulta que permita la indicación de todas las preferencias como entradas a la consulta de búsqueda. Por lo tanto, la definición de la consulta puede tener el siguiente aspecto:

Consulta: Búsqueda de vuelos

- **Entrada:** departure_date, return_date, origin_airport, destination_airport, number_of_passengers, baby_friendly_connections, adjacent_seats, max_connections, minimum_connection_time, max_connection_time, order_criteria [objeto], customer_id.
- **Salida:** Lista de vuelos que cumplen los criterios.

Un ejemplo de una acción es el siguiente:

Acción: Volver a reservar un viaje

- **Entrada:** original_reservation_id, new_flight_id, seat_ids
- **Salida esperada:** Nuevo vuelo reservado o error devuelto; Si el nuevo vuelo se reserva correctamente, se cancela el antiguo.
- **Salida:** Un mensaje de éxito o un mensaje de error detallado.

Acción: Cambiar asiento

- **Entrada:** reservation_id, customer_id, requested_seat_ids
- **Salida esperada**: Nuevo asiento reservado o error devuelto; Si el asiento está disponible y el viajero está calificado, el asiento antiguo es cancelado y el nuevo asiento termina reservándose correctamente.
- **Salida**: Un mensaje de éxito o un mensaje de error detallado.

Paso 5. Describir cada consulta y acción como una especificación con un estándar abierto

En este paso se tomará la definición de una acción que se describe en la fase y se diseñará un endpoint RESTful utilizando el estándar de especificación OpenAPI (https://oreil.ly/JoiGg). Para realizar este diseño se puede utilizar el editor de Visual Studio Code con la extensión de OpenAPI Designer. A continuación, se muestra un ejemplo para la acción de cambio de reserva. En la Figura 9 se muestra un ejemplo de la estructura de la extensión utilizando el formato YML con su respectiva salida renderizada.

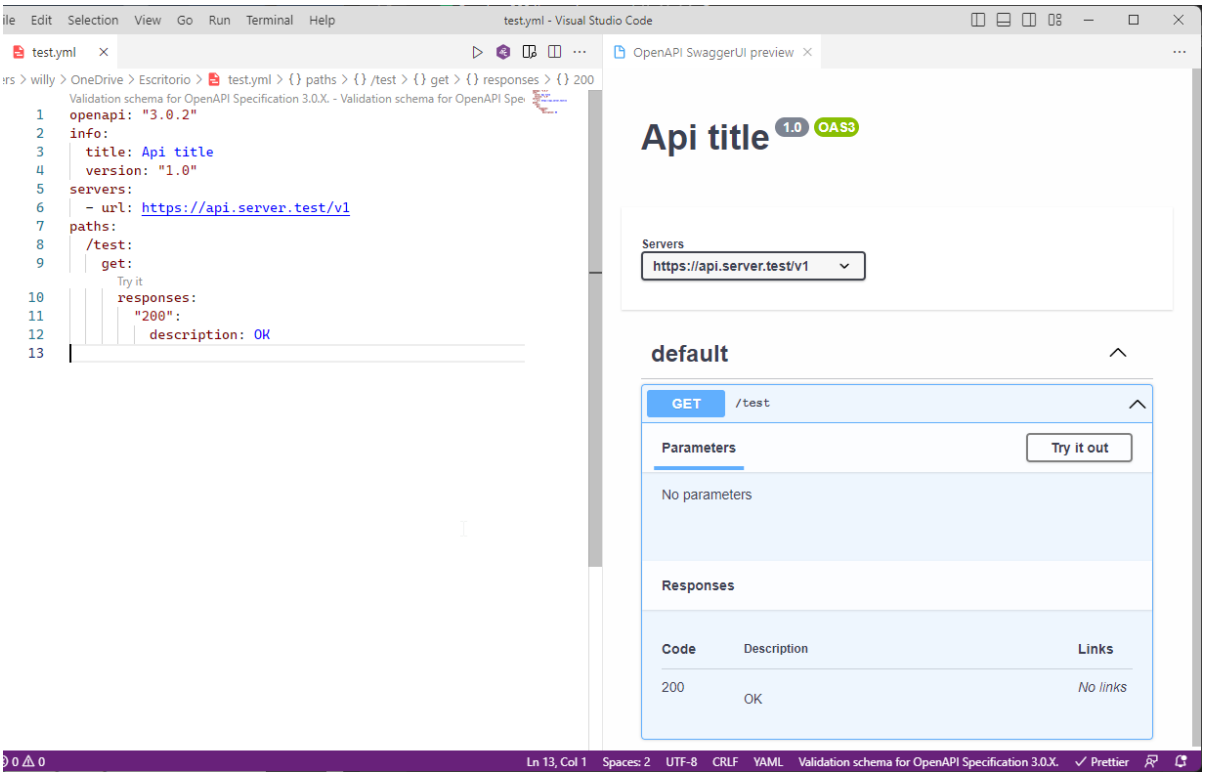


Figura 9. Ejemplo de representación de un documento de muestra de la especificación OpenAPI (OEA).

Paso 6. Obtención de comentarios sobre la especificación de la API

En este paso se debe recopilar comentarios sobre los diseños de los servicios, ya que el diseño del servicio no se realiza hasta que se presenta al público objetivo para el servicio y los comentarios se recopilan y se aplican a los diseños iniciales. Se debe tener en cuenta dos grupos al momento de diseñar los servicios y la API.

- Usuarios finales del sistema.
- Desarrolladores que codificarán los servicios (API o microservicios).

Al comienzo de la metodología SEED(S), se entrevista a los usuarios finales para recopilar y comprender las historias de trabajo relevantes para ellos. Sin embargo, más adelante en el proceso se comienza a recibir comentarios de los desarrolladores. Esto puede suceder tan pronto como la fase de diseño de las interacciones, y luego nuevamente una vez que se produce la OEA, antes de codificar.

Los desarrolladores de la API, deben ser entrevistados para probar la usabilidad de los diseños, para evitar codificar algo que pueda terminar siendo rechazado por ellos debido a la mala usabilidad.

Paso 7. Implementación de microservicios

El último paso de la metodología SEED(S) es la implementación de microservicios, consiste en la codificación de la funcionalidad que se diseñó inicialmente.

1.3.2. Aplicación web

Una aplicación web o web app es un programa, sistema o software que se puede usar mediante el uso del internet, este tipo de sistema se utiliza cada vez más para mostrar información en tiempo real a los usuarios con el uso de una interfaz gráfica que es accedida a través de un navegador.

Las aplicaciones web son desarrolladas por una variedad de usos, ya sea consulta de información, automatización de procesos, procesos transaccionales, etc., ya sea para una persona o una organización, además combinan información de diferentes fuentes, como servicios, recursos estáticos, datos obtenidos de una base de datos, sensores, etc. y todo esto lo hace en tiempo real [44].

Lenguaje de programación

Para de un sistema web, móvil o escritorio u otro sistema funcional se necesita de un lenguaje de programación [45]. Un lenguaje de programación es un lenguaje que se usa para el diseño y desarrollo de un sistema, las cuales son instrucciones en formato de código que son interpretados por la máquina.

Los sistemas se crean utilizando un lenguaje de programación para controlar el comportamiento y la salida de las máquinas a través de algoritmos precisos, similares al proceso de comunicación humana.

Algunos ejemplos de lenguajes programación populares son: Java, C++, C#, Python, JavaScript, TypeScript, etc, de los cuales JavaScript, TypeScript son muy populares en el área de desarrollo web.

JavaScript

JavaScript es un lenguaje de programación desarrollado originalmente por Netscape Communications Corporation, es muy popular en el desarrollo de aplicaciones web, tanto en el lado frontend como en el backend [46]. Aunque este lenguaje está basado en prototipos, los desarrolladores pueden emular las abstracciones basadas en clases para dominar la complejidad

creciente de las aplicaciones. Además se utiliza para muchas aplicaciones del lado del servidor web, así como aplicaciones independientes en dispositivos portátiles y de escritorio, las aplicaciones independientes de Javascript se basan en infraestructuras PhoneGap [47].

La identificación de clases en el código JavaScript heredado puede ayudar a los desarrolladores en las siguientes actividades [46]:

- Comprensión del programa.
- Migración a la nueva sintaxis de JavaScript que admite clases.
- Implementación de herramientas de apoyo.

JavaScript es procesado por el navegador web y no por el servidor web, es decir que la funcionalidad se puede ejecutar después de que se haya cargado la página web sin conectarse al servidor, además tiene un modelado universal para el diseño único de sistemas distribuidos complejos que abordan todos los niveles y aspectos para implementación de los sistemas [48].

Algunas características de JavaScript son las siguientes [49]:

- Lenguaje de script centrado en objetos
- Es una tecnología para desarrollo de aplicaciones web.
- Permite validaciones de la entrada del usuario.
- Permite realizar declaraciones else y if.
- Es centrado en el intérprete.
- Capacidad para realizar la función incorporada.
- Peso ligero y delicado.
- Permite utilizar bucle de instrucciones.
- Permite el manejo de eventos.

Pero actualmente también se utiliza este lenguaje de programación para el lado servidor, permitiendo el desarrollo de APIs con Node.js debido a su gran rendimiento para desarrollar aplicaciones escalables [50].

TypeScript

El lenguaje de programación TypeScript es una extensión de JavaScript que fue desarrollado por Microsoft, este lenguaje es fuertemente tipado y muy utilizado para desarrollar aplicaciones a gran escala en la actualidad [51]. Las aplicaciones suelen ser desarrolladas aplicando programación

orientada a objetos (POO), con complejas jerarquías de clases o interfaces y tienen comportamientos altamente polimórficos.

Con TypeScript se aplica programación estructurada, estática, fuerte, mayormente inferido, y gradualmente tipificado, por esto es que es exitoso entre los otros lenguajes de programación, además todo programa JavaScript también es un programa TypeScript, debido a que si se tiene un archivo JavaScript que termina con.js se le cambia el nombre de la extensión .ts, y este compila [52].

TypeScript es un superconjunto sintáctico de JavaScript y conserva su semántica, las cuáles proporciona abstracciones de programa robustas, como clases e interfaces, para desarrollar aplicaciones JavaScript a gran escala, además las aplicaciones de TypeScript a menudo usan bibliotecas de JavaScript no modulares [53]. Un sistema programado con TypeScript, su código fuente se transpila a JavaScript plano que luego se ejecuta en una máquina virtual de JavaScript, donde las máquinas virtuales (VM) comerciales modernas de JavaScript, como V8, ejecutan el código JavaScript transpilado con su intérprete que recoge los tipos y la información de perfil.

TypeScript tiene una variedad de características, las cuáles son [51]:

- Si se conoce el lenguaje de programación de JavaScript, se puede programar con TypeScript.
- Permite la programación orientada a objetos.
- Permite utilizar funciones de orden superior y genéricos, que son un reto para las pruebas automatizadas.
- Tiene una escritura estática fuerte.
- Permite utilizar bibliotecas de JavaScript.
- Permite la manipulación de DOM.
- Se lo puede compilar en cualquier navegador.

YML

YML o YAML es un lenguaje o formato de serialización de datos, este lenguaje funciona de forma correcta con los lenguajes de programación modernos para realizar tareas cotidianas. Además, tecnologías como Docker utiliza la extensión YAML, esto se puede evidenciar en el archivo Docker compose el cual es un archivo YAML que permite definir los servicios, redes y volúmenes para una aplicación Docker y luego pueda ser ejecutado con un solo comando [54].

Además, YAML es un formato de texto, en la cual se puede definir clases de objetos de datos y la sintaxis como una cadena. El motor de representación YAML es una herramienta que permite convertir la información entre estas representaciones adicionales.

Base de datos

Las bases de datos es una de las tecnologías más que antiguas que se siguen utilizando en las aplicaciones modernas. Las bases de datos pueden ser usadas mediante un sistema de gestión de bases de datos (SGBD) [55], los cuáles ofrecen esquemas relacionales modernizados diseñados para aplicaciones transaccionales, así como sistemas no relacionales y tecnologías orientadas al flujo de datos para un análisis rápido.

Los usuarios disponen de una amplia gama de tecnologías de SGBD para que mejor se adapten a sus necesidades de aplicaciones en línea, sociales, móviles, analíticas, integradas, en tiempo real y en la nube.

Los SGBD se han aplicado con éxito en varias aplicaciones, ya que han demostrado ser una solución eficaz para los requisitos de gestión de datos en organizaciones grandes y pequeñas [56]. Este enfoque de base de datos satisface las necesidades de las empresas que tratan con conjuntos de datos estáticos y de consulta intensiva si estos conjuntos de datos son de naturaleza relativamente pequeña.

Algunas bases de datos populares son las siguientes: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB, IBM DB2, Elasticsearch, Redis, SQLite, etc.

PostgreSQL

PostgreSQL es un sistema de base de datos relacional, es de código abierto, donde su lenguaje de programación o funcionalidad es el estándar SQL para gestionar los datos almacenados [57]. En PostgreSQL las entidades de las bases de datos se las trabaja con tablas, las cuáles almacenan los datos.

Las bases de datos PostgreSQL se pueden utilizar en computación en la nube , además es un SGDB fácil de usar, permite utilizar funciones y consultas SQL, de requiere una red distribuida de computadoras, permite la rápida integración de los datos y realiza las consultas de una forma eficaz [58]. Esta base de datos puede trabajar en conjunto con otras tecnologías, por ejemplo, realizar la

comunicación de los datos con una API, los datos ser consumidos en un Frontend, deployar los servicios con Docker, etc [59].

Cada tabla es una colección de filas con nombre cada fila de una tabla determinada tiene el mismo conjunto de columnas con nombre y cada columna es de un tipo de datos específica. Mientras que las columnas tienen un orden fijo en cada fila, es importante recordar que SQL no garantiza el orden de las filas dentro de la tabla de ninguna manera [60]. Las tablas se agrupan en bases de datos y una colección de bases de datos administradas por una única instancia de servidor PostgreSQL constituye un clúster de bases de datos.

PostgreSQL tiene muchas características avanzadas que ofrecen otros sistemas de administración de bases de datos de clase empresarial, tales como [57]:

- Tipos definidos por el usuario.
- Herencia de tablas.
- Sofisticado mecanismo de bloqueo.
- Integridad referencial de clave externa.
- Vistas, reglas, subconsulta.
- Transacciones anidadas.
- Control de simultaneidad multi versión (MVCC).
- Replicación asincrónica.

MongoDB

MongoDB es una base de datos NoSQL más potente que existen en la actualidad, es de código abierto y orientado a documentos. La arquitectura de MongoDB se basa en documentos y colecciones, donde los datos se almacenan en los documentos en formato BSON.

Este tipo de base de datos tiene un sistema de replicación que proporciona una variedad de niveles de consistencia que permite a los desarrolladores seleccionar las compensaciones que quieren hacer cuando se trata de consistencia y latencia, a nivel de operación [57], [61].

La mayoría de desarrolladores utilizan este tipo de base de datos debido a las siguientes razones [62]*:*

- Los datos se almacenan en formato JSON binario, que es un par clave-valor, no se necesita realizar relaciones de tablas.

- Utiliza la memoria RAM para almacenar los datos; esto hace que el acceso a los datos sea más rápido.
- Soporta la propiedad ACID (atomicidad, consistencia, aislamiento y durabilidad) para realizar transacciones.
- Soporta la replicación; si el servidor primario se cae durante la transacción, entonces el servidor secundario maneja la transacción sin interacción humana.
- Es rentable porque reduce los costes de hardware y almacenamiento.
- Puede ahorrar una gran cantidad de datos que ayudarán a un procesamiento más rápido de las consultas.
- Los datos se almacenan en todos los nodos del clúster, por lo que no habrá ningún punto de fallo en el servidor de la base de datos.
- Debido al esquema dinámico, puede probar cosas nuevas a un coste menor.

Frontend

La tecnología frontend, es la tecnología usada para el desarrollo frontal de una página, aplicación o sistema, especialmente en el desarrollo web, es decir que es la tecnología que usa para diseñar y desarrollar los elementos visuales que los usuarios ven y usan para interactuar y así tengan una experiencia interactiva. El desarrollo frontend es la programación y gestión frontal de una aplicación para proporcionar.

Una aplicación frontend es manipulada, y accedida mediante el uso de un navegador, el cuál recibe archivos HTML, con diseños CSS, SCSS o SASS, con programación JavaScript o TypeScript, que manipulan el documento HTML y el DOM. Esto da lugar a aplicaciones interactivas donde las interacciones del usuario resultan en animaciones o cambios de la GUI, sin necesidad de actualizar la página. Para facilitar esto, el navegador expone una API llamada Document Object Model (DOM), que permite a los lenguajes de scripting acceder y manipular documentos HTML. Esta manipulación se realiza normalmente con los lenguajes de programación funcionales [63], [64].

Cuando los equipos de desarrollo empiezan a diseñar y crear nuevas aplicaciones de aplicaciones web, existe una variedad de arquitecturas, tecnologías y herramientas que son conocidos como framework, algunos de estos son: Angular, Vue.js, React, NuxtJS, Gatsby, etc.

Vue.js

Vue.js es un framework frontend desarrollado con JavaScript, el cual permite a los desarrolladores construir interfaces de usuario para sistemas, aplicaciones o software web. Esta tecnología se basa en el uso de HTML, CSS y JavaScript estándar, además proporciona un modelo de programación basado en componentes el cual permite desarrollar de manera eficiente las interfaces de usuario, ya sean de forma simples o complejas [65].

Este framework permite al desarrollador elegir entre dos enfoques:

- Sólo tiempo de ejecución.
- Tiempo de ejecución más compilador.

El enfoque de sólo tiempo de ejecución implica la no entrega de un compilador, esto significa que en lugar de la plantilla dentro de la instancia [66], un desarrollador necesita utilizar archivos ".vue" que serán compilados en Javascript, donde estos archivos, junto con código HTML, también contienen un código específico de Vue compilado como parte de un flujo de trabajo, por lo que lo que se entrega a un navegador al final es un código optimizado que es sólo Javascript y no incluye ningún compilador ni código HTML.

Vue.js se centra únicamente en la capa funcional y, por tanto, no es un marco versátil [67]. Este framework puede implementar la vinculación de datos bidireccional responsiva a través de una sencilla API y construir rápidamente interfaces de usuario.

NuxtJS

NuxtJS es un framework frontend construido en JavaScript que permite desarrollar aplicaciones web reactivas de forma intuitiva y de alto rendimiento mediante el uso de componentes, además NuxtJS también es un marco de renderizado del lado del servidor basado en Vue.js, el cual elimina la mayor parte de la configuración compleja involucrada en el procesamiento de datos asíncronos, el middleware y el enrutamiento. Esta tecnología permite a los desarrolladores realizar lo siguiente [68]:

- Estructurar aplicaciones Vue.js mediante el uso de una arquitectura estándar
- Desarrollar páginas estáticas.
- Desarrollar aplicaciones de una sola página (SPA).
- Renderizar estáticamente las aplicaciones Vue.js con beneficio universal.

Backend

La tecnología backend, es la tecnología usada para el desarrollo trasero de una aplicación, es decir de los aspectos que no son visibles para el usuario, la parte donde se realizan consultas y peticiones con la base de datos, es conocido como el lado servidor de un sitio web o aplicación, como la transformación y el almacenamiento de datos de usuario y la capacidad de servir datos de una manera escalable y altamente disponible [64]. Las tecnologías usadas para desarrollar el lado backend de las aplicaciones son las siguientes: Java, PHP, Ruby on Rails, Python, ASP.NET, Golang, SQL, Node.js.

Node.js

Node.js es una plataforma de código abierto que se utilizar en el desarrollo backend que permite ejecutar código JavaScript en el lado del servidor de un sistema, una ventaja de esta tecnología es que permite crear aplicaciones con ejecución de alto rendimiento y de JavaScript [67], además posee un marco de trabajo basado en eventos que evita la activación secundaria de las peticiones web a través de mecanismos impulsados por eventos [69].

Node.js es una tecnología muy utilizada actualmente debida a las siguientes razones [70]:

- Tecnología ligera y de alto rendimiento.
- Facilidad de utilizar y conectar con aplicaciones frontend programadas en JavaScript.
- Tiene buenos tiempos de respuesta y rendimiento en el backend en comparación con otras tecnologías del lado del servidor.
- Permite la escalabilidad en aplicaciones modernas.
- Es compatible para trabajar con la tecnología de microservicios.
- Reduce el tiempo de carga mediante el almacenamiento en caché rápido.
- Permite la creación de aplicaciones multiplataforma.

Express

Express es un framework de desarrollo web para Node.js de código abierto que se utiliza para desarrollar aplicaciones web de una forma fácil, minimalista, sencilla y flexible [71]**.**

Esta tecnología es utilizada para desarrollar APIs RESTful, el cual permite el uso de métodos HTTP y middleware para una correcta construcción de una API.

Para poder programar con Express se requiere del uso de los lenguajes de programación de JavaScript o TypeScript. Algunas características de Express, son las siguientes:

- Desarrollo rápido en el lado servidor (backend).
- Permite configurar middlewares.
- Se puede usar enrutamiento.
- Creación de aplicaciones webs de forma sencilla y eficaz.
- Permite la programación con microservicios.
- Permite usar plantilla HTML (ni no se quiere utilizar una tecnología frontend).
- Permite el despliegue rápido de aplicaciones.
- Programación con JavaScript y TypeScript.
- Útil para aplicaciones pequeñas, medianas y grandes.
- Soporta la arquitectura modelo, vista, controlador (MVC).
- Conexión rápido con cualquier tipo de base de datos.

1.4. Antecedentes Contextuales

Se realizó el análisis de diversos estudios relacionado con las arquitecturas de alta disponibilidad con microservicios y contenedores. En la investigación [12], se puede destacar que los sistemas que cuentan con una arquitectura de alta disponibilidad con contenedores y microservicios facilitan los procesos de mantenibilidad, reutilización, escalabilidad, disponibilidad e implementación automatizada, en donde cada servicio se encuentra en un contenedor, además si un componente del sistema falla, no afecta a todo el sistema.

Las tecnologías TI siempre pasan en constante cambio y se vuelven cada vez más complejas aumentando las dificultades en el desarrollo y el mantenimiento, provocando innovación y presión en la investigación para hacer frente a estos problemas; En la investigación [72], explica cómo efecto de esto apareció conceptos como diseño dirigido por dominio, integración continua, sistemas escalables y software como servicio, que forman la base del concepto de microservicios. Por tal razón, decidieron poner a prueba está nueva tecnología con la mejora de una plataforma de aprendizaje electrónico en educación STEM utilizando la arquitectura de microservicios. Con la implementación de la arquitectura lograron que la plataforma se pueda dividir en varios dominios desarrollados de forma independiente, trabajar con servicios independiente, tener un despliegue más conveniente y haciéndola altamente escalable.

Las arquitecturas con microservicios son uno de los temas más relevantes o destacados en el área de desarrollo de software en los últimos tiempos en cuanto a tecnología, este estilo arquitectónico se centra en crear reducidos componentes de software que focalizan su esfuerzo en una y solo una funcionalidad, siendo independientes, autosuficientes y altamente cohesivos [12].

La mayoría de las aplicaciones están construidas con arquitecturas monolíticas, pero para las aplicaciones modernas no es muy recomendable. Las aplicaciones modernas tienen una gran demanda de usuarios y como buena opción para satisfacer esta demanda es usar las nuevas tecnologías, por tal razón en la investigación [73], desarrollaron una aplicación con microservicios utilizando NodeJS, NextJs, Docker, Kubernetes ingress-nginx y nats streaming server. Además, explican que los microservicios es una tecnología escalable de forma independiente. Si en el sistema existe un aumento en la demanda de una característica en concreto, los recursos de la característica demandada se pueden escalar fácilmente en lugar de escalar toda la aplicación. Además, un sistema con arquitectura de microservicios tendrá una base de código pequeña y, como toda la aplicación se divide en pequeñas funciones o servicios, es fácil de mantener, implementar y mantener limpio el código.

El diseño e implementación de un sistema con una arquitectura de microservicios, como en la investigación [74], le permitió mejorar la infraestructura en cuanto a rendimiento a la plataforma de compras en línea “Ala orden”, donde la implementación de la tecnología, beneficiara a la plataforma al momento de realizar procesos de forma eficaz, cambios en cualquier servicio en caso de fallos o cambios requeridos, ya que cada microservicio se los puede tratar de manera aislada evitando interrupciones en los demás servicios de la aplicación, además de implementar mejoras y añadir funcionalidades futuras de forma ágil.

En el trabajo de titulación [10], realizó la migración hacia una arquitectura basada en microservicios del sistema de gestión centralizadas de laboratorios de la DGIP debido a que las arquitecturas monolíticas son incapaces de crecer, por lo cual las arquitecturas de microservicios permiten la escalabilidad, modificabilidad, alto grado de resiliencia y compresibilidad de los sistemas. Además, el desarrollo de aplicaciones con arquitectura de microservicios se lo realiza en aplicaciones modernas debido a la gran estabilidad que se tiene y la rápida retroalimentación para los usuarios.

Al día de hoy los desarrolladores de software estudian nuevas arquitecturas y técnicas, como es la arquitectura de microservicios y Docker como tecnología de contenedores; Permitiendo mejorar

la escalabilidad y eficiencia en la implementación y funcionamiento en producción de aplicaciones web.

Docker permite crear, probar e implementar aplicaciones de forma liviana, rápida y escalable con la virtualización de contenedores, y por sus características es ideal para aplicaciones implementadas en arquitectura de microservicios ofreciendo una gran oportunidad para reducir el costo de implementación de aplicaciones y mejorar la experiencia de los usuarios finales [75].

1.4.1. Ámbito de aplicación

Se creó una arquitectura basada en microservicios y contenedores con la capacidad de soportar grandes cantidades de solicitudes de forma concurrente y con una aceptable tolerancia al fallo.

La arquitectura fue puesta a prueba mediante la simulación de peticiones del proceso de matriculación del Centro de Educación Continua de la UTMACH, donde se sometió dicha arquitectura a pruebas de rendimiento, para medir su comportamiento frente a la alta disponibilidad y concurrencia del sistema.

Con esta arquitectura se benefician los sistemas que tienen procesos concurrentes y necesiten soportar un gran número de solicitudes simultáneas evitando fallas y caídas del sistema y brindando una mejor experiencia de uso a los usuarios finales.

1.4.2. Establecimiento de requerimientos

La arquitectura propuesta satisface las necesidades de alta concurrencia, tomando como referencia la cantidad de peticiones simultaneas que se definieron en la muestra, todo esto mediante una herramienta de pruebas que permita la emulación de los usuarios, obteniendo métricas de rendimiento relevantes para concluir la viabilidad de implementación del prototipo en un entorno real.

El prototipo se basa exclusivamente en las funcionalidades del sistema vigente del Centro de Educación Continua de la UTMACH, utilizando tecnologías modernas y adecuadas para un buen rendimiento del prototipo web evaluado.

CAPÍTULO II. DESARROLLO DEL PROTOTIPO

2.1. Definición del prototipo

Para definir el prototipo, se parte del análisis del esquema arquitectónico actual que posee el sistema del Centro de Educación Continua de la UTMACH, se puede resumir en la Figura 10:

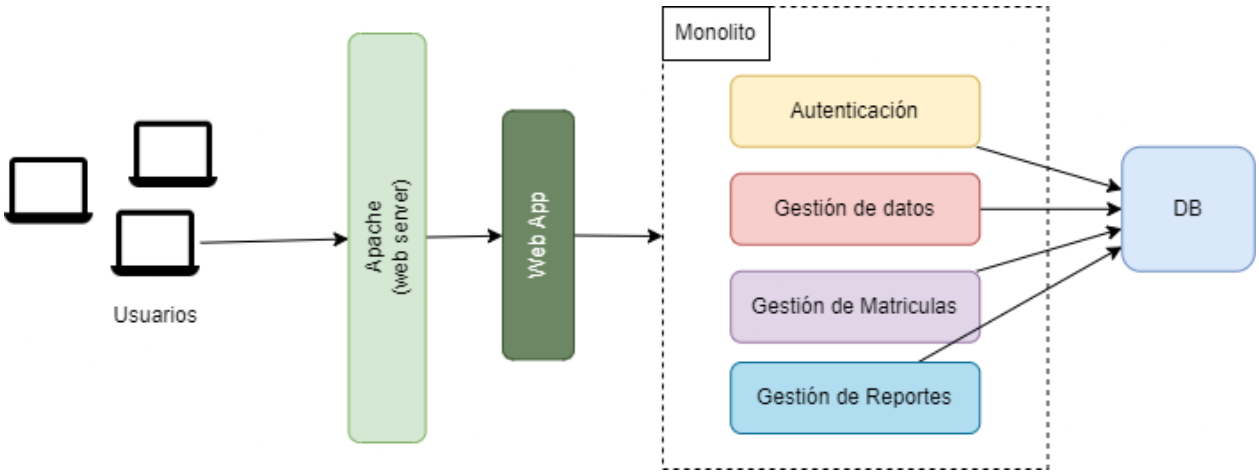


Figura 10. Arquitectura del sistema actual de matriculación.

Esta arquitectura, si bien, es una implementación correcta y funcional, sufre de ciertos problemas de rendimiento que se hacen notorios al momento en el que concurren un cúmulo considerable de estudiantes al mismo tiempo, solicitando tanto el ingreso, como la gestión de matrículas, los cuales se ven congestionados de peticiones, desencadenando un bajo rendimiento en la carga del aplicativo web, y demora en las solicitudes que procesa el servidor.

Allí es donde se plantean crear una serie de implementaciones nuevas, con técnicas claves como el load-balancing y la separación de lógica de negocio basado en pequeños contextos acotados bien definidos (microservicios), los cuales permiten el escalamiento horizontal permitiendo responder a una gran cantidad de solicitudes cuanto se demande. Inicialmente, se plantea en la Figura 11, una propuesta arquitectónica bajo estos principios.

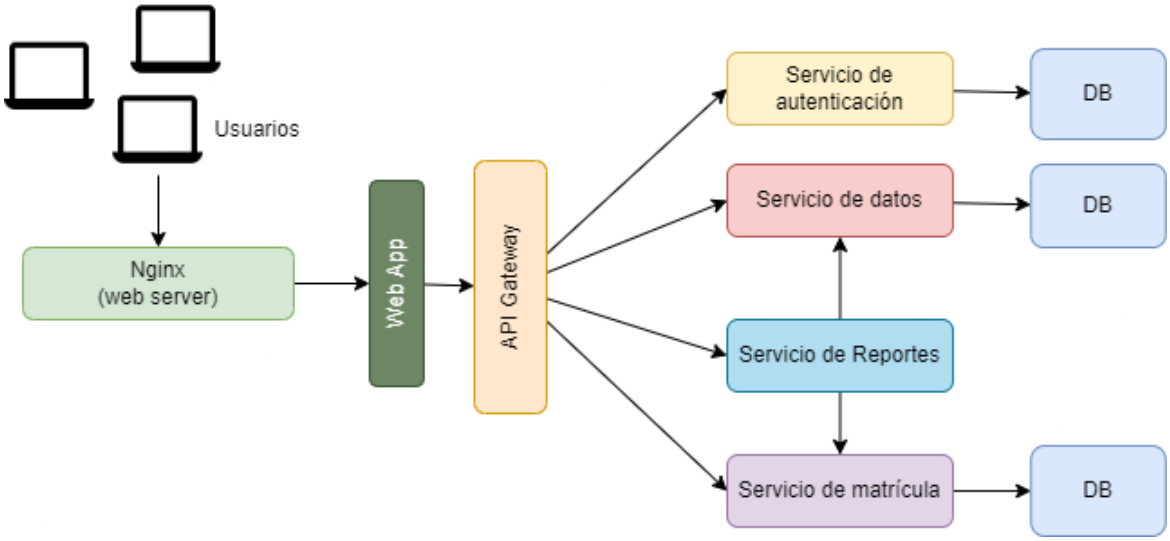


Figura 11. Propuesta arquitectónica para el sistema de matriculación altamente concurrente.

Esta arquitectura, destaca por sobre la anterior, mostrando cómo es posible separar tanto la lógica de negocio, como sus datos, permitiendo técnicas como clustering, o el uso de tecnologías especificas a las necesidades que requieran los datos o servidores para afrontar eficientemente la concurrencia a la que el sistema pueda someterse.

2.2. Metodología de desarrollo del prototipo

2.2.1. Enfoque, alcance y diseño de investigación

Enfoque de la investigación

Se empleó un enfoque cuantitativo, dado que fue pertinente para probar la hipótesis planteada, basándose en las métricas claves obtenidas en el proceso de pruebas realizadas al sistema de matrículas que implementa la arquitectura propuesta.

Alcance de la investigación

Se planteó un alcance exploratorio inicial, para indagar sobre la tendencia actual de microservicios y su aplicabilidad en sistemas con necesidades de alta concurrencia, para luego evolucionar a un alcance correlacional, en donde, se buscó crear un sistema web prototipo basado en las características funcionales que posee el sistema actual de matriculación accesible por parte de los estudiantes del Centro de Educación Continua de la UTMACH, para desarrollar y probar tanto las arquitectura de microservicios propuesta, como una arquitectura monolítica, siendo esta última necesaria para contrastar los resultados de rendimiento obtenidos, para demostrar la hipótesis planteada.

Diseño de la investigación

El diseño de la investigación será cuasiexperimental, cuyo objetivo es comprobar la hipótesis con un grupo definido de peticiones simuladas, donde se pretende establecer el efecto de implementar una arquitectura basada en microservicios para satisfacer las necesidades de alta concurrencia en el sistema de matriculación.

2.2.2. Unidades de análisis

Población (universo)

Considerando el Art. 80 del Reglamento de Régimen Académico, "…Para el tercer nivel de grado se requerirá al menos el nivel B1."[3], provoca una gran afluencia de interés por parte de la comunidad universitaria a obtener los niveles de inglés necesarios. Según el informe emitido en el presente año, se reporta que la población estudiantil de la Universidad Técnica de Machala en el año 2021, alcanzó una cifra de **11917** estudiantes en total [76]. Sin embargo, dentro de la población total contabilizada no todos los estudiantes aplican a cupos de matrícula para los cursos de inglés.

El Centro de Educación Continua de la Universidad Técnica de Machala en el último periodo proyectado para julio - agosto del 2022, oferta una cantidad de **3200** cupos disponibles [77]. En una entrevista realizada (ver Anexo 1 - Entrevista del estado actual del sistema de matriculación Centro de Educación Continua de la UTMACH) al departamento de TIC's de la universidad, no se logró determinar con exactitud el número de estudiantes que realizan el proceso de matriculación al mismo tiempo, pese a eso, se conoce que los cupos se acaban en pocos minutos por la alta demanda de los mismos. Para lo cual, se estima un rango aproximado del 50% de interesados que realizan varias peticiones debido al inherente flujo de información del sistema actual [78].

Muestra

El cálculo de la muestra se realizó con un nivel de confianza del 95% y un margen de error del 1%, obteniendo como resultado 5319 usuarios, el cuál va a ser el número de peticiones simuladas consideradas como umbral de concurrencia mínima aceptable para la arquitectura propuesta.

2.2.3. Técnicas e instrumentos de recopilación de datos

La técnica e instrumento de recopilación de datos para la investigación creación de una arquitectura de alta disponibilidad con microservicios y contenedores se detallan en la Tabla 6:

Tabla 6. Técnicas e instrumentos de recopilación de datos.

Técnicas	Instrumento
Observación	Reporte para el análisis y recopilación de datos, en el cual se obtendrán métricas de la prueba de carga (Load Testing) aplicando a las arquitecturas mediante las peticiones simuladas al momento de realizar transacciones, el reporte será obtenido utilizando la herramienta de testing JMeter (ver Anexo 3).

2.2.4. Técnicas de procesamiento de datos para la obtención de resultados

Las técnicas de procesamiento y análisis de datos será la aplicación de gráficos de barras y gráficos de líneas que permitirán analizar los resultados obtenidos de la herramienta de testing JMeter.

2.2.5. Metodología o métodos específicos

La metodología que se utilizó en la investigación, creación de la arquitectura de alta disponibilidad con microservicios y contenedores es la metodología SEED(S), la cual tiene siete pasos esenciales para el diseño de servicios, los cuáles se visualizan en la Figura 12.

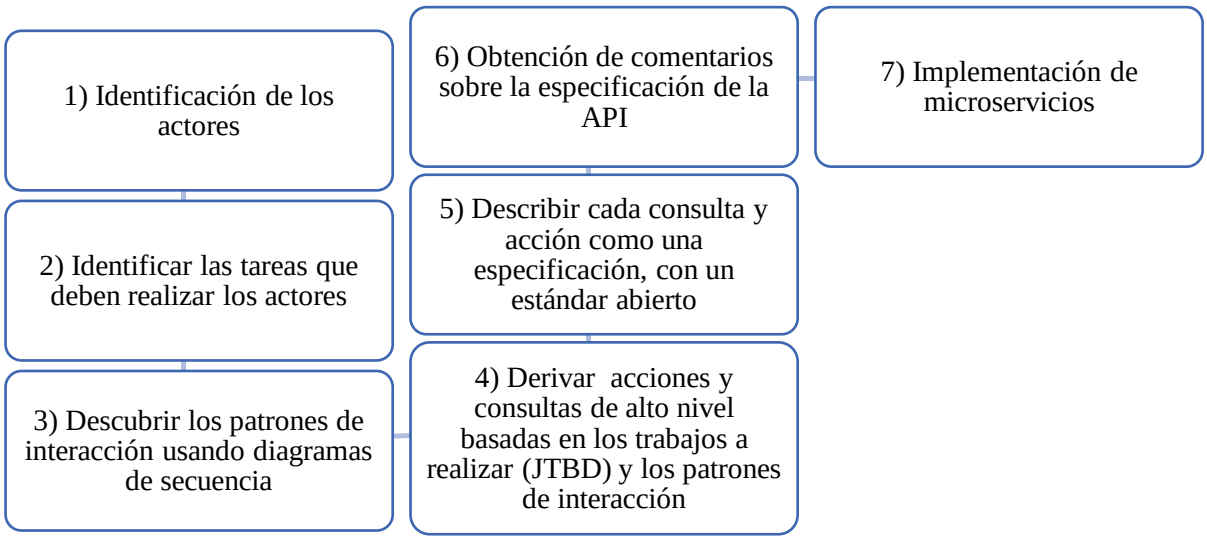


Figura 12. Pasos de la metodología SEED(S).

2.2.6. Herramientas y/o Materiales

Tabla 7. Herramientas y/o materiales.

Categoría	Herramientas y/o materiales
Software	• Node.js • Docker
Hardware	• Computador personal.
Lenguajes	• Typescript/JavaScript • YML

Datos	• Métricas de rendimiento.
	  ○ Rendimiento.
	  ○ Tiempo de carga.
	  ○ Errores por Segundo.
	  ○ Número máximo de usuarios.

Técnicas de pruebas	• Prueba de carga.

2.3. Desarrollo del prototipo

Para poder empezar a desarrollar la arquitectura, fue necesario partir desde la definición del sistema preexistente, para considerar el contexto en el que el sistema se desenvuelve, los requisitos funcionales y no funcionales que cumple, para así poder implementar mejoras a dicha arquitectura, utilizando para ello un enfoque orientado a microservicios. El sistema informático de matriculación actual se compone de 3 apartados diferenciados:

Apartado de login/registro

- Permite el acceso, registro, restablecimiento de contraseña que permita el ingreso al sistema.

Una vez el usuario acceda, se debe permitir cerrar la sesión actual y la navegación entre dos apartados, datos personales y matriculación.

Apartado de datos personales

- Se permite el cambio de contraseña.
- Se permite la consulta o modificación de los datos personales, los cuales se pueden observar en la Tabla 8.

Tabla 8. Datos personales de los usuarios del prototipo del sistema de matriculación.

Datos personales
Tipo Documento
Número del Documento
Nombres
Apellidos
Estado Civil
Sexo
Tipo Discapacidad
% Discapacidad
Fecha de Nacimiento
País
Provincia
Ciudad
Dirección
Teléfono Fijo
Teléfono Móvil
Email

Apartado de matriculación

- Se permite revisar resultados previos de estados de exámenes de ubicación realizados, con los datos que se encuentran en la Tabla 9:

Tabla 9. Datos de matriculación del prototipo del sistema de matriculación.

Datos de matriculación
Periodo
Tipo de curso
Curso
Horario
Nivel de ubicación obtenido

- Se permite el acceso al listado de hojas de matrícula, con los datos que se observan en la Tabla 10:

Tabla 10. Datos de hoja de matrícula del prototipo del sistema de matriculación.

Datos de hoja de matrículas
Periodo
Tipo de curso
Sección
Curso
Horario
Estado

- Se permite realizar el proceso de matriculación, con el ingreso de los datos que se visualizan en la Tabla 11:

Tabla 11. Datos de solicitud de la matrícula del prototipo del sistema de matriculación.

Datos de solicitud de matricula
Programa (Listado de cursos disponibles)
Tipo Periodo
Tipo Curso
Sección
Nivel
Curso
Te enteraste del Curso por
Días
Inicia el
Termina
Horario De
A

Donde:

- Programa: Listado de cursos disponibles ofertados registrados.
- Tipo Periodo: listado de periodos disponibles (dependiente al programa).
- Tipo Curso: listado de cursos disponibles (dependiente del tipo periodo).
- Sección: listado de secciones disponibles (dependiente del).
- Nivel: listado de niveles disponibles (dependiente de la sección)
- Curso: (dependiente del nivel).

- Te enteraste del Curso por: Listado de plataformas registradas.
- Días, Inicia el, Termina, Horario De, A: Auto rellenado por el sistema, datos del curso seleccionado.

Además, antes de enviar la solicitud de matrícula, se debe validar el cruce de horarios del curso seleccionado, contra otras matrículas actualmente vigentes.

2.3.1. Metodología SEED(S)

Paso 1. Identificación de los actores

Los actores principales que componen la arquitectura de microservicios son los estudiantes, pero estos actores se clasifican en dos grupos:

- Estudiantes de la UTMACH.
- Estudiantes particulares.

Paso 2. Identificar las tareas que deben realizar los actores

Para identificar las tareas que deben realizar cada actor, se emplearon el uso de historias de trabajo (JTBD).

Historia de trabajo de los estudiantes de la UTMACH y estudiantes particulares.

Servicio de autenticación

Iniciar sesión

Cuando un estudiante de la UTMACH o un estudiante particular ingresa a la página principal del sistema de la matriculación, **quiere** iniciar sesión, para **poder** acceder a todos los servicios que ofrece el sistema. De esta historia de trabajo se obtiene la tarea de "iniciar sesión".

Recuperar contraseña

Cuando un estudiante de la UTMACH o un estudiante particular se olvide la contraseña, **quiere** recuperarla, para **poder** acceder de nuevo al sistema. De esta historia de trabajo se obtiene la tarea de "recuperar contraseña".

Actualizar contraseña

Cuando un estudiante de la UTMACH o un estudiante particular necesite modificar la contraseña, **quiere** actualizarla, para **poder** tener una contraseña diferente a la anterior. De esta historia de trabajo se obtiene la tarea de "actualizar contraseña".

Registrarse

Cuando un estudiante particular ingresa a la página principal del sistema de la matriculación y este no tenga una cuenta para acceder al sistema, **quiere** registrarse en el sistema, para **poder** acceder a todos los servicios que ofrece el sistema. De esta historia de trabajo se obtiene la tarea de "registrarse".

Servicio de datos

Registrar datos personales

Cuando un estudiante particular al momento de registrarse en el sistema de matriculación, **quiere** registrar sus datos personales, para **poder** visualizar después de haber accedido al sistema. De esta historia de trabajo se obtiene la tarea de "registrar datos personales".

Visualizar datos personales

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar sus datos personales, para **poder** ver su respectiva información. De esta historia de trabajo se obtiene la tarea de "visualizar datos personales".

Editar datos personales

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación y este visualizando sus datos personales, **quiere** modificar o cambiar algún dato, para **poder** actualizar alguna información que tenga errónea. De esta historia de trabajo se obtiene la tarea de "editar datos personales".

Visualizar países

Cuando un estudiante de la UTMACH o un estudiante particular al momento de estar registrándose en el sistema de matriculación este debe seleccionar su país de residencia, entonces

quiere visualizar países, para **poder** registrar dicho país. De esta historia de trabajo se obtiene la tarea de “visualizar países”.

Visualizar provincias de un país

Cuando un estudiante de la UTMACH o un estudiante particular al momento de estar registrándose en el sistema de matriculación este debe seleccionar la provincia de residencia, entonces **quiere** visualizar las provincias del país seleccionada, para **poder** registrar dicho dato. De esta historia de trabajo se obtiene la tarea de “visualizar provincias de un país”.

Visualizar ciudades de una provincia

Cuando un estudiante de la UTMACH o un estudiante particular al momento de estar registrándose en el sistema de matriculación este debe seleccionar la provincia ciudad de residencia, entonces **quiere** visualizar las ciudades de la provincia seleccionada, para **poder** registrar dicho dato. De esta historia de trabajo se obtiene la tarea de “visualizar las ciudades de una provincia”.

Servicio de matriculación

Visualizar programas ofrecidos

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar los programas que ofrece la institución, para **poder** seleccionar el programa correspondiente. De esta historia de trabajo se obtiene la tarea de “visualizar programas”.

Visualizar los tipos de periodos

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar los tipos de periodo, para **poder** seleccionar el periodo de matriculación. De esta historia de trabajo se obtiene la tarea de “visualizar los tipos de periodos”.

Visualizar los tipos de cursos

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar los tipos de cursos, para **poder** seleccionar el tipo de curso respectivo. De esta historia de trabajo se obtiene la tarea de “visualizar los tipos de cursos”.

Visualizar las secciones

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar las secciones, para **poder** seleccionar respectiva sección. De esta historia de trabajo se obtiene la tarea de "visualizar las secciones".

Visualizar los niveles

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar los niveles, para **poder** seleccionar el respectivo nivel. De esta historia de trabajo se obtiene la tarea de "visualizar los niveles".

Visualizar medios de difusión (te enteraste del curso por)

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar los medios de difusión, para **poder** seleccionar el respectivo nivel de difusión. De esta historia de trabajo se obtiene la tarea de "visualizar medios de difusión (te enteraste del curso por)".

Visualizar los cursos

Cuando un estudiante de la UTMACH o un estudiante particular ya haya seleccionado un programa, **quiere** visualizar los cursos de un respectivo programa, para **poder** matricularse. De esta historia de trabajo se obtiene la tarea de "visualizar cursos".

Matricularse

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** matricularse en un curso, para **poder** matricularse en un curso de un respectivo programa seleccionado. De esta historia de trabajo se obtiene la tarea de "matricularse".

Visualizar los datos de la matrícula

Cuando un estudiante de la UTMACH o un estudiante particular ya se hay matriculado a un curso, **quiere** visualizar los datos matricula, para **poder** visualizar dicha información. De esta historia de trabajo se obtiene la tarea de "visualizar los datos de la matrícula".

Servicio de reportes

Resultados de examen de ubicación

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar el resultado que obtuvo en el examen de ubicación, para **poder** conocer su nota y saber a qué curso fue nivelado. De esta historia de trabajo se obtiene la tarea de "resultado de examen de ubicación".

Visualizar lista de matrículas

Cuando un estudiante de la UTMACH o un estudiante particular este usando el sistema de matriculación, **quiere** visualizar los cursos en los que se ha matriculado, para **poder** visualizar una lista de los cursos en los que se ha inscrito. De esta historia de trabajo se obtiene la tarea de "visualizar lista de matrículas".

Paso 3. Descubrir los patrones de interacción usando diagramas de secuencia.

En este paso se realizará los diagramas de secuencia para cada actividad de los servicios (autenticación, datos, matriculación y reportes) analizados en el paso anterior. La herramienta que se utilizó para la elaboración de los diagramas es PlantUML. Los diagramas de secuencia pueden ser observados en el Apéndice 1.

Servicio de autenticación

Iniciar sesión

El patrón de interacción de iniciar sesión se define con los actores "estudiante UTMACH" y "estudiante particular", los cuáles visitan la "página de login" del sistema de matriculación, allí los actores deben rellenar un formulario y luego deben presionar el botón "ingresar", luego se enviará los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de autenticación" para utilizar el método de "login(data)".

Recuperar contraseña

El patrón de interacción de recuperar contraseña se define con los actores "estudiante UTMACH" y "estudiante particular", los cuáles visitan la "página de login" del sistema de matriculación, deben dar clic en "Olvidé mi contraseña", luego los redireccionan a la "página de recuperar

contraseña", llenan el formulario y luego deben presionar el botón "recuperar contraseña", entonces así se enviarán los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de autenticación" para utilizar el método de "recoveryPassword(data)".

Actualizar contraseña

El patrón de interacción de actualizar contraseña se define con los actores "estudiante UTMACH" y "estudiante particular". Estos deben dirigirse a la "página de principal del sistema", posteriormente le dan clic en "cambiar contraseña" y se encontraran en la "página de cambiar contraseña", allí el actor debe rellenar un formulario y luego debe presionar el botón de "actualizar contraseña", el cual enviara todos los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de autenticación" para utilizar el método de "updatePassword(userID, contrasena)".

Registrarse

El patrón de interacción de registrarse se define con el actor "estudiante particular" visitan la "página de login" del sistema de matriculación, luego dan clic en el botón "registrarse", se redireccionan a la "página de registrarse", allí el actor debe rellenar sus datos en el formulario y luego debe presionar el botón de "enviar solicitud de registro", el cual enviara todos los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de autenticación" para utilizar el método de "createUser(user)".

Servicio de datos

Registrar datos personales

El patrón de interacción de registrar datos personales se define con el actor "estudiante particular", el cual debe estar ubicado en la "página de login" del sistema de matriculación, luego dar clic en el botón "registrarse", se redirecciona a la "página de registrarse", allí el actor debe rellenar sus datos en el formulario y luego debe presionar el botón de "enviar solicitud de registro", el cual enviara todos los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de datos" para utilizar el método de "register(data)".

Visualizar datos personales

El patrón de interacción de visualizar datos personales se define con los actores "estudiante UTMACH" y "estudiante particular", estos se deben ubicarse en la "página principal" del sistema

de matriculación, luego deben dar clic en “actualizar datos”, los redireccionaran en la “página de datos personales”. Una vez que se encuentran en dicha página, se realizará una petición a la API mediante el uso de peticiones HTTP. La API invocará el “servicio de datos” para utilizar el método de “getData(userID)”, posteriormente se visualizarán los datos en el formulario que se encuentra en la página.

Editar datos personales

El patrón de interacción de editar datos personales se define con los actores “estudiante UTMACH” y “estudiante particular”, deben encontrarse en la “página principal” del sistema de matriculación, luego deben dar clic en “actualizar datos”, los redireccionaran en la “página de datos personales”. Una vez que se encuentran en dicha página, visualizarán un formulario con los datos personales, ese formulario modificar la información. Una vez modificada la información deben presionar el botón de “actualizar datos personales”, para que se pueda enviar los datos a la API mediante el uso de peticiones HTTP. La API invocará el “servicio de datos” para utilizar el método de “updateData(userID, data)”.

Visualizar países

El patrón de interacción de visualizar países se define con los actores “estudiante UTMACH” y “estudiante particular”, deben encontrarse en la “página de registrarse” del sistema de matriculación, en el formulario deben seleccionar el “país”, al seleccionar se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de datos” para utilizar el método de “getCountries()”.

Visualizar provincias de un país

El patrón de interacción de visualizar países se define con los actores “estudiante UTMACH” y “estudiante particular”, deben encontrarse en la “página de registrarse” del sistema de matriculación, en el formulario deben seleccionar la “provincia”, al seleccionar se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de datos” para utilizar el método de “getStates(countryID)”.

Visualizar ciudades de una provincia

El patrón de interacción de visualizar países se define con los actores “estudiante UTMACH” y “estudiante particular”, deben encontrarse en la “página de registrarse” del sistema de

matriculación, en el formulario deben seleccionar la “ciudad”, al seleccionar se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de datos” para utilizar el método de “getCities(stateID)”.

Servicio de matriculación

Visualizar programas ofrecidos

El patrón de interacción de visualizar programas ofrecidos se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar el “programa”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getPrograms()”.

Visualizar los tipos de periodos

El patrón de interacción de visualizar los tipos de periodos se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar el “tipo de periodo”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getPeriodsByProgram(programID)”.

Visualizar los tipos de cursos

El patrón de interacción de visualizar los tipos de cursos se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar el “tipo de curso”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getCoursesTypesByPeriod(periodID)”.

Visualizar las secciones

El patrón de interacción de visualizar las secciones se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar

el “sección”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getSectionsByCourseType(courseTypeID)”.

Visualizar los niveles

El patrón de interacción de visualizar los niveles se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar el “nivel”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getLevelsBySection(sectionID)”.

Visualizar medios de difusión (te enteraste del curso por)

El patrón de interacción de visualizar medios de difusión (te enteraste del curso por) se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar el “te enteraste el curso por”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getBroadcastTypes()”.

Visualizar los cursos

El patrón de interacción de visualizar los cursos se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, en dicha página se encuentra con un formulario, allí deben seleccionar el “curso”, al seleccionarlo se llamarán los datos de la API mediante el uso de peticiones HTTP. La API invocará el “servicio de matriculación” para utilizar el método de “getCoursesByLevel(levelID)”.

Matricularse

El patrón de interacción de matricularse se define con los actores “estudiante UTMACH” y “estudiante particular”, estos deben encontrarse en la “página de matriculación” del sistema de matriculación, allí deben rellenar un formulario seleccionado los datos solicitados y posteriormente presionar el botón de “matricularme al curso” para que se pueda enviar los datos a

la API mediante el uso de peticiones HTTP. La API invocará el "servicio de matriculación" para utilizar el método de "enroll(curso, usuario)".

Visualizar los datos de la matrícula

El patrón de interacción de matricularse se define con los actores "estudiante UTMACH" y "estudiante particular", estos deben encontrarse en la "página de hojas de matrículas" del sistema de matriculación, en esa página se llamarán los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de matriculación" para utilizar el método de "getRegularEnrollmentsByUser(userID)", luego se visualizará una lista de matrículas, después de se debe seleccionar la matrícula para visualizar sus respectivos datos, se volverá a llamar a la API invocando el "servicio de matriculación" para utilizar el método de "getRegularEnrollmentsById(enrollmentID)".

Servicio de reportes

Resultados de examen de ubicación

El patrón de interacción de resultados de examen de ubicación se define con los actores "estudiante UTMACH" y "estudiante particular", estos deben encontrarse en la "página de examen de ubicación" del sistema de matriculación. Una vez que se encuentran en dicha página, se realizará una petición a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de reportes" para utilizar el método de "getLocationExams(userID)". Una vez realizada la petición, en esa sección cargaran los datos y visualizara una lista de todos los resultados de los exámenes de ubicación que haya dado el estudiante.

Visualizar lista de matrículas

El patrón de interacción de lista de matrículas se define con los actores "estudiante UTMACH" y "estudiante particular", estos deben encontrarse en la "página de hojas de matrículas" del sistema de matriculación, en esa página se llamarán los datos a la API mediante el uso de peticiones HTTP. La API invocará el "servicio de reportes" para utilizar el método de "getEnrollments(userID)". Una vez realizada la petición, en esa sección cargaran los datos y visualizara una lista de todos los resultados de los exámenes de ubicación que haya dado el estudiante.

Paso 4. Derivar acciones y consultas de alto nivel basadas en los trabajos a realizar (JTBD) y los patrones de interacción.

Servicio de autenticación

Iniciar sesión

- **Entradas:** usuario, contraseña.
- **Salida:** Mensaje de éxito "Inicio de sesión correcto" o un mensaje de error "Cédula y contraseña incorrectos".

Recuperar contraseña

- **Entradas:** usuario, email.
- **Salida:** Mensaje de "recuperación de contraseña" o un mensaje de error "Ocurrió un error".

Actualizar contraseña

- **Entradas:** id_usuario, contrasena.
- **Salida:** Mensaje de "recuperación de contraseña" o un mensaje de error "Ocurrió un error".

Registrarse

- **Entradas:** id_usuario, usuario, nombres, contrasena.
- **Salida:** Mensaje de éxito "Creación correcta de usuario" o un mensaje de error "Ocurrió un error".

Servicio de datos

Registrar datos personales

- **Entradas:** tipo_documento, numero_documento, nombres, apellidos, estado_civil, sexo, tipo_discapacidad, porcentaje_discapacidad, fecha_nacimiento, pais, provincia, ciudad, direccion, telefono_fijo, telefono_movil, email, contrasena.
- **Salida:** Mensaje de éxito "Registrado con éxito" o un mensaje de error "No se enviaron los datos a registrar / No se pudo registrar al Estudiante".

Visualizar datos personales

- **Entradas:** userID.
- **Salida:** Datos personales del usuario.

Editar datos personales

- **Entradas:** tipo_documento, numero_documento, nombres, apellidos, estado_civil, sexo, tipo_discapacidad, porcentaje_discapacidad, fecha_nacimiento, pais, provincia, ciudad, direccion, telefono_fijo, telefono_movil, userID.
- **Salida:** Mensaje de éxito "Datos actualizados con éxito" o un mensaje de error "Ocurrió un error".

Visualizar países

- **Entradas:** N/A.
- **Salida:** Listado de los países.

Visualizar provincias de un país

- **Entradas:** countryID.
- **Salida:** Listado de las provincias de un país.

Visualizar ciudades de una provincia

- **Entradas:** stateID.
- **Salida:** Listado de las ciudades de una provincia.

Servicio de matriculación

Visualizar programas ofrecidos

- **Entradas:** N/A.
- **Salida:** Lista de programas ofrecidos.

Visualizar los tipos de periodos

- **Entradas:** programID.
- **Salida:** Listado de los tipos de periodos del programa seleccionado.

Visualizar los tipos de cursos

- **Entradas:** periodID.
- **Salida:** Listado de los tipos de cursos del periodo seleccionado.

Visualizar las secciones

- **Entradas:** courseTypeID.
- **Salida:** Listado de las secciones del tipo de curso seleccionado.

Visualizar los niveles

- **Entradas:** sectionID.
- **Salida:** Listado de los niveles de la sección seleccionada.

Visualizar medios de difusión (te enteraste del curso por)

- **Entradas:** N/A.
- **Salida:** Listado de los medios de difusión.

Visualizar los cursos

- **Entradas:** levelID.
- **Salida:** Lista de los cursos del nivel seleccionado.

Matricularse

- **Entradas:** curso, usuario.
- **Salida:** Mensaje de éxito “Matriculado con éxito” o un mensaje de error “Ocurrió un error”.

Visualizar los datos de la matrícula

- **Entradas:** enrollmentID.
- **Salida:** Listado de los datos de la matrícula.

Servicio de reportes

Resultados de examen de ubicación

- **Entradas:** userID.
- **Salida:** Lista de resultados de examen de ubicación de un estudiante en específico.

Visualizar lista de matrículas

- **Entradas:** userID.
- **Salida:** Lista de matrículas de un estudiante en específico.

Paso 5. Describir cada consulta y acción como una especificación con un estándar abierto.

Para el diseño de las plantillas de la API de la arquitectura de microservicios se utilizó la extensión OpenAPI (Swagger) Editor en la herramienta de Visual Studio Code.

Se procedió a realizar el diseño de la API por cada servicio que se identificaron en los puntos anteriores. Los servicios tienen sus respectivos métodos (actividades/tareas) los cuales son los métodos HTTP que se utilizaran para el funcionamiento de la API. A continuación, se procede a realizar la descripción del diseño de la API. Los diseños de la API con la especiación de estándar abierto pueden ser observados en el Apéndice 2.

Servicio de autenticación

El servicio de autenticación consta de dos métodos HTTP, los cuales son:

- Método HTTP POST /login (Método de iniciar sesión).
- Método HTTP POST /recovery-password (Método de recuperar contraseña).
- Método HTTP POST /update-password (Método de actualizar contraseña).

Además, este servicio tiene un método que se usará en la comunicación entre servicios, el cual es:

- Método HTTP POST /create-user (Método de crear un usuario dentro del servicio de autenticación).

Servicio de datos

El servicio de datos consta de tres métodos HTTP, los cuales son:

- Método HTTP POST /register (Método de visualizar datos personales del estudiante).
- Método HTTP GET /data/{userID} (Método de editar datos personales del estudiante).
- Método HTTP PUT /data/{userID} (Método de actualizar/modificar la contraseña del estudiante).
- Método HTTP GET /countries
- Método HTTP GET /state/{countryID}
- Método HTTP GET /cities/{stateID}

Servicio de matriculación

El servicio de matriculación consta de tres métodos HTTP, los cuales son:

- Método HTTP GET /programs (Método de obtención de programas).
- Método HTTP GET /periods (Método de obtención de periodos).
- Método HTTP GET /course-types (Método de obtención de tipos de cursos).
- Método HTTP GET /sections (Método de obtención de secciones).
- Método HTTP GET /levels (Método de obtención de niveles).
- Método HTTP GET /courses (Método de obtención de cursos).
- Método HTTP GET /courses/{courseID} (Método de obtención de un curso en específico).
- Método HTTP GET /broadcast-types (Método de obtención de medios de difusión).
- Método HTTP POST /enrollment (Método para matricularse en el curso especificado).

Además, este servicio tiene un método que se usará en la comunicación entre servicios, el cual es:

- Método HTTP GET /enrollment/{enrollmentID} (Método que permite obtener datos de la matrícula).

Servicio de reportes

El servicio de reportes consta de dos métodos HTTP, los cuales son:

- Método HTTP GET /location-exam (Método de visualizar la lista de resultados de exámenes de ubicación).

- Método HTTP GET /enrollment (Método de visualizar lista de matrículas).

Paso 6. Obtención de comentarios sobre la especificación de la API.

En este paso se realizó reuniones de revisión con el grupo de trabajo de titulación, el tutor y cotutor. El objetivo de las reuniones es la revisión de los pasos anteriores, primeramente, identificando los actores y sus respectivas actividades en las que se fue añadiendo más, luego se revisó el flujo de los diagramas de secuencia, las entradas y salida de los métodos. Por último, se dieron opiniones en la mejora de la especificación del diseño de la API.

Paso 7. Implementación de microservicios.

Una vez obtenido el contexto necesario, los casos de uso bien identificados, se empezó a trazar una primera aproximación sobre los microservicios necesarios para satisfacer los requerimientos del negocio, agrupando de forma cohesiva dentro de un contexto bien definido cada servicio.

Como resultado del proceso metodológico, la arquitectura que se obtuvo se puede resumir en el esquema de la Figura 13, donde se observa la toma de importantes decisiones con base en el contexto del negocio, puesto que se proponen réplicas para los servicios de autenticación y matrícula, al ser los más concurridos en periodos de matriculación, haciendo énfasis en el escalamiento horizontal que el enfoque arquitectónico provee, además de poder separar las bases de datos, aplicando tecnologías acorde a las necesidades de concurrencia de cada servicio, apuntando hacia un mejor rendimiento de los servicios y el sistema en general.

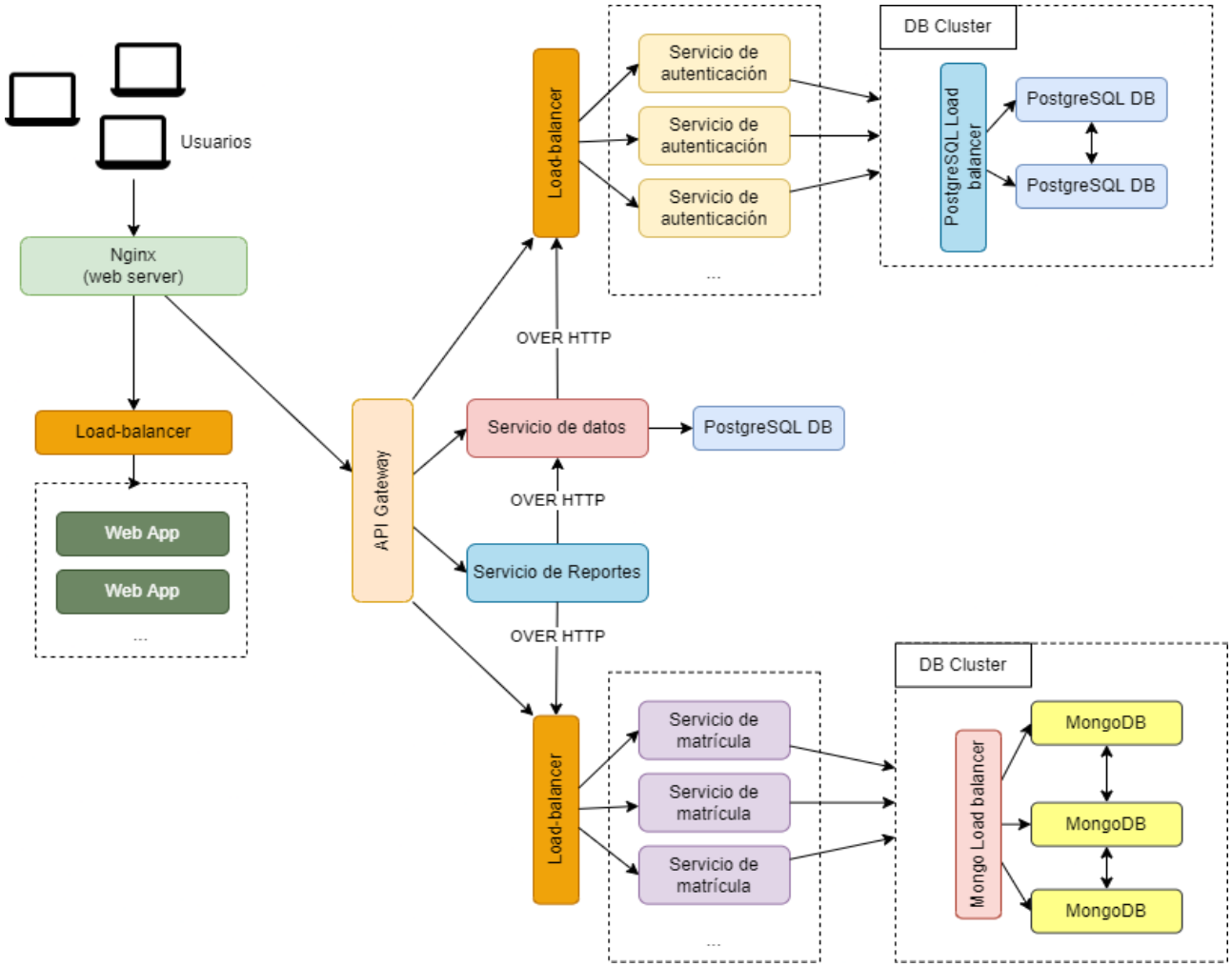


Figura 13. Arquitectura de microservicios del sistema de matriculación.

2.4. Ejecución del prototipo

Con un mapa claro de lo que se necesita, fue posible empezar con la implementación de toda la infraestructura necesaria para que la comunicación del sistema de microservicios y el desarrollo de código para que el prototipo de sistema de matriculación funcione según el ámbito establecido. La implementación de la arquitectura se puede dividir en tres secciones:

- ***Desarrollo de la infraestructura de comunicación:*** Implica la implementación de tecnologías y técnicas para la comunicación de todo el sistema de microservicios, usando Docker, NGINX y bases de datos como MongoDB y PostgreSQL.
- ***Desarrollo de la interfaz de usuario:*** Corresponde a la creación grafica del prototipo, usando la tecnología NuxtJS.
- ***Desarrollo del código de servicios:*** Involucra el desarrollo de los procesos modulares principales del negocio, desarrollados bajo la tecnología de servidor Express.

2.4.1. Desarrollo de la infraestructura de comunicación

La infraestructura de comunicación de microservicios se puede considerar como todas las técnicas, herramientas y tecnologías, aplicadas con el fin de comunicar el prototipo cliente, con el sistema de microservicios construido en el Backend.

Para la organización del proyecto en general, se partió de una separación modular por carpetas, iniciando con la capa de comunicación centralizada (API Gateway) que sirve también como balanceador de carga de los microservicios que lo requieran, módulos de servicio (auth-module, data-module, enrollment-module, report-module) agrupando el respectivo servicio y módulo de datos en caso de ser requeridos, y por último en el módulo de presentación (Frontend) que contiene el prototipo del aplicativo web.

En la raíz de todo el proyecto, se encuentra el archivo principal de configuración de cada uno de los servicios y tecnologías que funcionarán en torno a variables de ambiente, preconfiguradas en cada módulo o servicio que lo requieran. Además, los códigos fuente de la configuración y desarrollo de la infraestructura de comunicación pueden ser observadas en el Apéndice 3.

Aplicativo web.

Configuración de la imagen

El Dockerfile asociado a la construcción de la imagen para la ejecución del prototipo web consta de una fase de liberación, de un contenedor basado en Node 16, corriendo sobre la distribución de alpine, la instalación de dependencias del aplicativo, y se ejecuta el comando que da inicio a la aplicación, exponiendo la misma según el puerto configurado por variable de entorno.

Configuración del servicio

La configuración del servicio del aplicativo consta de:

- Build: ubicación de la carpeta del Dockerfile de configuración de la imagen.
- Puerto de salida (80:5001): Expone el puerto para el protocolo HTTP (80), desde la ejecución del aplicativo SSR interno en el puerto 5001.
- Variable de entorno (PORT): Puerto configurable donde se expondrá el aplicativo de NuxtJS dentro del contenedor.

Servicios de autenticación, datos, matrícula y reportes

Dado que, los 4 servicios fueron desarrollados bajo el mismo stack de tecnologías (servidor express con Typescript), la imagen y el servicio comparten las mismas características, siendo las siguientes:

Configuración de la imagen

El Dockerfile asociado a las imágenes de los servicios se componen de dos fases, ambas basadas en Node 16, corriendo sobre la distribución de alpine. La primera denominada fase de 'constructor', ejecuta una instalación completa de dependencias, con el fin de elaborar el build con las utilidades de desarrollo y generar como resultado la carpeta "dist". En la segunda fase, llamada 'liberación', se obtiene la carpeta dist con el código transpilado, se realiza una instalación de los paquetes de producción mínimos necesarios para correr el aplicativo, para luego ejecutar con Node el archivo de entrada, exponiendo el servidor bajo el puerto configurado por la variable de entorno PORT.

Configuración del servicio

Para la configuración de los servicios se utiliza:

- Build: ubicación de la carpeta del Dockerfile de configuración de la imagen.
- Variable de entorno (PORT): Puerto configurable donde se expondrá el aplicativo de Express dentro del contenedor.
- Variable de entorno (NODE_NUMBER): número identificador de nodo replica.
- Variable de entorno (SERVICE_NAME): nombre del servicio.

Servicios de base de datos para autenticación, datos y matrículas

Para los servicios de base de datos, se utilizaron las siguientes configuraciones.

Configuración de la imagen

El Dockerfile para las bases de datos de los módulos de autenticación y datos, parte de la imagen oficial de PostgreSQL, corriendo bajo una distribución ligera de alpine, mientras que el Dockerfile para la base de datos del módulo de matrícula, se basa en la imagen oficial de MongoDB, corriendo bajo una distribución de focal.

Ambas configuraciones de imágenes, constan de una única fase, en donde se establecen las variables de entorno que utiliza la imagen para configurar las credenciales de acceso a base de datos, y además se especifica la ruta de un archivo SQL, que se copia en el directorio denominado '/docker-entrypoint-initdb.d/' que es utilizado por la imagen para inyectar scripts de código que se ejecutarán en la fase de creación del contenedor, como parte de su configuración.

Los scripts SQL de configuración para PostgreSQL, son aprovechados en ambos casos para establecer tanto el esquema como las tablas asociadas a los datos que se guardarán para cada servicio de datos en particular.

Mientras que, para la configuración del contenedor de MongoDB se utiliza la Shell de mongo bajo el lenguaje JavaScript, para crear un usuario y las colecciones respectivas para guardar los documentos de la base de datos.

Configuración del servicio

Indistintamente del gestor de base de datos, para la configuración de los servicios de base de datos se utiliza:

- Ports: puertos expuestos para conectarse localmente a las bases de datos (habilitados únicamente para desarrollo y pruebas).
- Build: ubicación de la carpeta del Dockerfile de configuración de la imagen.
- Archivo de entorno (env_file): ubicación del archivo de variables de entorno, con las credenciales de acceso a la base de datos.

Api Gateway

Configuración de la imagen

El Dockerfile asociado a la imagen del API Gateway, surge de una imagen base oficial de NGINX, corriendo sobre una ligera distribución de alpine. Este servidor ligero sirve de punto de entrada único para las peticiones que realizara el cliente, y a la vez, actuará como load balancer, para los servicios que lo requieran, además de la copia de archivos y carpetas de configuración externos, necesarios para la ejecución del servidor.

La configuración del servidor NGINX, se detalla la inclusión de los clústeres de servidores al que se envían las solicitudes actuando como proxy, el puerto de ejecución del servidor y las configuraciones de redirecciones incluidas en la carpeta 'api/'.

Los clústeres mencionados anteriormente, se configuran en el archivo 'api_backends.conf', donde se encuentran categorizados por servicios y conectados bajo el nombramiento de cada servicio y el puerto configurado respectivamente, Docker facilita esta configuración ya que maneja un servicio de DNS interno, que resuelve la dirección host de cada 'server', evitando colocar direcciones IP estáticas.

Por último, las configuraciones del API necesarias para las redirecciones se registran las interfaces (API) que se obtuvieron en el Paso 5, del desarrollo metodológico, para permitir el acceso y balanceo de carga de usuarios aplicando el algoritmo Round Robin que provee el servicio de NGINX.

Configuración del servicio

La configuración del servicio para el API Gateway indicado consta de:

- Build: ubicación de la carpeta del Dockerfile de configuración de la imagen.
- Puerto de salida (8000:5000): Expone el puerto para el protocolo HTTP (8000), desde la ejecución de NGINX interno en el puerto 5000.
- Servicios dependientes (depends_on): Servicios relacionados al API Gateway que son necesarios para el correcto funcionamiento del mismo.

2.4.2. Desarrollo de prototipo web

Aprovechando las bondades de los microservicios, fue posible trabajar paralelamente en el desarrollo del prototipo web para pruebas, diseñando las interfaces correspondientes al sistema de matriculación accesible por parte de los estudiantes del Centro de Educación Continua de la UTMACH. Para el desarrollo del prototipo web, se empleó la tecnología NuxtJS en su versión 2, que permite el renderizado de lado del servidor (SSR), el cual es una buena alternativa de cara al rendimiento de la aplicación.

Las interfaces de usuarios del prototipo web pueden ser visualizados en el Apéndice 4.

Interfaz de inicio de sesión

El prototipo, incluye la interfaz o pantalla de inicio de sesión, el cual permitirá al estudiante loguearse al sistema. Esta interfaz consta de un campo de usuario (cedula o número de documento) y contraseña.

Interfaz de registro del estudiante

La pantalla de registro, es un replica de los datos que requiere la ventana de registro del sistema actual de matriculación, con opción a volver al inicio.

Interfaz de matriculación del estudiante

La interfaz de matriculación, emula y registra todos los datos correspondientes a la selección del curso, a los que el estudiante puede postular en ese momento.

Interfaz de consulta de resultados de exámenes de ubicación

La interfaz correspondiente a los resultados de los exámenes de ubicación, contiene una tabla básica, que muestra la misma información que la utilizada en el sistema vigente.

Interfaz de consulta de hojas de matrícula

La página de consulta de hojas de matrícula, incluye la misma funcionalidad de impresión con la que cuenta el sistema de matriculación actual.

Interfaz de consulta de edición de datos personales

La pantalla de edición de datos personales, incluye los mismos datos del formulario que se reflejan en el sistema vigente.

Interfaz de consulta de cambio de contraseña

La interfaz de cambio de contraseña muestra un formulario sencillo donde se comprueba la validez de la contraseña, tal como el sistema vigente lo realiza.

2.4.3. Desarrollo del código de los servicios

Para el desarrollo del código de servicios se realizó la programación en el backend de la arquitectura de alta disponibilidad utilizando Express.js con Node.js y el lenguaje de programación TypeScript. Los códigos fuentes de cada método pueden ser observados en el Apéndice 5.

Servicio de autenticación

Método de iniciar sesión

El método de iniciar sesión recibe los datos usuario y contrasena del req.body, estos datos son usados para realizar las consultas a la base de datos para saber si ese usuario se encuentra registrado y así poder realizar el logueo al sistema.

Método de recuperar contraseña

El método de recuperar contraseña recibe los datos usuario y email del req.body, en el cual realiza se genera una nueva contraseña al usuario con texto aleatorio y esa contraseña es actualizada en la base de datos y es enviada al correo electrónico del usuario.

Método actualizar contraseña

El método de actualizar contraseña recibe el id_usuario, contrasena_actual y contrasena_nueva del req.body en el cual se compara si contraseña actual y la contraseña nueva son iguales, en caso de que son iguales no se podrá actualizar la contraseña, caso contrario si la contraseña nueva es diferente está será actualizada en la base de datos reemplazando la contraseña anterior.

Método de registrarse

El método de registrarse recibe id_usuario, usuario, nombres y contrasena del req.body que son utilizados para realizar el registro o ingreso de los datos del usuario en la base de datos.

Servicio de datos

Método de registrar datos personales

El método de registrar datos personales recibe tipo_documento, numero_documento, nombres, apellidos, estado_civil, sexo, tipo_discapacidad, porcentaje_discapacidad, fecha_nacimiento, pais, provincia, ciudad, direccion, telefono_fijo, telefono_movil, email y contrasena del req.body que son utilizados para realizar el registro o ingreso de los datos personales del usuario en la base de datos.

Método de visualizar datos personales

El método de visualizar datos personales recible el userID del req.body que es utilizado para realizar una consulta en la base de datos y así obtener los respectivos datos personales del usuario especificado.

Método de editar datos personales

El método de editar datos personales recible el userID del req.body que será usado para realizar la actualización e de la información del respectivo usuario especificado en la base de datos.

Método de visualizar países

El método de visualizar países realiza una consulta en la base de datos y así obtener una lista de todos los países que se encuentran registrados.

Método de visualizar provincias de un país

El método de provincias de un país recibe el countryID del req.params, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todas las provincias del país especificado.

Método de visualizar ciudades de una provincia

El método de visualizar ciudades de una provincia obtiene el stateID del req.params, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todas las ciudades de la provincia especificada.

Servicio de matriculación

Método de visualizar programas ofrecidos

El método de visualizar programas ofrecidos realiza una consulta en la base de datos y así obtener una lista de todos los programas que se encuentran registrados.

Método de visualizar los tipos de periodos

El método de visualizar los tipos de periodos obtiene el programID del req.query, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todos los periodos de un programa en específico.

Método de visualizar los tipos de cursos

El método de visualizar los tipos de cursos obtiene el periodID del req.query, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todos los tipos de cursos de un periodo en específico.

Método de visualizar las secciones

El método de visualizar las secciones obtiene el courseTypeID del req.query, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todos las secciones de un tipo de curso en específico.

Método de visualizar los niveles

El método de visualizar los niveles obtiene el sectionID del req.query, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todos los niveles de una sección en específica.

Método de visualizar medios de difusión (te enteraste del curso por)

El método de visualizar los medios de difusión realiza la consulta en la base de datos y así obtener una lista de todos los medios de difusión que se encuentran registrados.

Método de visualizar los cursos

El método de visualizar los cursos obtiene el levelID del req.query, dato que es usado para realizar la consulta en la base de datos y así obtener una lista de todos los cursos de un nivel en específico.

Método de matricularse

El método de matricularse obtiene el curso, usuario del req.body, datos que son usados para realizar la realizar la matriculación del usuario en un respectivo curso del programa seleccionado, esta información es registrada en la base de datos al usuario y el curso en el que se matriculó.

Método de visualizar los datos de la matrícula

En el método de visualizar se encuentran dos métodos, donde:

- El primero recibe el userID del req.params, dato que será utilizado para obtener una lista de las matrículas regulares del usuario.
- El segundo recibe el enrollmentID del req.params, dato que será utilizado para obtener la información o datos de la matricula regular seleccionada.

Servicio de reportes

Método de resultados de examen de ubicación

El método de resultados de examen de ubicación obtiene el userID del req.query, dato que será usado para realizar la consulta en la base de datos y así obtener una lista de todos los resultados de exámenes de ubicación del usuario especificado.

Método de visualizar lista de matrículas

El método de visualizar lista de matrículas obtiene el userID del req.query, dato que será usado para realizar la consulta en la base de datos y así obtener la lista matrículas del usuario especificado.

CAPÍTULO III. EVALUACIÓN DEL PROTOTIPO

3.1. Plan de evaluación

3.1.1. Objetivo

Evaluar el prototipo del trabajo de integración curricular mediante la aplicación de pruebas de carga con peticiones simuladas para la obtención y análisis de resultados entre la arquitectura de microservicios propuesta y la arquitectura monolítica.

3.1.2. Cronograma

El cronograma de activadas se planificó en 4 semanas (del 23 de enero del 2023 al 17 de febrero del 2023) para la realización de la evaluación del prototipo. El cronograma se puede observar en la Tabla 12:

Tabla 12. Cronograma de actividades para el capítulo III.

Actividades	Semanas			
	Semana 11	**Semana 12**	**Semana 13**	**Semana 14**
Desarrollo del plan de evaluación.				
Elaboración del prototipo Backend con arquitectura monolítica.				
Elaboración del Testing plan con la herramienta JMeter.				
Realización de pruebas y obtención de métricas en ambos prototipos				
Análisis de los resultados y elaboración de conclusiones				

3.1.3. Proceso

Elaboración del prototipo Backend con arquitectura monolítica

Se creó un sistema monolítico utilizando como punto de partida el análisis y diseño del prototipo basado en microservicios, para desarrollar uno nuevo que ejecute las mismas funcionalidades, y someter a ambos sistemas a las mismas pruebas de rendimiento.

Elaboración del Testing plan con la herramienta JMeter

La herramienta seleccionada JMeter, se utilizó para realizar la configuración de los planes de prueba de ambos sistemas, considerando que deben ser las mismas en ambos ambientes, para poder contrastar objetivamente los resultados en la etapa final.

Realización de pruebas y obtención de métricas en ambos prototipos.

Para la realización de pruebas y con la finalidad de eliminar lo más posible el consumo de recursos, se configuraron dos equipos físicos separados conectados mediante una red de área local doméstica, para los cuales se realizaron configuraciones de hardware, que se pueden observar en la Tabla 13 y Tabla 14.

Laptop DELL (utilizada como servidor):

Tabla 13. Características del hardware de la maquina utilizada como servidor.

Característica	Valor
RAM (cantidad módulos/espacio total de memoria)	2 módulos / 32 GB
CPU (Cores / marca y modelo)	6 núcleos / Intel Core I7 7ma generación
Disco duro (almacenamiento/marca y modelo)	500 GB / Western Digital Blue / SSD Nvme
Sistema operativo	Windows server 2022

Computador de escritorio (utilizada como maquina cliente):

Tabla 14. Características del hardware de la maquina utilizada como cliente.

Característica	Valor
RAM (cantidad módulos/espacio total de memoria)	2 módulos / 16 GB
CPU (Cores / marca y modelo)	6 núcleos / Intel Core I5 12va generación
Disco duro (almacenamiento/ marca / tecnología)	500 GB / Kingston / SSD Nvme
Sistema	Windows 11 Pro

La configuración de contenedores, es exclusiva del computador utilizado como servidor, en la que se instaló Docker 4.16.3 con la build 96739. Particularmente la arquitectura de microservicios, fue escalada horizontalmente con 8 servidores por servicio (matriculación, autenticación, datos y reportes).

Se consideró la prueba de carga (Load testing), como el instrumento de prueba que ayudará a constatar la correcta respuesta a las solicitudes de ciertas cantidades específicas de usuarios concurrentes. Se planificaron 4 pruebas de carga que fueron aplicadas a ambos prototipos, para comprobar el rendimiento de los sistemas ante grandes cantidades de concurrencia de usuarios, cuyas configuraciones se encuentran descritas en la Tabla 15.

Tabla 15. Configuración para las pruebas de carga (Load testing).

# Prueba	Carga objetivo (hilos concurrentes)	Paso de transición	Tiempo de paso de transición (segundos)	Tiempo manteniendo la carga objetivo (segundos)
1	1 000	10	50	180
2	3 000	10	50	180
3	5 000	10	50	180
4	8 000	10	50	180

Con la utilización de estos planes de prueba configurados en la herramienta, se esperó obtener y almacenar tanto los resultados brutos de las peticiones realizadas (véase Anexo 2), como el informe condensado por la herramienta JMeter (véase Anexo 3).

Análisis de los resultados y elaboración de conclusiones

Para el análisis de los resultados obtenidos se emplearon gráficos de barras y gráficos de líneas, que serán agrupados por tipo de arquitectura para evidenciar de forma clara la comparación de rendimiento entre ambos sistemas y concluir basado en estas.

3.1.4. Actividades

Las actividades realizadas en el plan de evaluación fueron las siguientes:

- Elaboración de un Backend con arquitectura monolítica.
- Elaboración de los testing plan para la arquitectura monolítica y de microservicios.
- Realizar pruebas de carga a la arquitectura de microservicios.
- Realizar pruebas de carga a la arquitectura monolítica.
- Condensar los resultados de las pruebas en estadísticos descriptivos.
- Analizar los resultados estadísticos obtenidos de la arquitectura de microservicios contra la arquitectura monolítica.

3.1.5. Resultados esperados

Se esperó obtener resultados coherentes respecto de la hipótesis planteada, apuntando hacia una clara tendencia de mejora en el rendimiento del sistema basado en microservicios frente la simulación de peticiones concurrentes, en cuanto a las métricas de: tiempo medio de respuesta, error porcentual, rendimiento y cantidad de solicitudes atendidas.

3.2. Resultados de la evaluación

Siguiendo el plan de evaluación estimado, se logró obtener los resultados mostrados en el Apéndice 6 y Apéndice 7, consiguiendo condensarlos en los siguientes estadísticos descriptivos:

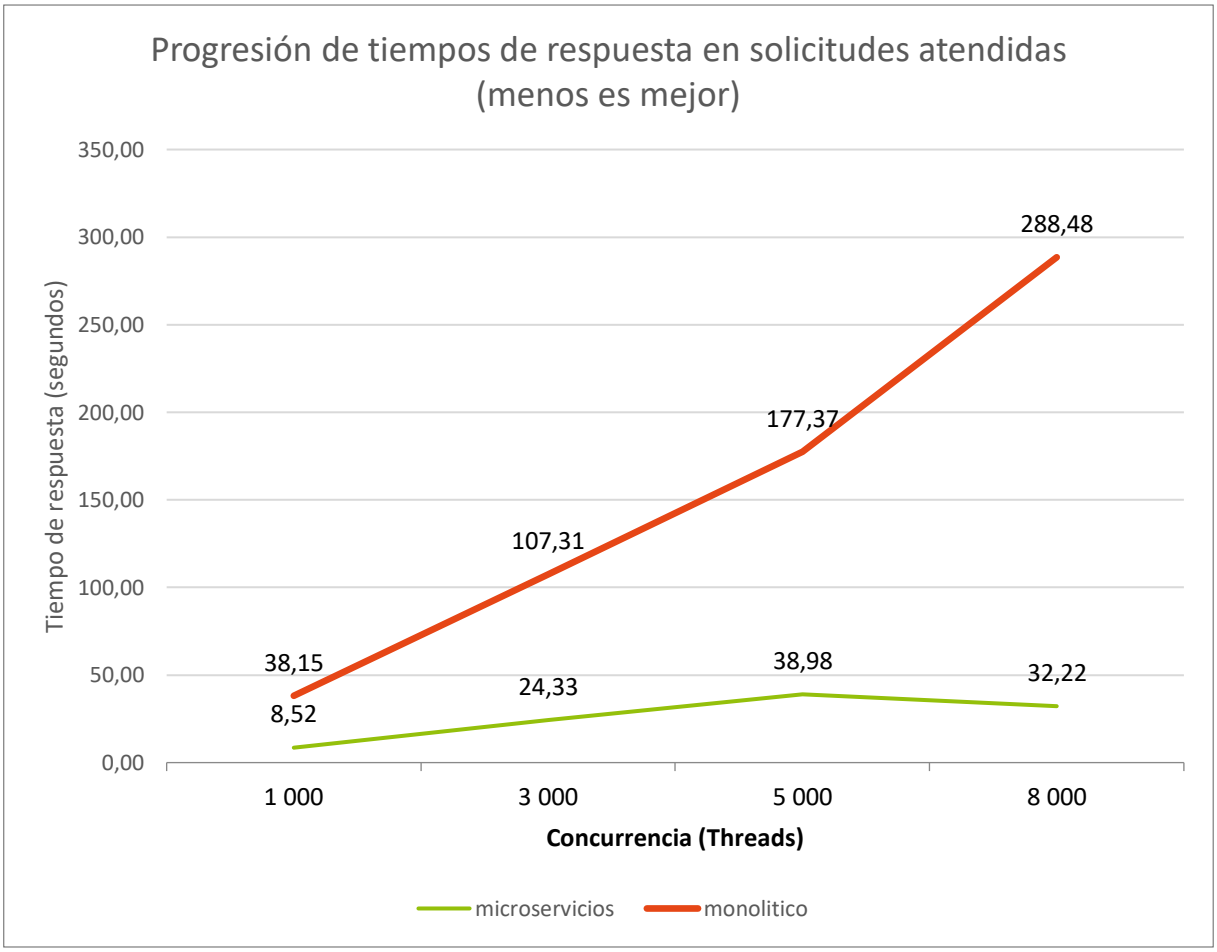


Figura 14. Gráfico comparativo de tiempos medios de respuesta por tipo de arquitectura.

Como es evidente en la Figura 14, la tendencia del incremento de tiempos de respuesta en la arquitectura monolítica es notable, mientras que la arquitectura microservicios presenta significativamente tiempos menores de respuesta, siendo la que mejor desempeño alcanzó en todas las pruebas realizadas.

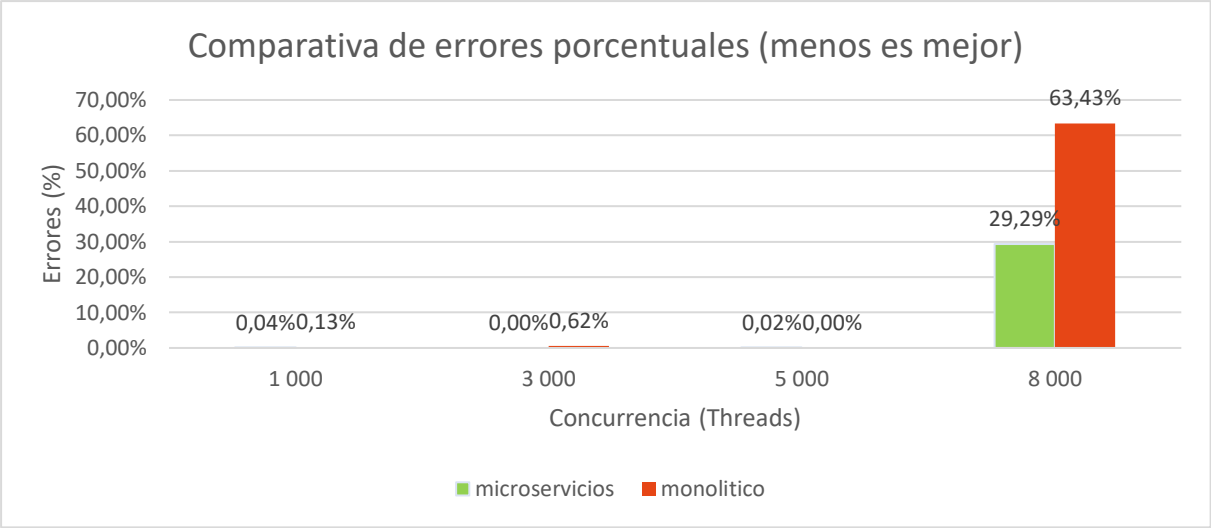


Figura 15. Gráfico comparativo de errores porcentuales.

En la Figura 15, se indican los errores porcentuales obtenidos y es pertinente enfatizar que ambas arquitecturas responden de manera aceptable hasta las 5000 peticiones concurrentes, con un error menor al 1%, sin embargo, en la última prueba realizada con una concurrencia de 8000 hilos, ambas arquitecturas fallan considerablemente, siendo la monolítica la que consiguió un aproximado de 34% más de error porcentual, frente a la arquitectura de microservicios.

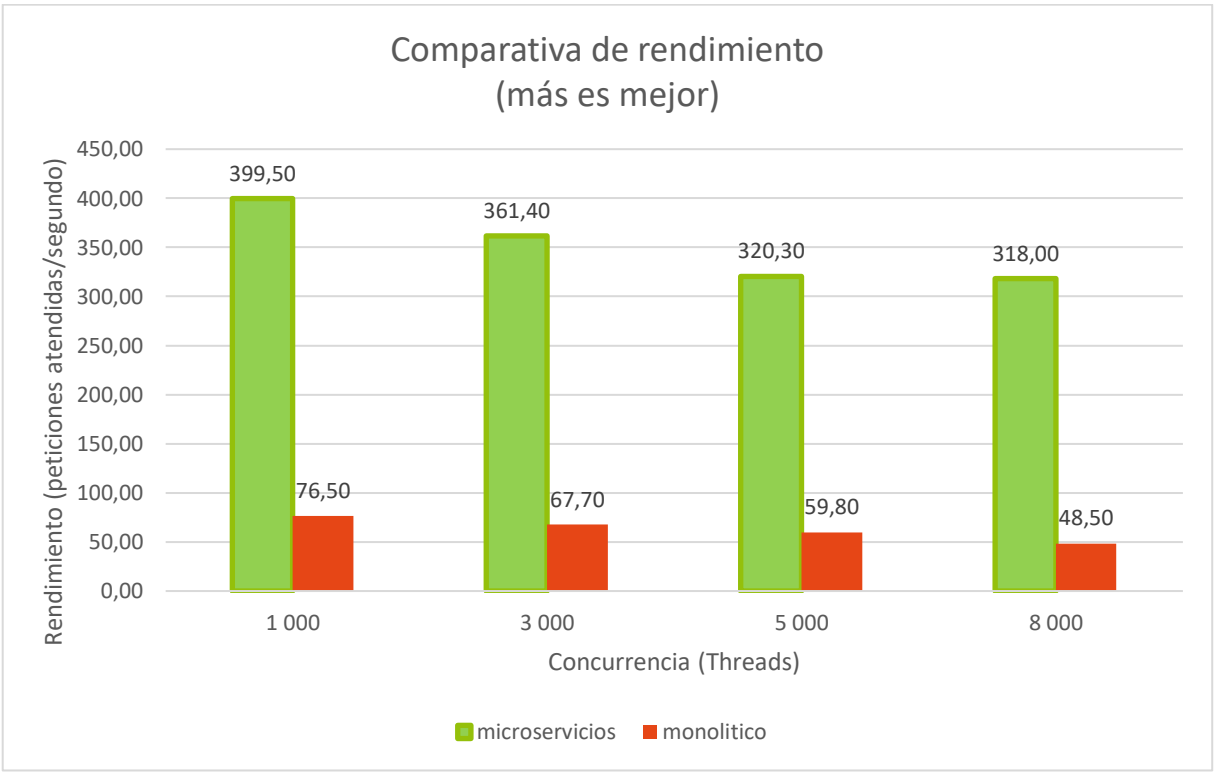


Figura 16. Gráfico comparativo del rendimiento.

En la métrica de rendimiento de solicitudes atendidas por segundo, se puede observar en la Figura 16, se puede observar el esperado decaimiento de rendimiento por parte de ambas arquitecturas y

una clara postura positiva en cuanto al rendimiento obtenido por parte de la arquitectura de microservicios, siendo la mejor puntuada con un rendimiento cinco veces mayor aproximadamente.

3.2.1. Información adicional sobre los resultados

Dentro del proceso estipulado de pruebas descrito previamente, inicialmente se consideraron planes de prueba de hasta 15000 hilos concurrentes, sin embargo, en el transcurso de la ejecución de las pruebas, ambos sistemas empezaron a demostrar comportamientos erráticos a partir de los 10000 hilos, llegando a superar casi el 50% de errores en la arquitectura de microservicios y el 80% en la arquitectura monolítica, siendo bastante imprecisos la obtención de estos resultados, debido a la caída y/o congelamiento de la herramienta JMeter durante estas pruebas, debido a las limitaciones de hardware de los equipos empleados. Por tal motivo, se decidió establecer el punto máximo de solicitudes en 8000 hilos concurrentes, en los que ambas arquitecturas pudieron completar los test de forma normal.

Asimismo, es idóneo mencionar que según el cálculo muestral mencionado en el punto 2.2.2 Unidades de análisis, se cumple el criterio de aceptación de concurrencia con nivel de confianza del 95% y un margen de error del 1%, superando las 5319 peticiones simultaneas indicadas.

Por otra parte, ambas arquitecturas se comportan aceptablemente bien y con un comportamiento similar en cuanto a caída de rendimiento, solicitudes atendidas y tiempos medios de respuesta, siendo todas estas métricas evidenciadas con un decaimiento constante y estable a lo largo de todas las pruebas con excepción de la métrica de error porcentual, que se dispara en la arquitectura monolítica.

CONCLUSIONES

- Es acertado concluir que la creación de la arquitectura de alta disponibilidad siguiendo las pautas, ejemplos, guías e indicaciones oficiales de los proveedores del software necesario para este tipo de implementaciones representó una complejidad no mayor a la esperada, gracias a la aplicación de conceptos clave acerca de software como la modularidad, separación de responsabilidades y las habilidades en cuanto al manejo, configuración de sistemas operativos, ingeniería de software, comprensión y lógica de programación obtenidas a lo largo de la formación académica. Además, el diseño e implementación de la arquitectura propuesta, junto con pruebas apropiadas con base en las métricas pertinentes, demostrando con datos, que el sistema mencionado cumple con los requerimientos de disponibilidad y concurrencia estipulados.
- Se realizó la búsqueda de información en fuentes bibliográficas de alto impacto enfocadas en las temáticas relacionadas al estilo arquitectónico de microservicios y contenedores con Docker u otros temas que fueron necesarios para la elaboración del marco teórico, encontrando un amplio cúmulo de investigaciones, ejemplos, aplicaciones, libros y tesis académicas, que demuestran el interés por parte de la comunidad científica, académica y empresarial, por apostar a estas tecnologías.
- La investigación literaria sobre metodologías de desarrollo de microservicios, determinó que SEED(S) es la única con un enfoque para la correcta separación de servicios lógicos, altamente cohesivos y desacoplados, utilizando Docker, como tecnología de contenedores que gracias a su versatilidad y facilidad de uso bajo la herramienta de administración Docker Desktop y sus archivos de configuración de imágenes y servicios (Dockerfile y Docker Compose respectivamente), permitieron un desarrollo ágil, proporcionando un entorno de fácil uso para el desarrollo de los prototipos y la ejecución de pruebas.
- Se desarrolló un prototipo de aplicación web replicando el sistema de matrículas actual del Centro de Educación Continua de la UTMACH basado en microservicios y contenedores, el mismo que sigue rigurosamente todas las funcionalidades del aplicativo original, para el cual se empleó el framework NuxtJS en su versión 2, que permitió crear una réplica del sistema enfocado hacia el buen rendimiento, al poseer el renderizado de lado del servidor (SSR) que mejora los tiempos de respuesta de las aplicaciones construidas bajo esta tecnología.
- Finalmente, se realizó la evaluación de rendimiento de la arquitectura propuesta y de un prototipo monolítico contrastando ambas arquitecturas frente a un alto número de

peticiones concurrentes, utilizando la herramienta JMeter, que ofrece una amplia gama de configuraciones como variables, preprocesado de solicitudes, de control de errores y plugins como el Concurrency Thread Group, que fue necesario para configurar los niveles de concurrencia descritos en el plan de pruebas, además de contar con la posibilidad de exportar los resultados de los test y proporcionar un cuadro resumen adecuado para extraer las métricas de rendimiento, solicitudes atendidas, porcentaje de error obtenido y media de tiempos de respuesta, que sirvieron de insumo para demostrar la hipótesis planteada, ya que se observa un rendimiento cinco veces mejor aproximadamente de la arquitectura basada en microservicios respecto de la arquitectura monolítica.

RECOMENDACIONES

- Para la implementación de una arquitectura de microservicios, se recomienda contar con personal suficientemente preparado que entienda los conceptos, implicaciones, virtudes y desventajas que conlleva este patrón arquitectónico, así como también contar con equipos de especificaciones técnicas idóneas para levantar entornos controlados de pruebas, acordes al tamaño de concurrencia a emular, para obtener resultados coherentes y propicios para elaborar conclusiones y tomar decisiones acertadas.
- Para la recopilación y búsqueda de la información bibliográfica se recomienda utilizar las diferentes bases de datos de revistas de alto impacto, libros y guías oficiales, además también se recomienda utilizar una metodología para la revisión bibliográfica como la SLR, la cual fue utilizada en la realización del presente trabajo.
- Para la creación de la arquitectura de alta disponibilidad es recomendable seguir una metodología para la separación de la lógica de negocio y elaborar servicios altamente cohesivos y bajamente acoplados, bajo contratos definidos en una especificación reconocida como OpenAPI. También es pertinente seguir las guías oficiales en la documentación de Docker para la correcta configuración de contenedores y levantamiento de servicios. Del mismo modo, se aconseja buscar y probar configuraciones eficientes para el servidor NGINX, que en la arquitectura propuesta cumplió las funciones de Api Gateway y load balancer, cuyo desempeño predefinido está orientado a un uso menos exigente, pero admite configuraciones para soportar altos niveles de concurrencia.
- Al ser una comparación de un servicio ya existente, se recomienda que el prototipo monolítico a medirse contra la arquitectura propuesta, sea el desplegado en un entorno de pruebas adecuado, para obtener métricas de rendimiento más cercanas a la realidad, o en caso de no contar con una copia del código fuente de dicho software, realizar una aproximación que se apegue lo más posible al funcionamiento real de la aplicación, sin agregar funcionalidades o modificar flujos de datos en medida de lo posible.
- Para la elaboración de las pruebas se recomienda revisar tutoriales y configuraciones de pruebas para realizar test pertinentes y adecuados para probar el rendimiento del sistema web, además de apoyarse en los plugins y herramientas brindadas por la comunidad de JMeter, que sin duda facilitan tanto la configuración, recolección, como el análisis de los resultados para las pruebas ejecutadas.
- Finalmente, cabe mencionar que las bases teóricas y prácticas contempladas en este proyecto de integración curricular, pueden ser utilizadas para elaborar implementaciones

de microservicios que incluyen, pero no se limitan a: sistemas de matriculación general de los Institutos de Educación Superior, sistemas de información gubernamentales de la Senecyt para las postulaciones de los aspirantes a la educación pública superior, sistema de postulación de aspirantes a la Policía Nacional del Ecuador, sistemas de reservas de tickets o compra de boletos para eventos multitudinarios, entre otros en donde se precise un sistema con buen rendimiento frente a grandes niveles de concurrencia.

REFERENCIAS BIBLIOGRÁFICAS

[1] E. Marel, "Digital-based Services Globalization and Multilateral Trade Cooperation," vol. 3, pp. 392–398, 2021, doi: 10.1111/1758-5899.12941.

[2] R. Echeverría, "Infraestructura de Internet en América Latina: puntos de intercambio de tráfico, redes de distribución de contenido, cables submarinos y centros de datos," 2020. [Online]. Available: https://repositorio.cepal.org/bitstream/handle/11362/46388/1/S2000651_es.pdf

[3] Consejo de Eduación Superior, Reglamento de régimen académico. 2019, pp. 21–22. [Online]. Available: https://www.ces.gob.ec/lotaip/Anexos%20Generales/a3_Reformas/r.r.academico.pdf

[4] F. Tapia, M. Á. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From Monolithic Systems to Microservices: A Comparative Study of Performance," Appl. Sci., vol. 10, no. 17, Art. no. 17, Jan. 2020, doi: 10.3390/app10175797.

[5] F. Tapia, M. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study," Appl. Sci. Switz., vol. 10, no. 17, 2020, doi: 10.3390/app10175797.

[6] G. Blinowski, A. Ojdowska, and A. Przybylek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," IEEE Access, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[7] O. J. B. Iturralde, Introducción a la arquitectura de software: Un enfoque práctico. Independently published, 2020.

[8] I. Ortiz-Garcés and A. Echeverría, "Arquitectura híbrida de Gestión de Microservicios en Infraestructura Locales," RISTI - Rev. Iber. Sist. E Tecnol. Inf., vol. 45, pp. 626–638, Oct. 2021.

[9] D. R. Martínez, P. V. Aranda, and V. T. Bosch, Microservicios: un enfoque integrado, 1st edition. Madrid: RA-MA Editorial, 2018.

[10] C. A. Chulca Quilachamín and R. P. Molina López, "Migración hacia una arquitectura basada en microservicios del sistema de gestión centralizada de laboratorios de la DGIP," Quito,

2020., 2020. Accessed: Jul. 08, 2022. [Online]. Available: http://bibdigital.epn.edu.ec/handle/15000/20834

[11] N. Dragoni et al., “Microservices: Yesterday, Today, and Tomorrow,” in Present and Ulterior Software Engineering, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.

[12] O. Al-Debagy and P. Martinek, “A Comparative Review of Microservices and Monolithic Architectures,” in 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), Nov. 2018, pp. 149–154. doi: 10.1109/CINTI.2018.8928192.

[13] W. Mengist, T. Soromessa, and G. Legese, “Method for conducting systematic literature review and meta-analysis for environmental science research,” MethodsX, vol. 7, p. 100777, 2020, doi: 10.1016/j.mex.2019.100777.

[14] L. Baresi and M. Garriga, “Microservices: The Evolution and Extinction of Web Services?,” in Microservices: Science and Engineering, A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds. Cham: Springer International Publishing, 2020, pp. 3–28. doi: 10.1007/978-3-030-31646-4_1.

[15] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li, “Microservices: architecture, container, and challenges,” in 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Dec. 2020, pp. 629–635. doi: 10.1109/QRS-C51114.2020.00107.

[16] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, “Microservices: The Journey So Far and Challenges Ahead,” IEEE Softw., vol. 35, no. 3, pp. 24–35, May 2018, doi: 10.1109/MS.2018.2141039.

[17] V. Bushong et al., “On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study,” Appl. Sci., vol. 11, no. 17, Art. no. 17, 2021, doi: 10.3390/app11177856.

[18] A. Hannousse and S. Yahiouche, “Securing microservices and microservice architectures: A systematic mapping study,” Comput. Sci. Rev., vol. 41, p. 100415, Aug. 2021, doi: 10.1016/j.cosrev.2021.100415.

[19] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, “Does migrating a monolithic system to microservices decrease the technical debt?,” J. Syst. Softw., vol. 169, p. 110710, Nov. 2020, doi: 10.1016/j.jss.2020.110710.

[20] F. Chen, L. Zhang, and X. Lian, “A systematic gray literature review: The technologies and concerns of microservice application programming interfaces,” Softw.-Pract. Exp., vol. 51, no. 7, pp. 1483–1508, Jul. 2021, doi: 10.1002/spe.2967.

[21] J. A. Valdivia, A. Lora-Gonzalez, X. Limon, K. Cortes-Verdin, and J. O. Ocharan-Hernandez, “Patterns Related to Microservice Architecture: a Multivocal Literature Review,” Program. Comput. Softw., vol. 46, no. 8, pp. 594–608, Dec. 2020, doi: 10.1134/S0361768820080253.

[22] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, Microservice Architecture: Aligning Principles, Practices, and Culture, 1st edition. Sebastopol, CA: O´Reilly Media, Inc., 2016.

[23] C. T. Joseph and K. Chandrasekaran, “Straddling the crevasse: A review of microservice software architecture foundations and recent advancements,” Softw.-Pract. Exp., vol. 49, no. 10, pp. 1448–1484, Oct. 2019, doi: 10.1002/spe.2729.

[24] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd edition. Beijing Boston Farnham Sebastopol Tokyo: O’Reilly Media, Inc., 2021.

[25] R. Mitra and I. Nadareishvili, Microservices: Up and Running: A Step-by-Step Guide to Building a Microservices Architecture, 1st edition. O’Reilly Media, Inc., 2020.

[26] F. H. Vera-Rivera, C. Gaona, and H. Astudillo, “Defining and measuring microservice granularity-a literature overview,” PeerJ Comput. Sci., vol. 7, p. e695, 2021, doi: 10.7717/peerj-cs.695.

[27] A. Bellemare, Building Event-Driven Microservices: Leveraging Organizational Data at Scale, 1st edition. Beijing Boston Farnham: O’Reilly Media, Inc., 2020.

[28] Mohammad Sadegh Hamzehloui, Shamsul Sahibuddin, Ardavan Ashabi are with University Technology Malaysia, Malaysia, M. S. Hamzehloui, S. Sahibuddin, and A. Ashabi, “A Study

on the Most Prominent Areas of Research in Microservices," Int. J. Mach. Learn. Comput., vol. 9, no. 2, pp. 242–247, Apr. 2019, doi: 10.18178/ijmlc.2019.9.2.793.

[29] S. Newman, Monolith to Microservices. O'Reilly Media, Inc, 2020.

[30] V. Ibarra-Junquera, A. Gonzalez-Potes, C. M. Paredes, D. Martinez-Castro, and R. A. Nunez-Vizcaino, "Component-Based Microservices for Flexible and Scalable Automation of Industrial Bioprocesses," IEEE Access, vol. 9, pp. 58192–58207, 2021, doi: 10.1109/ACCESS.2021.3072040.

[31] A. A. Khaleq and I. Ra, "Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications," IEEE Access, vol. 9, pp. 35464–35476, 2021, doi: 10.1109/ACCESS.2021.3061890.

[32] H. Yu, X. Wang, C. Xing, and B. Xu, "A Microservice Resilience Deployment Mechanism Based on Diversity," Secur. Commun. Netw., vol. 2022, p. e7146716, Jun. 2022, doi: 10.1155/2022/7146716.

[33] P. Ghiya, TypeScript Microservices. Packt Publishing Ltd., 2018.

[34] J. A. Rasheedh and S. Saradha, "Design and Development of Resilient Microservices Architecture for Cloud Based Applications using Hybrid Design Patterns," Indian J. Comput. Sci. Eng., vol. 13, no. 2, pp. 365–378, 2022, doi: 10.21817/indjcse/2022/v13i2/221302067.

[35] D. Jha et al., "A study on the evaluation of HPC microservices in containerized environment," Concurr. Comput. Pract. Exp., vol. 33, no. 7, p. e5323, 2021, doi: 10.1002/cpe.5323.

[36] M. Bogo, J. Soldani, D. Neri, and A. Brogi, "Component-aware orchestration of cloud-based enterprise applications, from TOSCA to Docker and Kubernetes," Softw. Pract. Exp., vol. 50, no. 9, pp. 1793–1821, 2020, doi: 10.1002/spe.2848.

[37] D. Apolinário and B. de França, "A method for monitoring the coupling evolution of microservice-based architectures," J. Braz. Comput. Soc., vol. 27, no. 1, 2021, doi: 10.1186/s13173-021-00120-y.

[38] A. Mohan, "Kafka Streaming Application using Java Spring Boot," Mar. 2022, Accessed: Jul. 10, 2022. [Online]. Available: https://repository.cityu.edu/handle/20.500.11803/1782

[39] A. Alaasam, G. Radchenko, A. Tchernykh, and J. González Compeán, “Analytic Study of Containerizing Stateful Stream Processing as Microservice to Support Digital Twins in Fog Computing,” Program. Comput. Softw., vol. 46, no. 8, pp. 511–525, Dec. 2020, doi: 10.1134/S0361768820080083.

[40] M. Miloradović and A. Milovanović, “Data streaming architecture based on Apache Kafka and GitHub for tracking students’ activity in higher education software development courses,” E-Bus. Technol. Conf. Proc., vol. 2, no. 1, Art. no. 1, Jun. 2022.

[41] I. Ghani, W. M. N. Wan-Kadir, A. Mustafa, and M. I. Babir, “Microservice Testing Approaches: A Systematic Literature Review,” Int. J. Integr. Eng., vol. 11, no. 8, Art. no. 8, Dec. 2019.

[42] “Back-End Design and Development on Rekaruang Application with Microservices Architecture | JATISI (Jurnal Teknik Informatika dan Sistem Informasi),” Mar. 2022, Accessed: Jul. 10, 2022. [Online]. Available: https://jurnal.mdp.ac.id/index.php/jatisi/article/view/1391

[43] N. Chaudhry and M. M. Yousaf, “Concurrency control for real-time and mobile transactions: Historical view, challenges, and evolution of practices,” Concurr. Comput.-Pract. Exp., vol. 34, no. 3, p. e6549, Feb. 2022, doi: 10.1002/cpe.6549.

[44] M. Mena, A. Corral, L. Iribarne, and J. Criado, “A Progressive Web Application Based on Microservices Combining Geospatial Data and the Internet of Things,” Ieee Access, vol. 7, pp. 104577–104590, 2019, doi: 10.1109/ACCESS.2019.2932196.

[45] A. Suljkanovic, B. Milosavljevic, V. Indic, and I. Dejanovic, “Developing Microservice-Based Applications Using the Silvera Domain-Specific Language,” Appl. Sci.-Basel, vol. 12, no. 13, p. 6679, Jul. 2022, doi: 10.3390/app12136679.

[46] L. H. Silva, M. T. Valente, A. Bergel, N. Anquetil, and A. Etien, “Identifying Classes in Legacy JavaScript Code,” J. Softw.-Evol. Process, vol. 29, no. 8, p. e1864, Aug. 2017, doi: 10.1002/smr.1864.

[47] N. Bouraqadi and D. Mason, “Test-driven development for generated portable Javascript apps,” Science of Computer Programming, vol. 161, pp. 2–17, Sep. 2018, doi: 10.1016/j.scico.2018.02.003.

[48] S. Bosse, “A Unified System Modelling and Programming Language based on JavaScript and a Semantic Type System,” Procedia Manuf., vol. 24, pp. 21–39, Jan. 2018, doi: 10.1016/j.promfg.2018.06.005.

[49] A. Wirfs-Brock and B. Eich, “JavaScript: The First 20 Years,” Proc. Acm Program. Lang.-Pacmpl, vol. 4, p. 77, Jun. 2020, doi: 10.1145/3386327.

[50] L. Nkenyereye and J.-W. Jang, “Performance Evaluation of Server-side JavaScript for Healthcare Hub Server in Remote Healthcare Monitoring System,” Procedia Comput. Sci., vol. 98, pp. 382–387, Jan. 2016, doi: 10.1016/j.procs.2016.09.058.

[51] E. K. Kristensen and A. Moller, “Type Test Scripts for TypeScript Testing,” Proc. Acm Program. Lang.-Pacmpl, vol. 1, p. 90, Oct. 2017, doi: 10.1145/3133914.

[52] N. Black, “Boris Cherny on TypeScript,” IEEE Softw., vol. 37, no. 2, pp. 98–100, Mar. 2020, doi: 10.1109/MS.2019.2958155.

[53] Z. Wu, Z. Sun, K. Gong, L. Chen, B. Liao, and Y. Jin, “Hidden Inheritance: An Inline Caching Design for TypeScript Performance,” Proc. Acm Program. Lang.-Pacmpl, vol. 4, p. 174, Nov. 2020, doi: 10.1145/3428242.

[54] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, “Developing Docker and Docker-Compose Specifications: A Developers’ Survey,” Ieee Access, vol. 10, pp. 2318–2329, 2022, doi: 10.1109/ACCESS.2021.3137671.

[55] G. Vial, “Different Databases for Different Strokes,” IEEE Softw., vol. 35, no. 2, pp. 80–85, Mar. 2018, doi: 10.1109/MS.2018.1661308.

[56] J. Lieponiene, “Recent Trends in Database Technology,” Balt. J. Mod. Comput., vol. 8, no. 4, pp. 551–559, 2020, doi: 10.22364/bjmc.2020.8.4.06.

[57] A. Makris, K. Tserpes, G. Spiliopoulos, D. Zissis, and D. Anagnostopoulosl, “MongoDB Vs PostgreSQL: A comparative study on performance aspects,” Geoinformatica, vol. 25, no. 2, pp. 243–268, Apr. 2021, doi: 10.1007/s10707-020-00407-w.

[58] M. Ilba, “Parallel algorithm for improving the performance of spatial queries in SQL: The use cases of SQLite/SpatiaLite and PostgreSQL/PostGIS databases,” Comput. Geosci., vol. 155, p. 104840, Oct. 2021, doi: 10.1016/j.cageo.2021.104840.

[59] P. Sha, S. Chen, L. Zheng, X. Liu, H. Tang, and Y. Li, “Design and Implement of Microservice System for Edge Computing,” IFAC-Pap., vol. 53, no. 5, pp. 507–511, Jan. 2020, doi: 10.1016/j.ifacol.2021.04.137.

[60] D. V. Levshin and A. S. Markov, “Algorithms for integrating PostgreSQL with the semantic web,” Program. Comput. Softw., vol. 35, no. 3, pp. 136–144, May 2009, doi: 10.1134/S0361768809030025.

[61] W. Schultz, T. Avitabile, and A. Cabral, “Tunable Consistency in MongoDB,” Proc. Vldb Endow., vol. 12, no. 12, pp. 2071–2081, Aug. 2019, doi: 10.14778/3352063.3352125.

[62] “Beginners Guide: MongoDB Basics,” MongoDB. https://www.mongodb.com/basics (accessed Aug. 12, 2022).

[63] S. Peltonen, L. Mezzalira, and D. Taibi, “Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review,” Inf. Softw. Technol., vol. 136, p. 106571, Aug. 2021, doi: 10.1016/j.infsof.2021.106571.

[64] A. Stojkov and Ž. Stojanov, “Review of methods for migrating software systems to microservices architecture,” 2021.

[65] M. Kaluza, K. Troskot, and B. Vukelic, “Comparison of Front-End Frameworks for Web Applications Development,” Zb. Veleucilista U Rijeci-J. Polytech. Rij., vol. 6, no. 1, pp. 261–282, May 2018, doi: 10.31784/zvr.6.1.19.

[66] “Vue.js.” https://vuejs.org/guide/introduction.html (accessed Aug. 10, 2022).

[67] J. Song, M. Zhang, and H. Xie, “Design and Implementation of a Vue.js-Based College Teaching System,” Int. J. Emerg. Technol. Learn., vol. 14, no. 13, pp. 59–69, 2019, doi: 10.3991/ijet.v14i13.10709.

[68] “nuxt.js,” Nuxt. https://nuxtjs.org/docs/get-started/installation (accessed Aug. 10, 2022).

[69] “Introduction to Node.js.” https://nodejs.dev/learn (accessed Aug. 10, 2022).

[70] K. Tserpes, M. Pateraki, and I. Varlamis, “Strand: scalable trilateration with Node.js,” J. Cloud Comput., vol. 8, no. 1, p. 16, Nov. 2019, doi: 10.1186/s13677-019-0142-y.

[71] “Express - Infraestructura de aplicaciones web Node.js.” http://expressjs.com/es/ (accessed Aug. 17, 2022).

[72] D. Bauer, B. Penz, J. Mäkiö, and M. Assaad, “Improvement of an Existing Microservices Architecture for an E-learning Platform in STEM Education,” Jul. 2018.

[73] P. Bohora, “Design and Develop Decentralized Microservices Architecture with Docker Container,” 2021. Accessed: Jul. 08, 2022. [Online]. Available: http://www.theseus.fi/handle/10024/505751

[74] P. Quevedo-Avila, M. G. Zhindón-Mora, and A. S. Quevedo-Sacoto, “Arquitectura de microservicios para compras en línea: caso de uso ‘ala orden,’” Polo Conoc., vol. 5, no. 1, p. 151, 2020, doi: 10.23857/pc.v5i1.1884.

[75] X. Wan, X. Guan, T. Wang, G. Bai, and B.-Y. Choi, “Application deployment using Microservice and Docker containers: Framework and optimization,” J. Netw. Comput. Appl., vol. 119, pp. 97–109, Oct. 2018, doi: 10.1016/j.jnca.2018.07.003.

[76] Comisión De Consolidación Del Informe De Rendición Anual De Cuentas 2021, “Número De Estudiantes 2021,” 2022. [Online]. Available: https://www.utmachala.edu.ec/archivos/filesportal/2022/RENDICION_DE_CUENTAS/N%C3%BAmero_de_Estudiantes_RC_2021.pdf

[77] Educación Continua de la Universidad Técnica de Machala, “Educación Continua - UTMACH,” Jul. 2022. https://cec.utmachala.edu.ec/curso/3/

[78] F. A. Rojas Vilela, “Entrevista del estado actual del sistema de matriculación Centro de Educación Continua de la UTMACH,” Jul. 2022.

ANEXOS

Anexo 1 - Entrevista del estado actual del sistema de matriculación Centro de Educación Continua de la UTMACH

UNIVERSIDAD TÉCNICA DE MACHALA
D.L. No. 69-04 DE 14 DE ABRIL DE 1969
Calidad, Pertinencia y Calidez
UNIDAD ACADÉMICA DE INGENIERÍA CIVIL

Machala, 1 de julio del 2022.

Entrevista del estado actual del sistema de matriculación Centro de Educación Continua de la UTMACH

Preguntas generales

- **¿Qué metodología se utiliza para el desarrollo de sistemas?**
  En general se aplica una metodología incremental, lo que implica establecer varias etapas del desarrollo haciendo entregas en cada una de ellas, permitiendo una mayor flexibilidad en relación al uso de la aplicación y el feedback que provee el usuario.

- **¿Existen reportes o registros de rendimiento o fallas en el sistema de matriculación del Centro de Educación Continua en periodos de matriculación?**
  El servidor de aplicaciones provee un sistema de log rotativos en los cuales se registran las posibles incidencias en relación a las peticiones que recibe la aplicación. En relación al rendimiento, es posible revisar los reportes del sistema de virtualización implementado, y así determinar los momentos en los cuales el consumo de recursos se incrementa.

- **¿En periodos de matriculación, cuál es el número de estudiantes que realizan el proceso de matriculación al mismo tiempo?**
  Por la naturaleza de la aplicación es un esquema de competición por llegar primero o esperar ser atendido por el servidor en un lapso de tiempo determinado. El número de estudiantes exacto no es posible determinarlo; esto debido a que, el proceso se compone de múltiples peticiones. Y en general jugamos con la voluntad del usuario, por llamarlo de alguna manera; por ejemplo, aun sabiendo que no le corresponde la matrícula ingresa a para ver si puede hacerlo, y pues ya forma parte del gran cúmulo.

- **¿En periodos de no matriculación, cual es el número máximo de estudiantes que ingresan a la plataforma?**
  En este caso, no hay otra necesidad de ingresar al sistema de matriculación parte de los estudiantes, más que para matrícula; a no ser, que se trate de revisar los resultados para los exámenes de ubicación. Por lo tanto, el ingreso es escaso, más que del personal del Centro de Educación Continua que en promedio serían 4 – 8 personas.

- **¿Cuál es el límite máximo de peticiones que soporta el sistema del Centro de Educación Continua de la UTMACH?**
  Para este punto, es necesario considerar el hecho que la aplicación de CEC comparte el servidor con otras aplicaciones de la universidad, por lo tanto, volvemos al esquema de competición de recursos. En su lugar podríamos cambiar la pregunta por peticiones que soporta la infraestructura sobre la cuál corre la aplicación; recordemos que, las limitaciones siempre son relativas al hardware, la capacidad del software es infinita.

- **¿Cuál o cuáles gestores de base de datos usan para el registro de matrículas, registro de estudiantes y login?**
  PostreSQL

- **¿Existe documentación del desarrollo del sistema (ERS) (requerimientos funcionales y no funcionales, modelo relacional, etc)?**
  NO

- **¿Qué estándares de seguridad y calidad utilizan en el departamento de sistemas?**
  Esquema de Roles y permisos por usuario o grupo de usuarios para determinar a que tiene acceso o no cada uno. En relación a la calidad, es evaluada por nada desarrollador, considerando los aspectos de análisis, pruebas, código limpio, refactorización. Lo óptimo sería tener un grupo QA que lo haga para garantizar la imparcialidad en los puntos mencionados.

Preguntas técnicas

- **¿Qué arquitectura de software implementa el sistema de matriculación del Centro de Educación Continua de la UTMACH?**
  MVC (Modelo Vista Controlador)

- **¿Qué tipo de servidor usan para servir el sitio web (nginx/apache)?**
  El servidor de aplicaciones utilizado es Apache.

- **¿Qué técnica de rendering (SPA/SSR/SSG) utiliza el sistema de matriculación del Centro de Educación Continua de la UTMACH?**
  Server-Side Rendering - SSR

- **¿El servidor donde se encuentra alojado el servicio implementa algún tipo de tecnología/ técnica para evitar la saturación del servicio (Load balancer / bloqueador de peticiones por IP/ otros)?**
  Actualmente se cuenta con la protección de un WAF para las aplicaciones, mismo que es proporcionad por el ISP.

- **¿Cuáles son las características técnicas del servidor en donde se encuentra alojado el sistema de matriculación del Centro de Educación Continua de la UTMACH?**
  - Memoria RAM
  - Almacenamiento
  - CPU
  - Sistema operativo

  No se puede proporcionar esta información.

Anexo 2 – Formato de resultados obtenidos en bruto por peticiones realizadas

Time Stamp	elapsed	label	Response Code	Response Message	Thread Name	Data Type	success	Failure Message	bytes	Sent Bytes	Grp Threads	All Threads	URL	Latency	Sample Count	Error Count	Idle Time	Connect

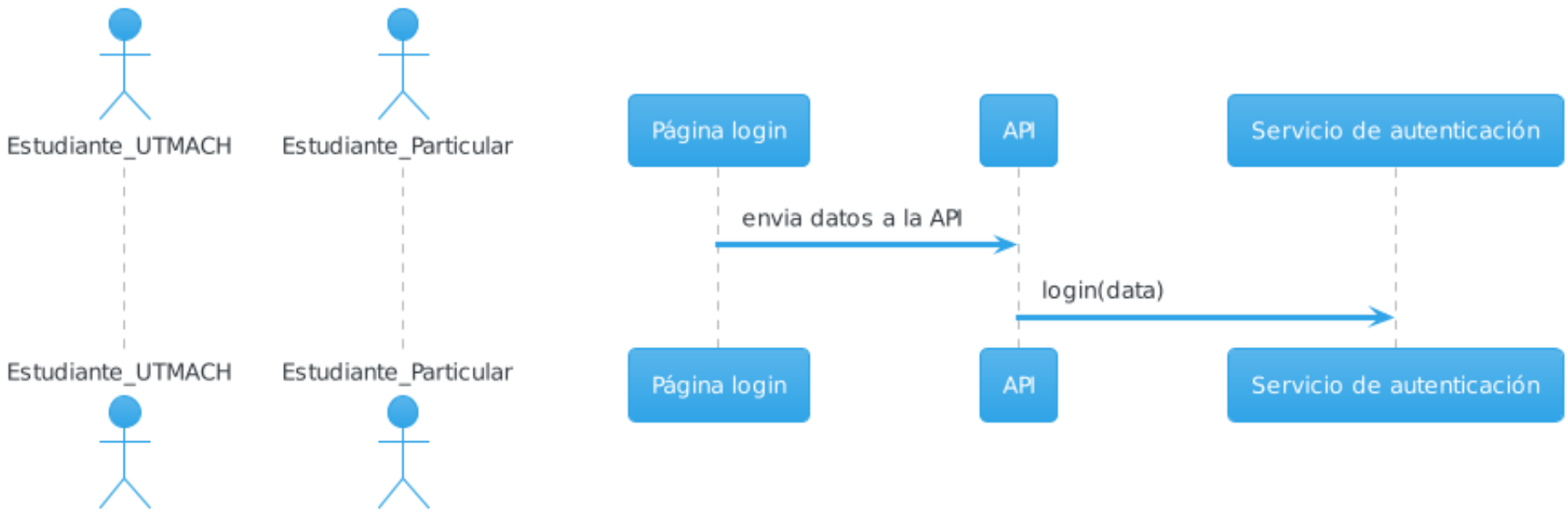
Anexo 3 – Formato de reporte de resultados

Reporte							
Nombre del archivo:							
Etiqueta	#Muestras	Media	Min	Max	Desv. Estándar	%Error	Rendimiento
Total:							

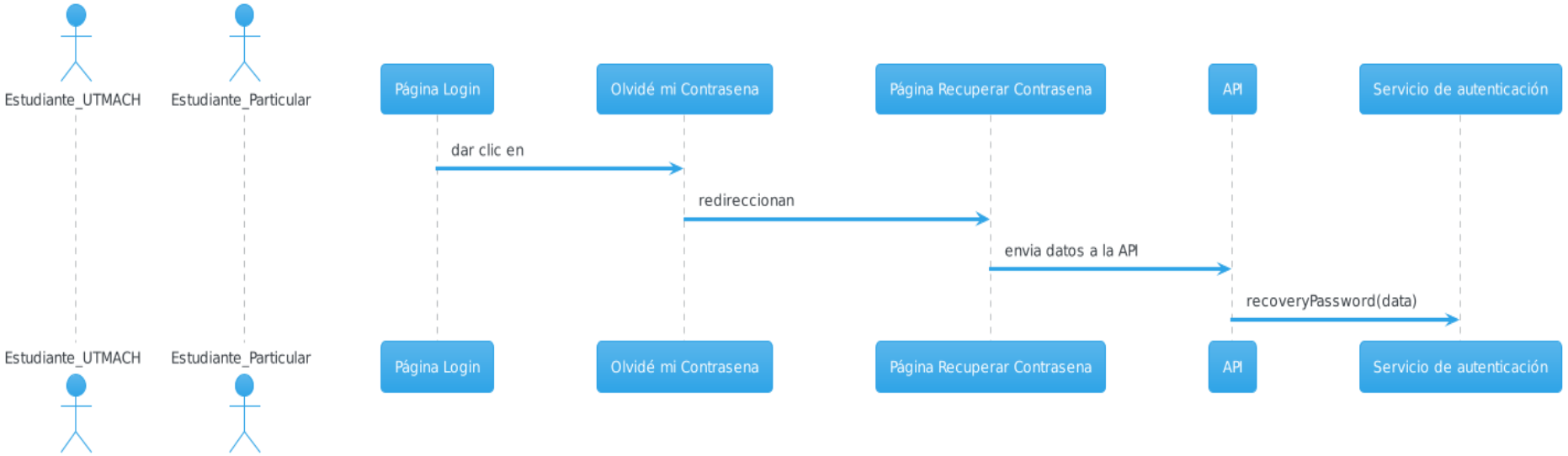
APÉNDICES

Apéndice 1 - Patrones de interacción usando diagramas de secuencia

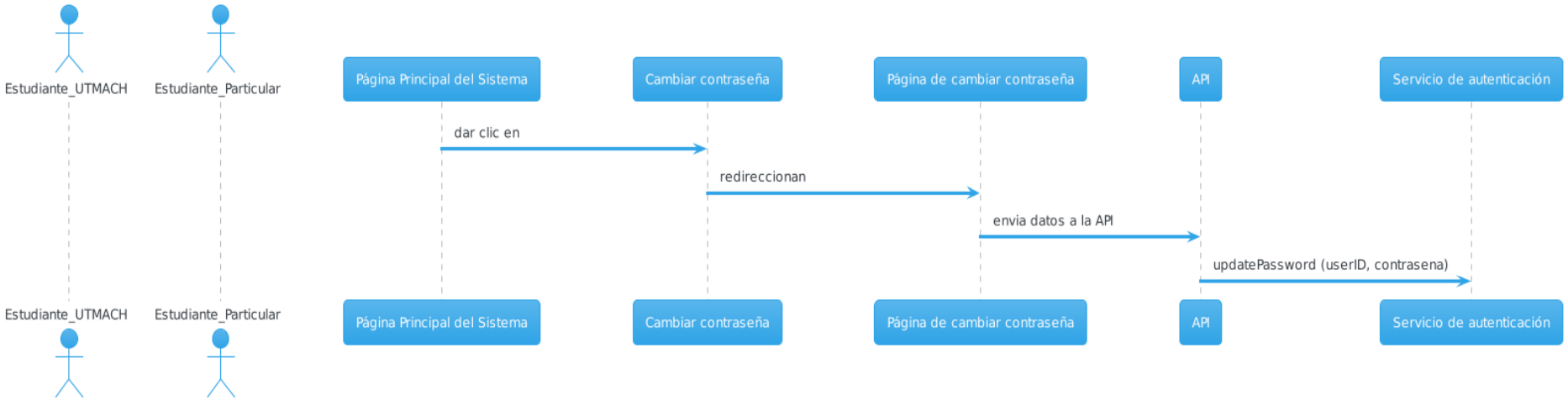
Apéndice 1.1. Diagrama de secuencia de iniciar sesión.



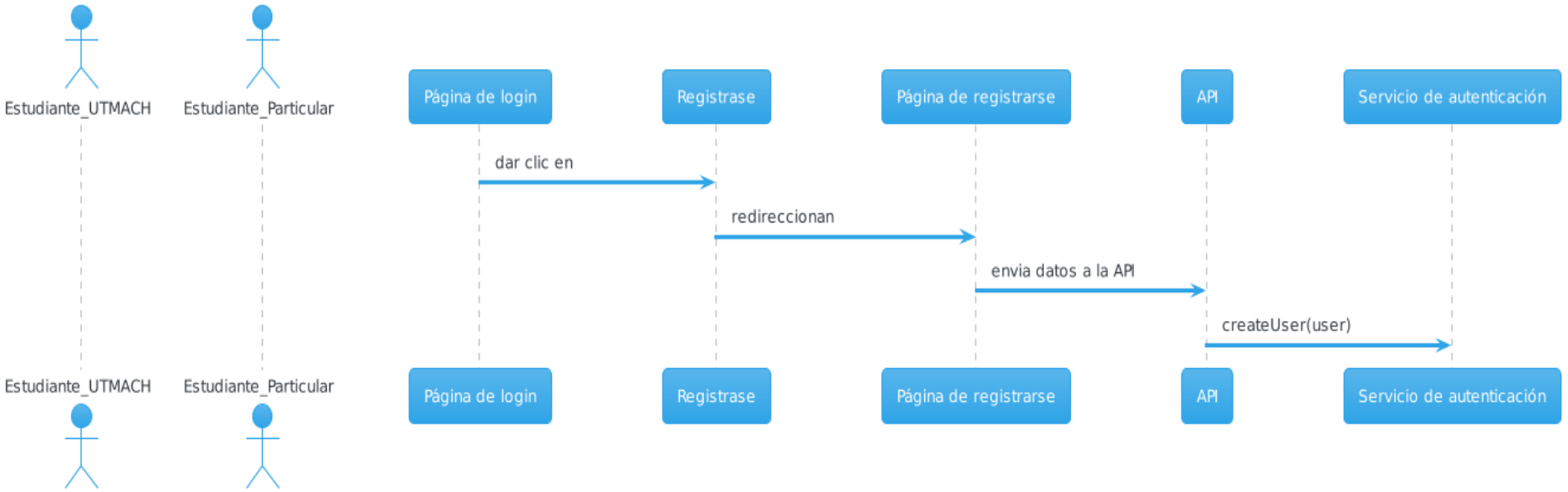
Apéndice 1.2. Diagrama de secuencia de recuperar contraseña.



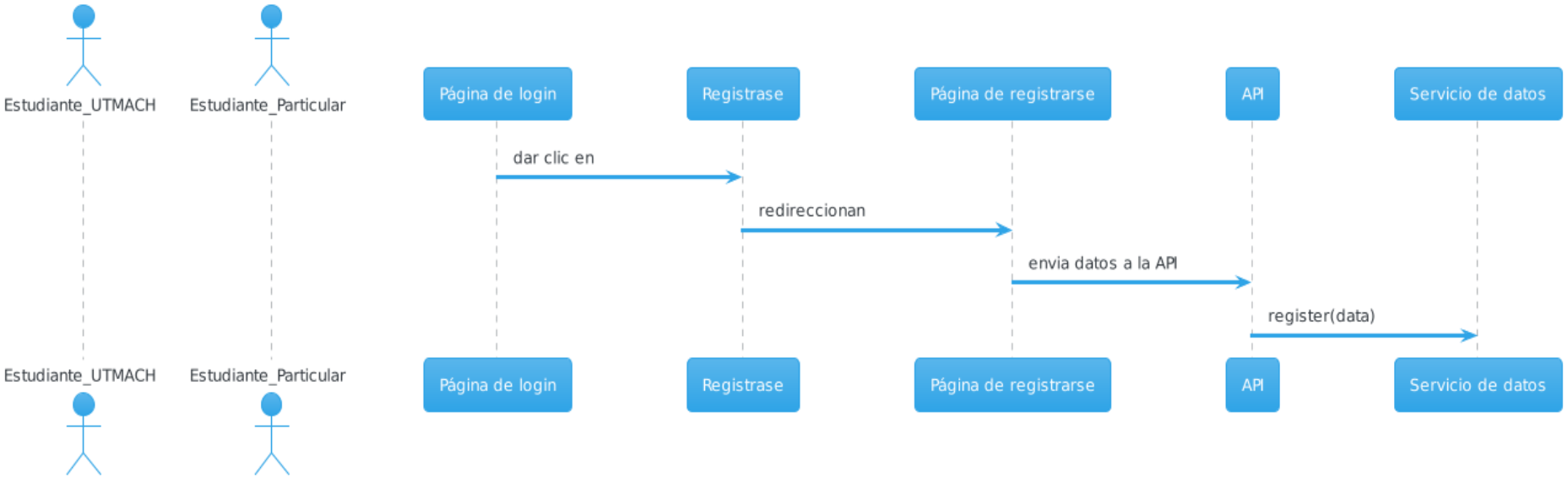
Apéndice 1.3. Diagrama de secuencia de actualizar contraseña.



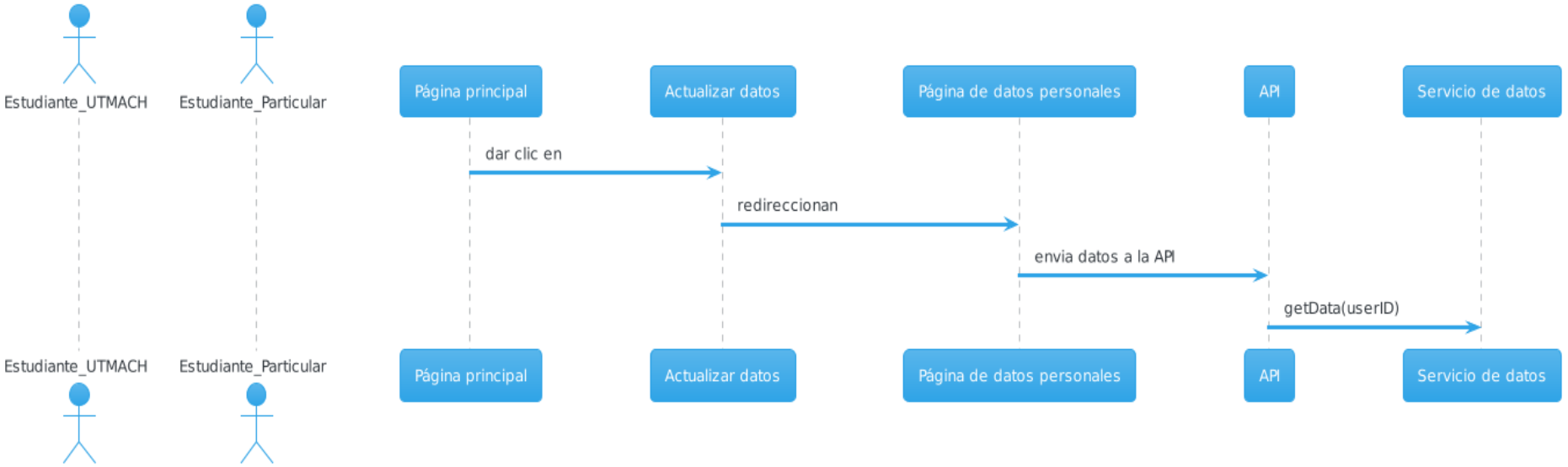
Apéndice 1.4. Diagrama de secuencia de registrarse.



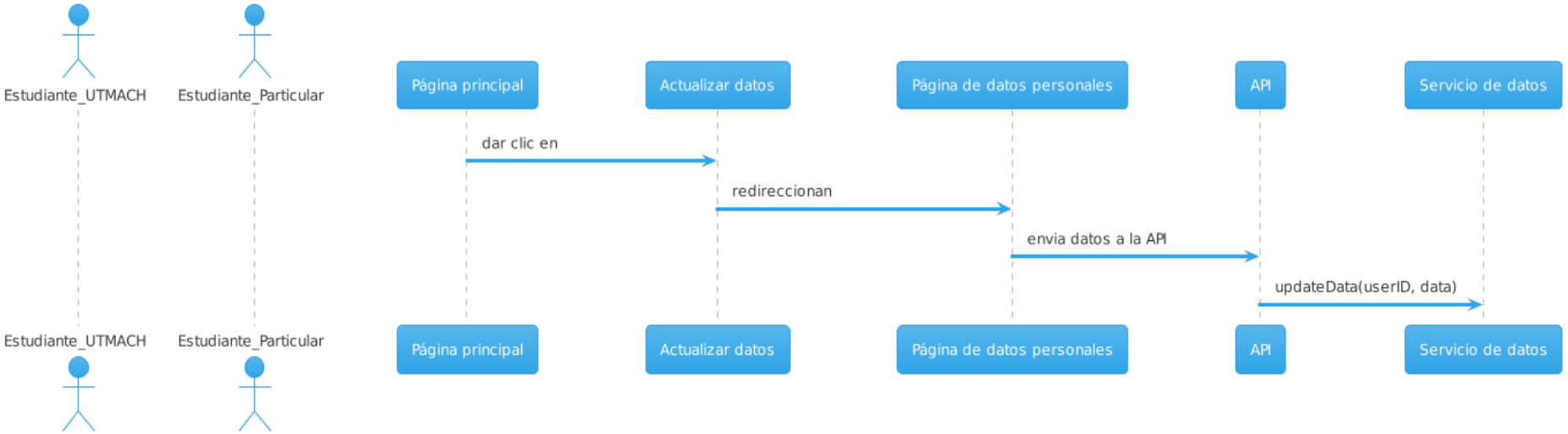
Apéndice 1.5. Diagrama de secuencia de registrar datos personales.



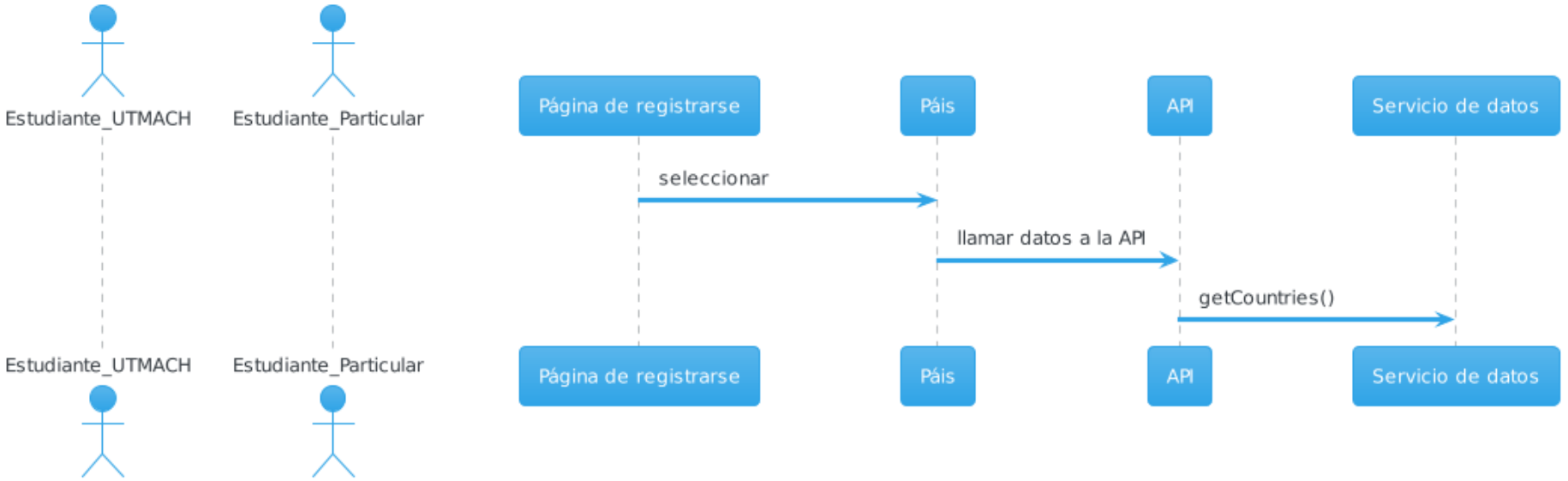
Apéndice 1.6. Diagrama de secuencia de visualizar datos personales.



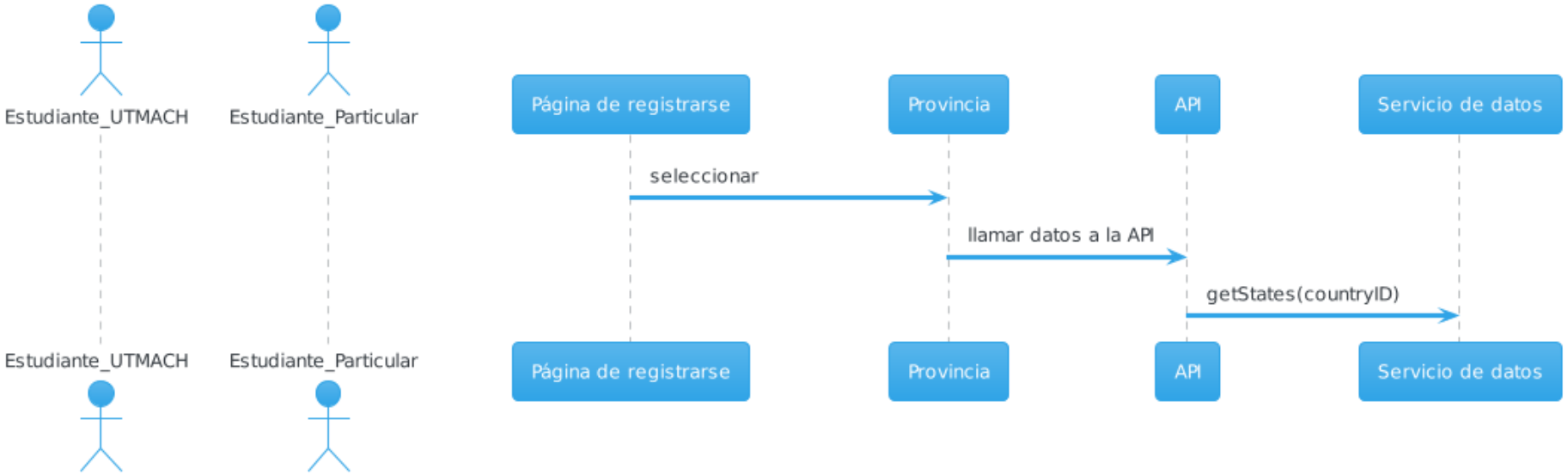
Apéndice 1.7. Diagrama de secuencia de editar datos personales.



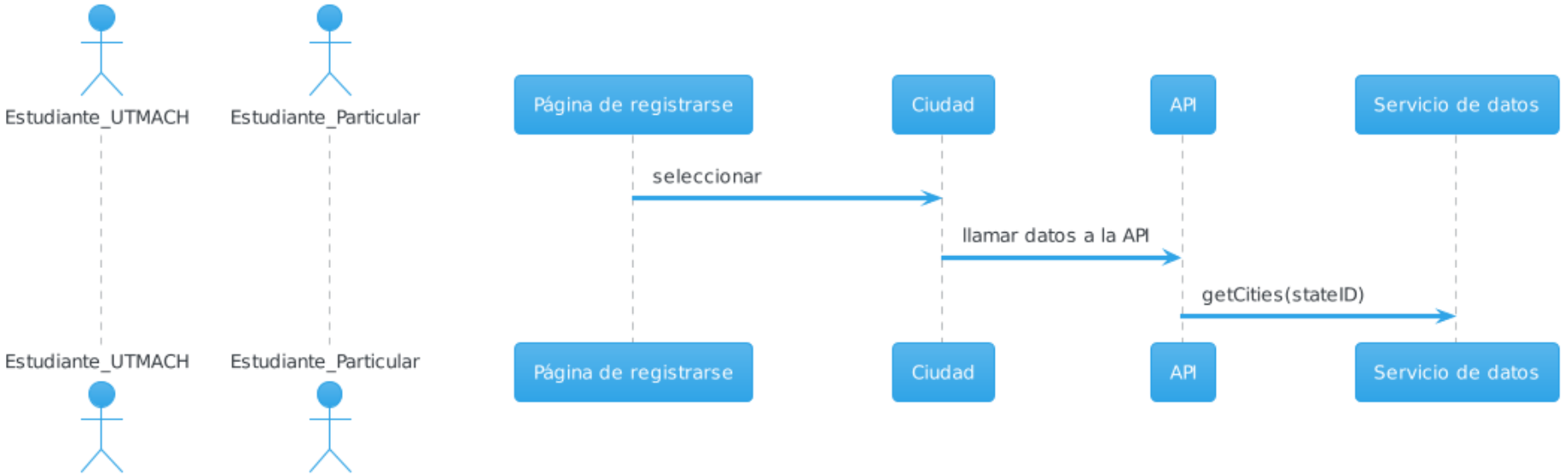
Apéndice 1.8. Diagrama de secuencia de visualizar país.



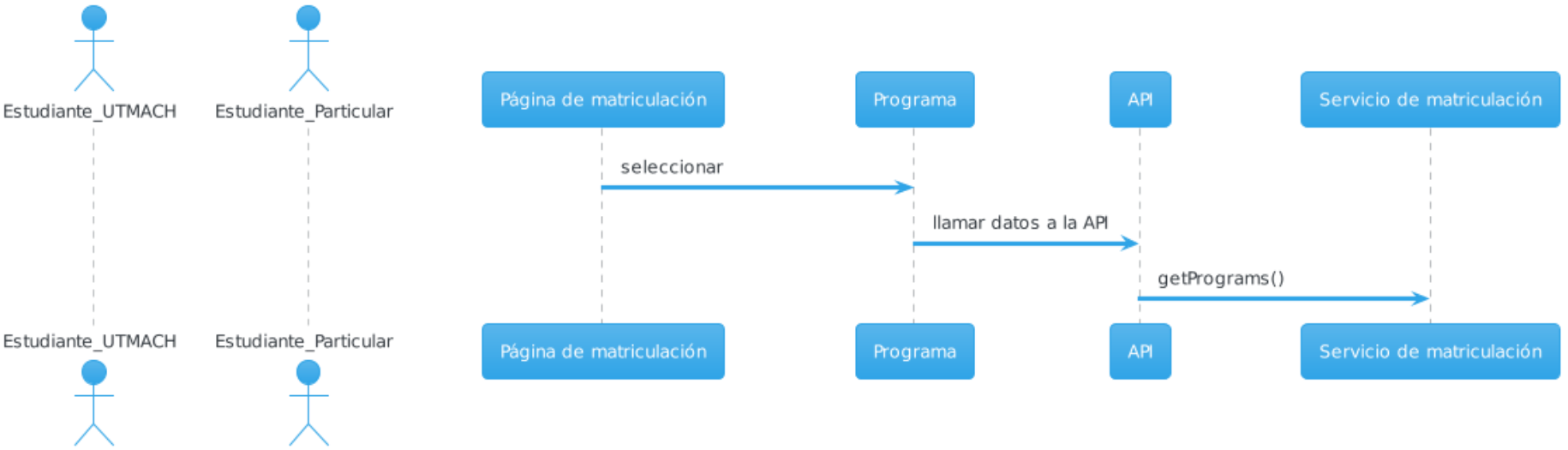
Apéndice 1.9. Diagrama de secuencia de visualizar provincia de un país.



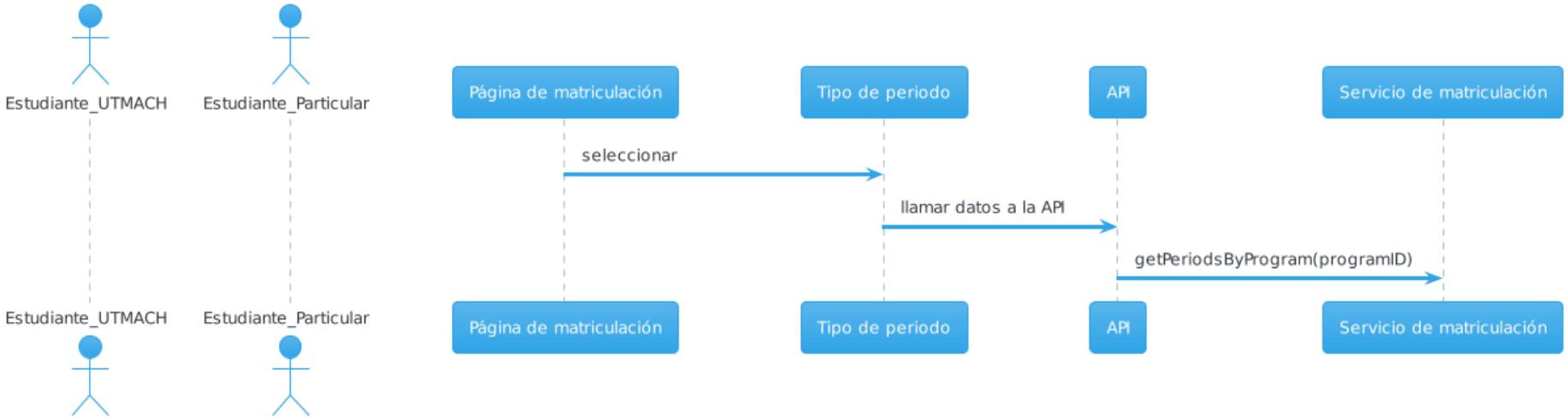
Apéndice 1.10. Diagrama de secuencia de visualizar ciudades de una provincia.



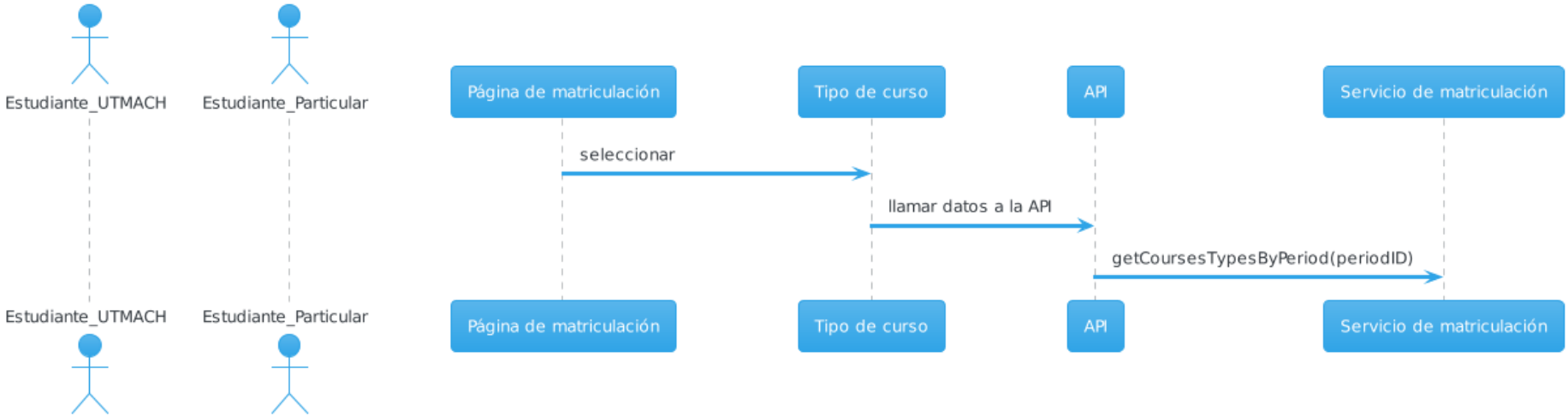
Apéndice 1.11. Diagrama de secuencia de visualizar programas ofrecidos.



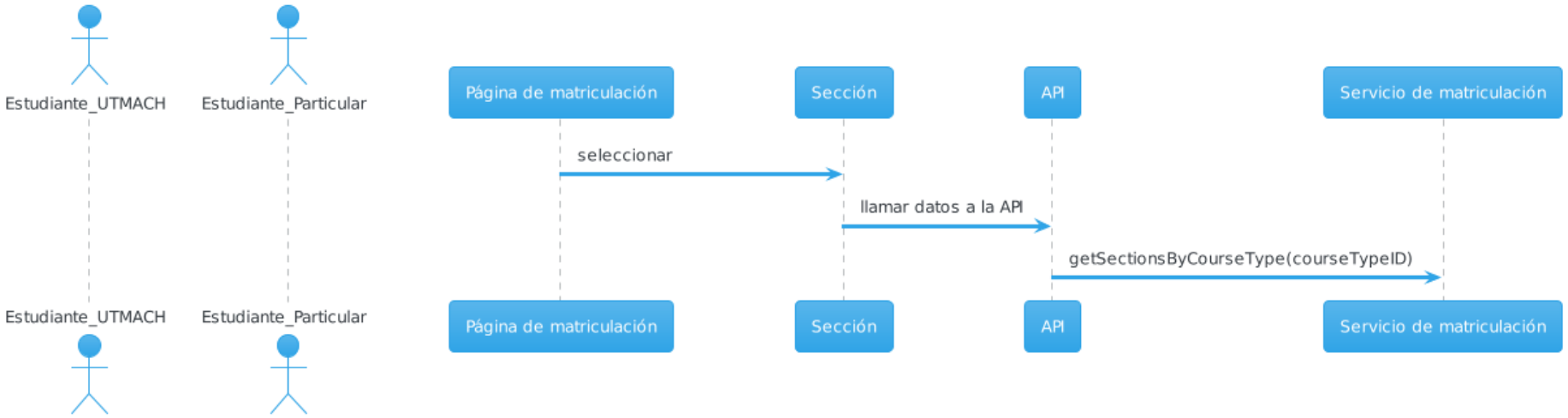
Apéndice 1.12. Diagrama de secuencia de visualizar los tipos de periodos.



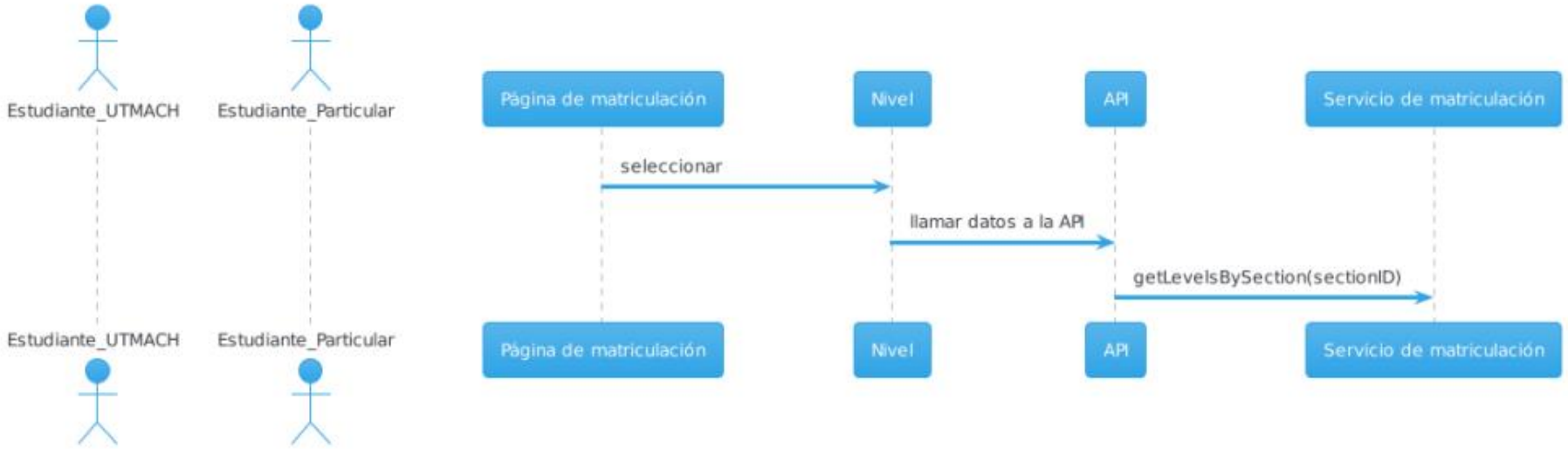
Apéndice 1.13. Diagrama de secuencia de visualizar los tipos de cursos.



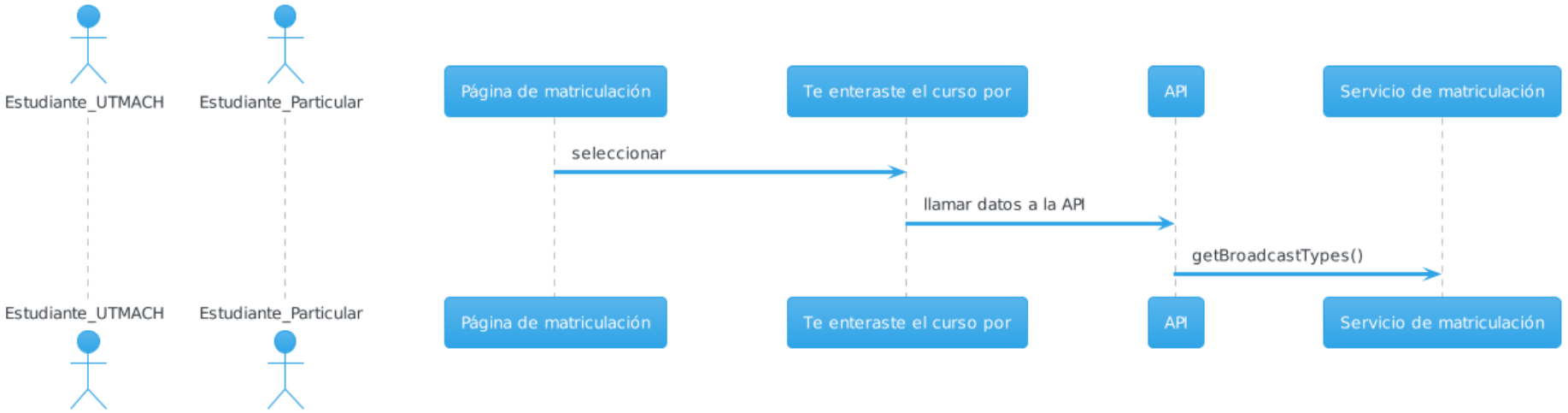
Apéndice 1.14. Diagrama de secuencia de visualizar las secciones.



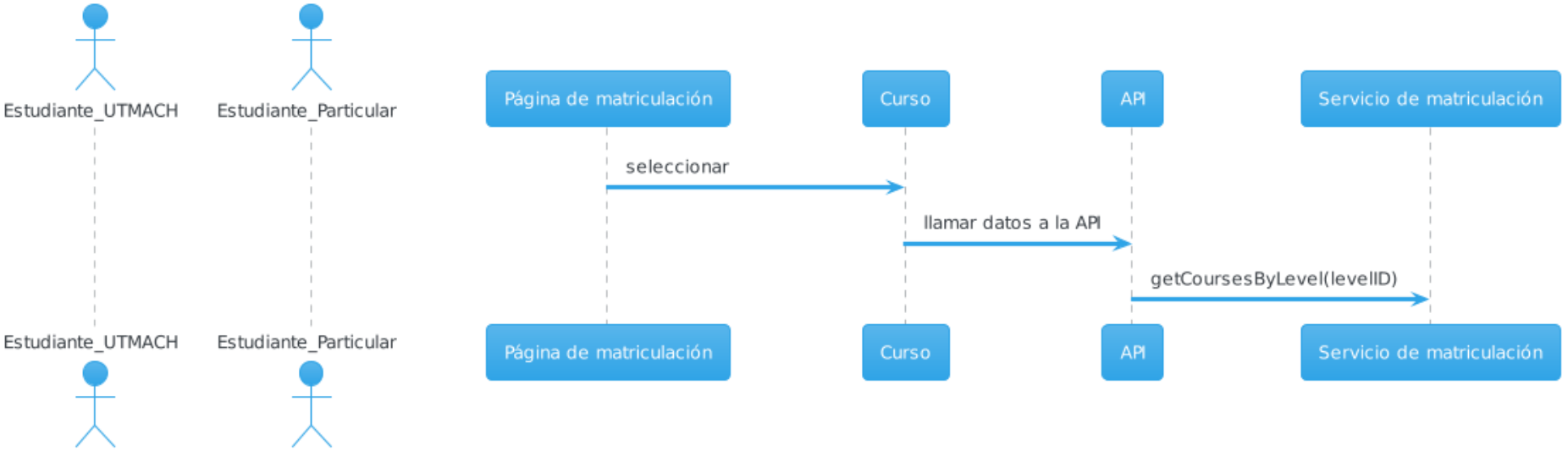
Apéndice 1.15. Diagrama de secuencia de visualizar los niveles.



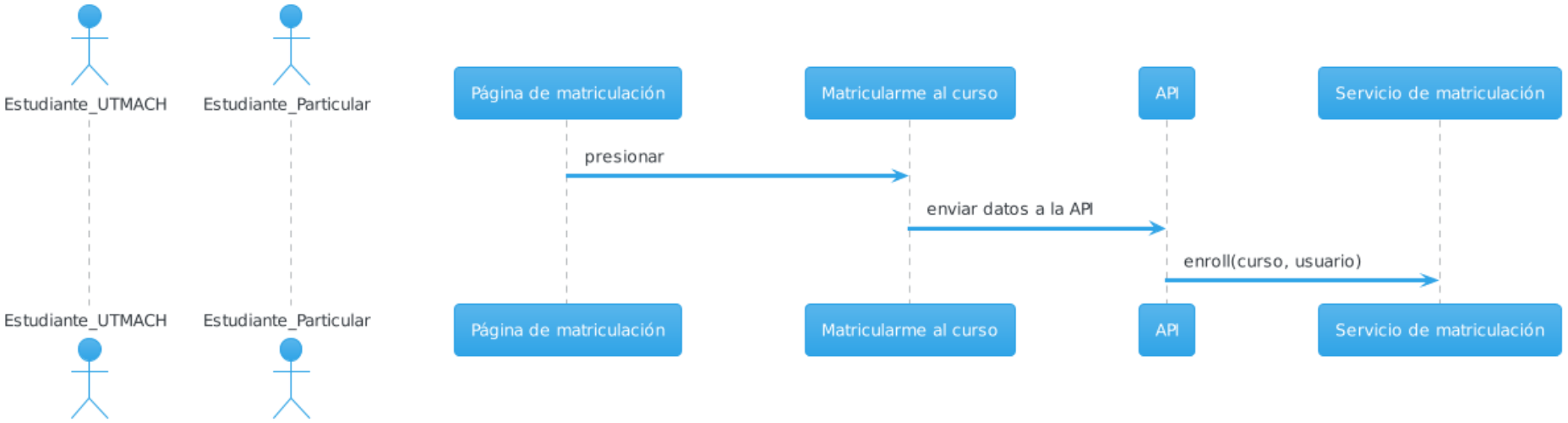
Apéndice 1.16. Visualizar medios de difusión (te enteraste del curso por).



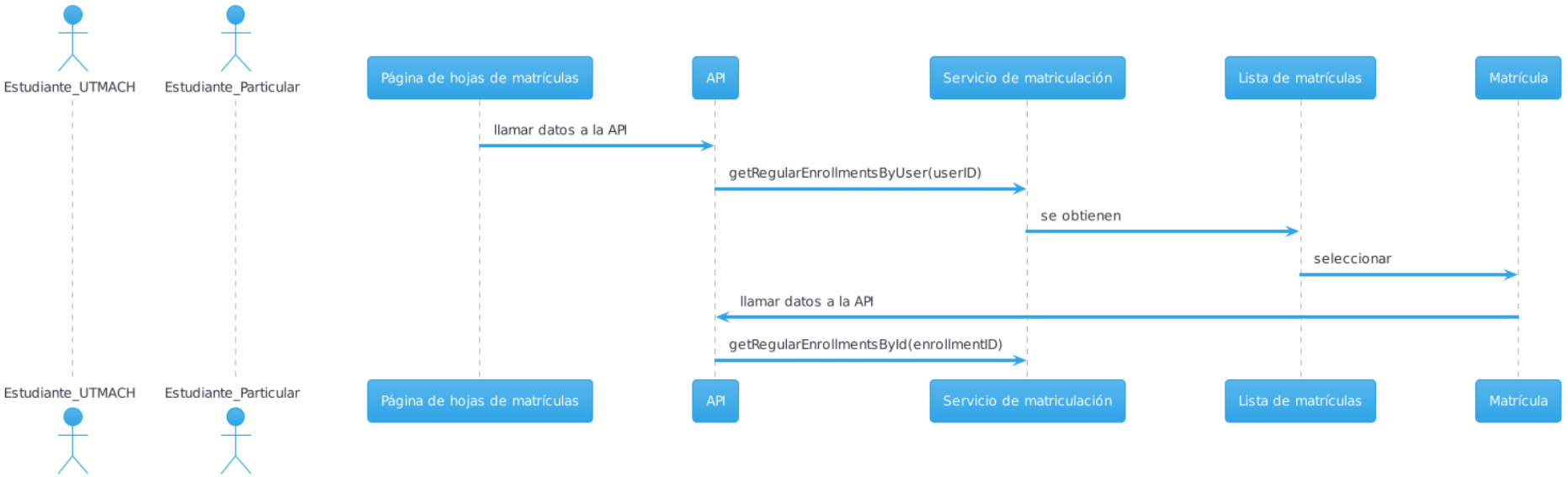
Apéndice 1.17. Diagrama de secuencia de visualizar los cursos.



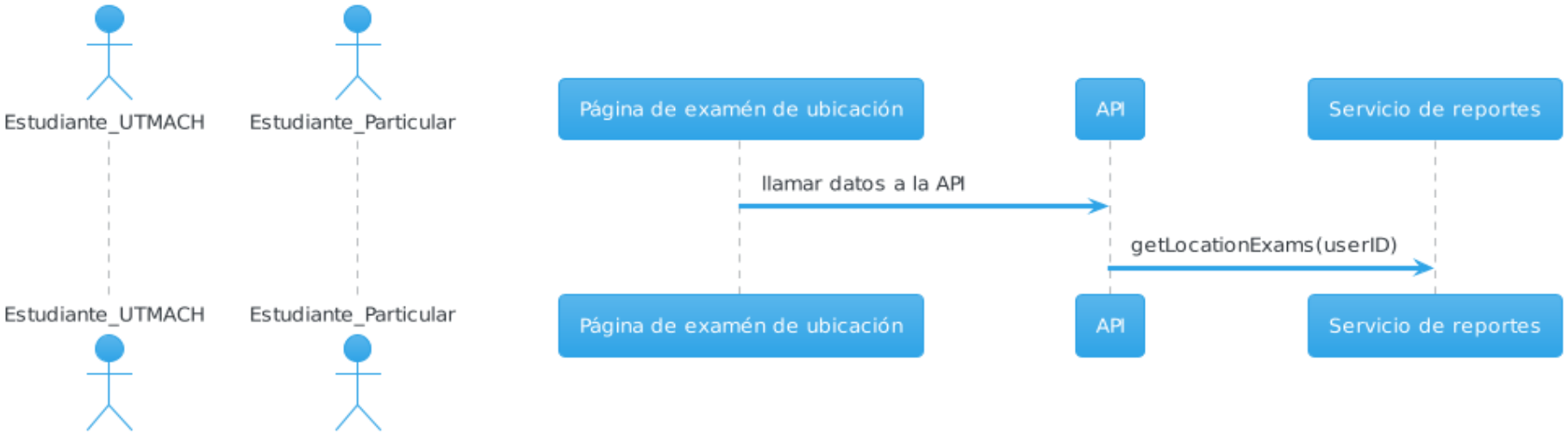
Apéndice 1.18. Diagrama de secuencia de matricularse.



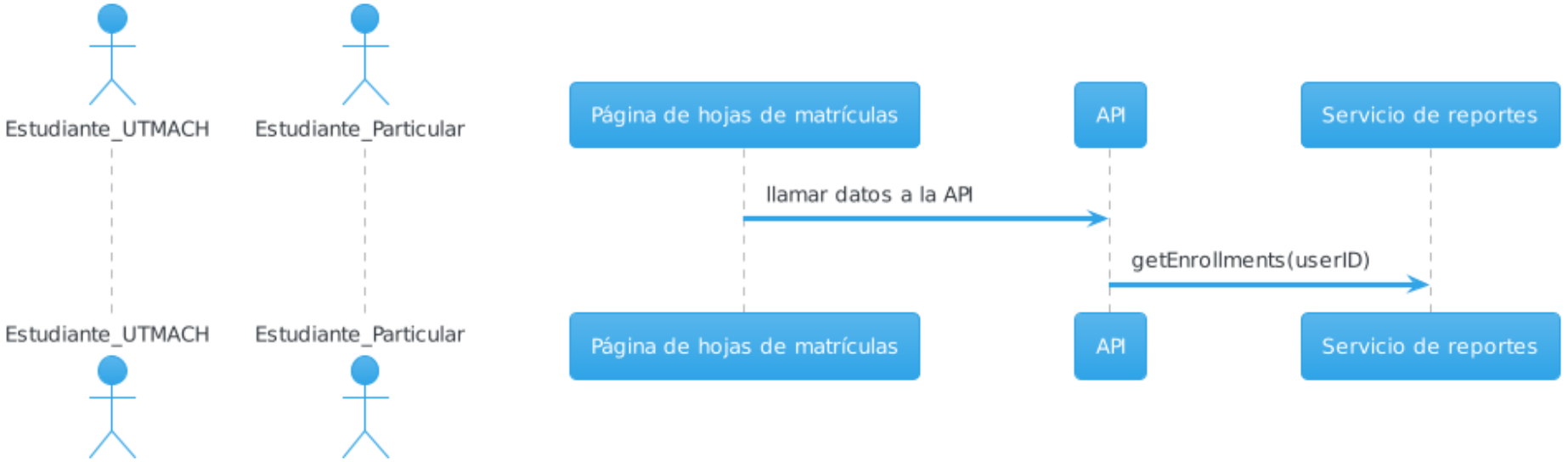
Apéndice 1.19. Diagrama de secuencia de visualizar los datos de la matricula.



Apéndice 1.20. Diagrama de secuencia de resultados de examen de ubicación.



Apéndice 1.21. Diagrama de secuencia de visualizar lista de matrículas.



Apéndice 2 - Diseño de la API con una especificación de estándar abierto

Apéndice 2.1. Diseño de los métodos HTTP de la API en el servicio de autenticación.

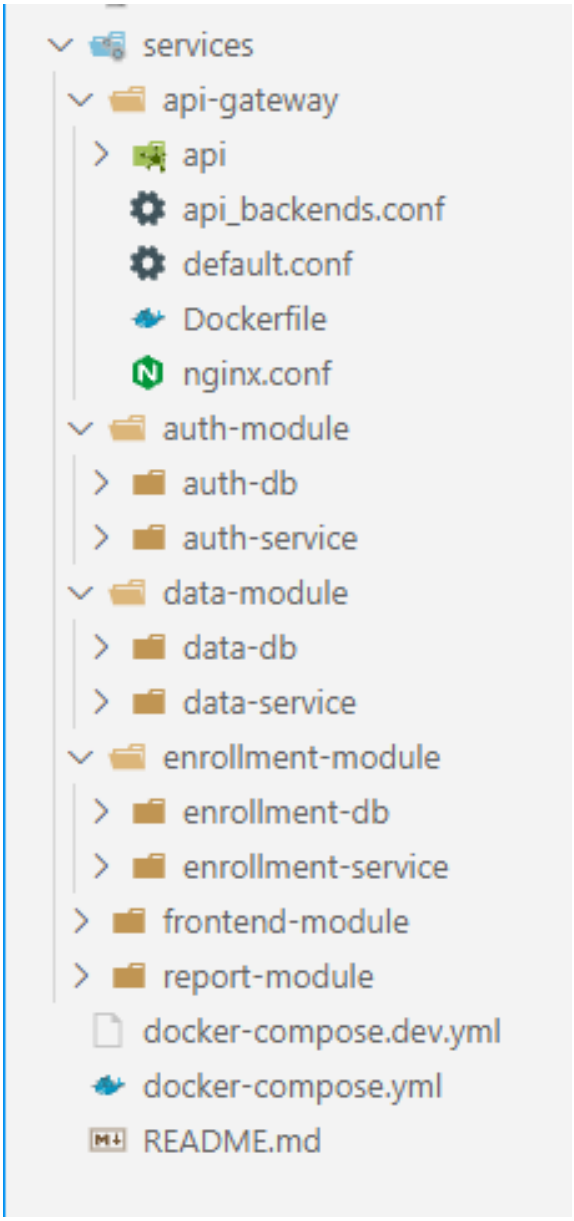
Apéndice 2.2. Diseño de los métodos HTTP de la API en el servicio de datos.

Apéndice 2.3. Diseño de los métodos HTTP de la API en el servicio de matriculación.

Apéndice 2.4. Diseño de los métodos HTTP de la API en el servicio de reportes.

Apéndice 3 - Desarrollo de la infraestructura de comunicación

Apéndice 3. 1. Estructura de carpetas del proyecto prototipo.



Apéndice 3.2. Configuración de la imagen para el contenedor del aplicativo web.

Dockerfile

services > frontend-module > web-app > Dockerfile > ...

```
# --- RELEASE PHASE ---
FROM node:16.14.2-alpine3.15 AS builder

WORKDIR /usr/app
ENV SERVER $SERVER

COPY package.json package-lock.json ./
RUN npm ci

COPY . .

RUN npm run build

# --- RELEASE PHASE ---
FROM node:16.14.2-alpine3.15

WORKDIR /usr/app
ENV NODE_ENV production
ENV SERVER $SERVER
ENV PORT $PORT

COPY package.json package-lock.json ./
RUN npm ci --production=true

COPY --from=builder /usr/app/.nuxt ./.nuxt/
ADD nuxt.config.js ./

EXPOSE ${PORT}
CMD ["npm", "start"]

```

Apéndice 3.3. Configuración de servicio para el prototipo web.

docker-compose.yml

```yaml
version: "3"
services:
  # frontend services
  web_app:
    build: ./services/frontend-module/web-app
    ports:
      - "80:5001"
    environment:
      - PORT=5001
      # physical machine (ip host)
      - SERVER=192.168.1.6
```

Apéndice 3.4. Configuración de la imagen para los diferentes contenedores de microservicios.

Dockerfile

services > auth-module > auth-service > Dockerfile > ...

```dockerfile
# --- BUILDER PHASE ---
FROM node:16.14.2-alpine3.15 AS builder

WORKDIR /usr/app

COPY package.json package-lock.json ./
RUN npm ci

COPY esbuild.config.js tsconfig.json ./
COPY ./src ./src
COPY index.ts ./

RUN npm run build

# --- RELEASE PHASE ---
FROM node:16.14.2-alpine3.15

WORKDIR /usr/app
ENV NODE_ENV production
ENV PORT $PORT

COPY package.json package-lock.json ./
RUN npm ci --production=true

COPY --from=builder /usr/app/dist ./dist

EXPOSE ${PORT}
CMD ["node", "dist/index.js"]
```

Apéndice 3.5. Configuración de servicio para los diferentes microservicios.

docker-compose.yml

```yaml
auth_service_1:
  build: ./services/auth-module/auth-service
  environment:
    - PORT=5001
    - NODE_NUMBER=1
    - SERVICE_NAME="AUTH SERVICE 1"
    - EMAIL_SENDER=false

auth_service_2:
  build: ./services/auth-module/auth-service
  environment:
    - PORT=5002
    - NODE_NUMBER=2
    - SERVICE_NAME="AUTH SERVICE 2"
    - EMAIL_SENDER=false

auth_service_3:
  build: ./services/auth-module/auth-service
  environment:
    - PORT=5003
    - NODE_NUMBER=3
    - SERVICE_NAME="AUTH SERVICE 3"
    - EMAIL_SENDER=false
auth_service_4:
  build: ./services/auth-module/auth-service
  environment:
    - PORT=5004
    - NODE_NUMBER=4
    - SERVICE_NAME="AUTH SERVICE 4"
    - EMAIL_SENDER=false
auth_service_5:
  build: ./services/auth-module/auth-service
  environment:
    - PORT=5005
    - NODE_NUMBER=5
    - SERVICE_NAME="AUTH SERVICE 5"
    - EMAIL_SENDER=false
```

Apéndice 3.6. Configuración de la imagen para los contenedores de servicio de base de datos basados en PostgreSQL.

Dockerfile

services > auth-module > auth-db > Dockerfile > ...

```
# --- RELEASE PHASE ---
FROM postgres:alpine

ENV POSTGRES_PASSWORD $POSTGRES_PASSWORD
ENV POSTGRES_USER $POSTGRES_USER
ENV POSTGRES_DB $POSTGRES_DB

COPY database.sql /docker-entrypoint-initdb.d/

```

Apéndice 3.7. Configuración de la imagen para el contenedor de servicio de base de datos basado en MongoDB.

Dockerfile

services > enrollment-module > enrollment-db > Dockerfile > ...

```
# --- RELEASE PHASE ---
FROM mongo:focal

ENV MONGO_INITDB_ROOT_USERNAME $MONGO_INITDB_ROOT_USERNAME
ENV MONGO_INITDB_ROOT_PASSWORD $MONGO_INITDB_ROOT_PASSWORD
ENV MONGO_INITDB_DATABASE $MONGO_INITDB_DATABASE

COPY database.js /docker-entrypoint-initdb.d/

```

Apéndice 3.8. Script de configuración inicial para el módulo de autenticación.

database.sql

services > auth-module > auth-db > database.sql

```
ALTER SYSTEM SET max_connections = 100000;

CREATE SCHEMA "cec_auth";

CREATE TABLE "cec_auth"."users" (
  "id_usuario" INT PRIMARY KEY,
  "usuario" text NOT NULL,
  "nombres" text NOT NULL,
  "contrasena" text NOT NULL
);
```

Apéndice 3.9. Script de configuración inicial para el módulo de datos.

database.sql

services > data-module > data-db > database.sql

```sql
ALTER SYSTEM SET max_connections = 100000;

CREATE SCHEMA "cec_data";

CREATE TABLE "cec_data"."estudiante" (
  id_usuario INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
  tipo_documento CHAR,
  numero_documento TEXT UNIQUE,
  nombres TEXT,
  apellidos TEXT,
  estado_civil CHAR,
  sexo CHAR,
  tipo_discapacidad CHAR,
  porcentaje_discapacidad CHAR,
  fecha_nacimiento DATE,
  pais INT,
  provincia INT,
  ciudad INT,
  direccion TEXT,
  telefono_fijo TEXT,
  telefono_movil TEXT,
  email TEXT
  );

```

Apéndice 3.10. Script de configuración inicial para el módulo de matriculación.

database.js

services > enrollment-module > enrollment-db > database.js > ...

```javascript
// create user
db.createUser({
  user: process.env.MONGO_INITDB_ROOT_USERNAME,
  pwd: process.env.MONGO_INITDB_ROOT_PASSWORD,
  roles: [{ role: "readWrite", db: process.env.MONGO_INITDB_DATABASE }],
});

// create collections
db.createCollection("programs", { validationLevel: "strict" });
db.createCollection("periods", { validationLevel: "strict" });
db.createCollection("course_types", { validationLevel: "strict" });
db.createCollection("sections", { validationLevel: "strict" });
db.createCollection("levels", { validationLevel: "strict" });
db.createCollection("courses", { validationLevel: "strict" });
db.createCollection("enrollments", { validationLevel: "strict" });
db.createCollection("broadcast_types", { validationLevel: "strict" });
db.createCollection("location_exams", { validationLevel: "strict" });

```

Apéndice 3.11. Configuraciones de servicio de base de datos para el módulo de autenticación.

docker-compose.yml

```
62  # auth services
63  auth_db:
64    ports:
65      - "4001:5432"
66    build: ./services/auth-module/auth-db
67    env_file: ./services/auth-module/auth-db/.env
68
```

Apéndice 3.12. Configuraciones de servicio de base de datos para el módulo de datos.

docker-compose.yml

```
128  # data services
129  data_db:
130    ports:
131      - "4002:5432"
132    build: ./services/data-module/data-db
133    env_file: ./services/data-module/data-db/.env
134
```

Apéndice 3.13. Configuraciones de servicio de base de datos para el módulo de datos.

docker-compose.yml

```
184  # enrollment services
185  enrollment_db:
186    ports:
187      - "4003:27017"
188    build: ./services/enrollment-module/enrollment-db
189    env_file: ./services/enrollment-module/enrollment-db/.env
190
```

Apéndice 3.14. Configuración de la imagen para el contenedor del API Gateway.

Dockerfile

services > api-gateway > Dockerfile > ...

```
# --- RELEASE PHASE ---
FROM nginx:stable-alpine
RUN rm /etc/nginx/conf.d/default.conf

COPY ./api /etc/nginx/api
#COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY default.conf /etc/nginx/nginx.conf
COPY api_backends.conf /etc/nginx/api_backends.conf

```

Apéndice 3.15. Configuración del servidor NGINX del API Gateway.

nginx.conf

services > api-gateway > nginx.conf

```
include api_backends.conf;

# buffering optimizations
client_body_buffer_size 10K;
client_header_buffer_size 1k;
client_max_body_size 8m;
large_client_header_buffers 2 1k;

# timeout optimization
client_body_timeout 32;
client_header_timeout 32;
send_timeout 32;

# log services
access_log on;

# keep alive
keepalive_requests 5000;

server {
    listen 8000;
    include api/*.conf;
    keepalive_timeout  32;
    # Error responses
    proxy_intercept_errors on;
    default_type application/json;
}
```

Apéndice 3.16. Configuración de clústeres de servicios transparentados por el API Gateway.

api_backends.conf ●

services > api-gateway > api_backends.conf

```
upstream auth_services {
    server auth_service_1:5001;
    server auth_service_2:5002;
    server auth_service_3:5003;
    server auth_service_4:5004;
    server auth_service_5:5005;
    server auth_service_6:5006;
    server auth_service_7:5007;
    server auth_service_8:5008;
}

upstream data_services {
    server data_service_1:5001;
    server data_service_2:5002;
    server data_service_3:5003;
    server data_service_4:5004;
    server data_service_5:5005;
    server data_service_6:5006;
    server data_service_7:5007;
    server data_service_8:5008;
}

upstream enrollment_services {
    server enrollment_service_1:5001;
    server enrollment_service_2:5002;
    server enrollment_service_3:5003;
    server enrollment_service_4:5004;
    server enrollment_service_5:5005;
    server enrollment_service_6:5006;
    server enrollment_service_7:5007;
    server enrollment_service_8:5008;
}

upstream report_services {
    server report_service_1:5001;
    server report_service_2:5002;
    server report_service_3:5003;
    server report_service_4:5004;
    server report_service_5:5005;
    server report_service_6:5006;
    server report_service_7:5007;
    server report_service_8:5008;
}
```

Apéndice 3.17. Configuración de proxy del API Gateway del servicio de autenticación.

auth_service_api.conf

services > api-gateway > api > auth_service_api.conf

```
# auth service API

location /api/auth-service/ {
    location /api/auth-service/status {
        proxy_pass http://auth_services/status;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/auth-service/login {
        proxy_pass http://auth_services/login;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/auth-service/recovery-password {
        proxy_pass http://auth_services/recovery-password;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/auth-service/update-password {
        proxy_pass http://auth_services/update-password;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    # comunication between services
    location /api/auth-service/create-user {
        proxy_pass http://auth_services/create-user;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    return 404;
}
```

Apéndice 3.18. Configuración de proxy del API Gateway del servicio de datos.

data_service_api.conf

services > api-gateway > api > data_service_api.conf

```
# data service API

location /api/data-service/ {
    location /api/data-service/status {
        proxy_pass http://data_services/status;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/data-service/register {
        proxy_pass http://data_services/register;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location ~ ^/api/data-service/data/?(.*) {
        proxy_pass http://data_services/data/$1;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/data-service/countries {
        proxy_pass http://data_services/countries;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location ~ ^/api/data-service/states/?(.*) {
        proxy_pass http://data_services/states/$1;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location ~ ^/api/data-service/cities/?(.*) {
        proxy_pass http://data_services/cities/$1;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    return 404;
}

```

Apéndice 3.19. Configuración de proxy del API Gateway del servicio de matriculación.

enrollment_service_api.conf

services > api-gateway > api > enrollment_service_api.conf

```
# enrollment service API

location /api/enrollment-service/ {
    location /api/enrollment-service/status {
        proxy_pass http://enrollment_services/status;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/programs {
        proxy_pass http://enrollment_services/programs;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/periods {
        proxy_pass http://enrollment_services/periods;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/course-types {
        proxy_pass http://enrollment_services/course-types;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/sections {
        proxy_pass http://enrollment_services/sections;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/levels {
        proxy_pass http://enrollment_services/levels;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/courses {
        proxy_pass http://enrollment_services/courses;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/broadcast-types {
        proxy_pass http://enrollment_services/broadcast-types;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/enrollment {
        proxy_pass http://enrollment_services/enrollment;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/enrollment-service/location-exam {
        proxy_pass http://enrollment_services/location-exam;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }

    return 404;
}
```

Apéndice 3.20. Configuración de proxy del API Gateway del servicio de reportes.

report_service_api.conf

services > api-gateway > api > report_service_api.conf

```
# report service API

location /api/report-service/ {
    location /api/report-service/status {
        proxy_pass http://report_services/status;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/report-service/enrollment {
        proxy_pass http://report_services/enrollment;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location /api/report-service/location-exam {
        proxy_pass http://report_services/location-exam;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    location ~ ^/api/report-service/enrollment-sheet/?(.*) {
        proxy_pass http://report_services/enrollment-sheet/$1;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
    return 404;
}
```

Apéndice 3.18. Configuración de servicio para el API Gateway.

docker-compose.yml

docker-compose.yml

```
# api Gateway
api_gateway:
  build: ./services/api-gateway
  ports:
    - "8000:8000"
  depends_on:
    - auth_db
    - auth_service_1
    - auth_service_2
    - auth_service_3
    #
    - data_db
    - data_service_1
    #
    - enrollment_db
    - enrollment_service_1
    - enrollment_service_2
    - enrollment_service_3
    #
    - report_service_1

```

Apéndice 4 – Interfaces del prototipo web

Apéndice 4.1. Interfaz de inicio de sesión.

Apéndice 4.2. Interfaz de registro del estudiante.

Registrarte

Tipo de documento
CEDULA

Número de documento

Nombres

Apellidos

Estado civil
CASADO

Sexo
MASCULINO

Tipo de discapacidad
AUDITIVA

Porcentaje de discapacidad

Fecha de nacimiento
dd/mm/aaaa

País
ECUADOR

Provincia
EL ORO

Ciudad
MACHALA

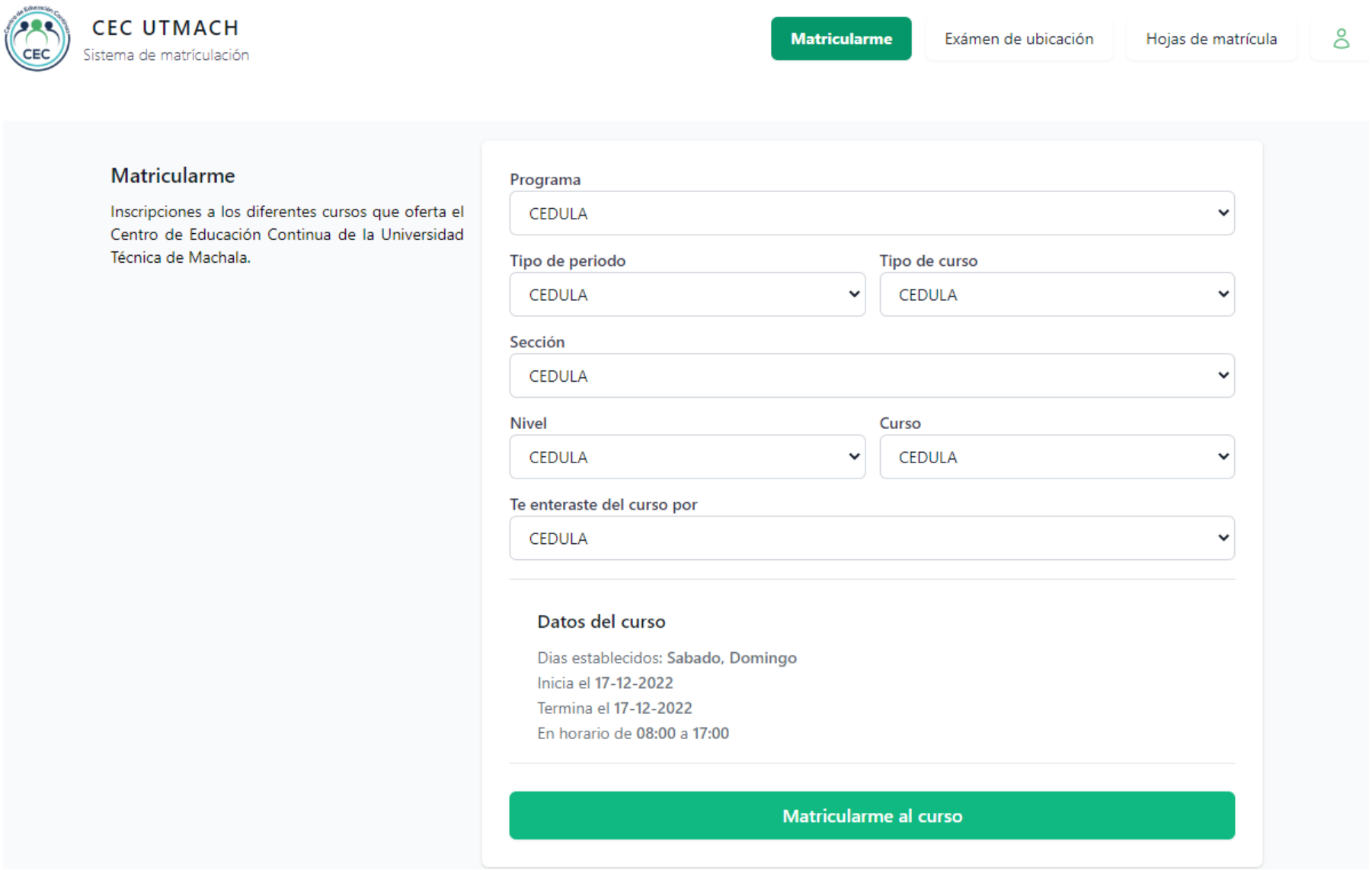
Dirección

Teléfono fijo

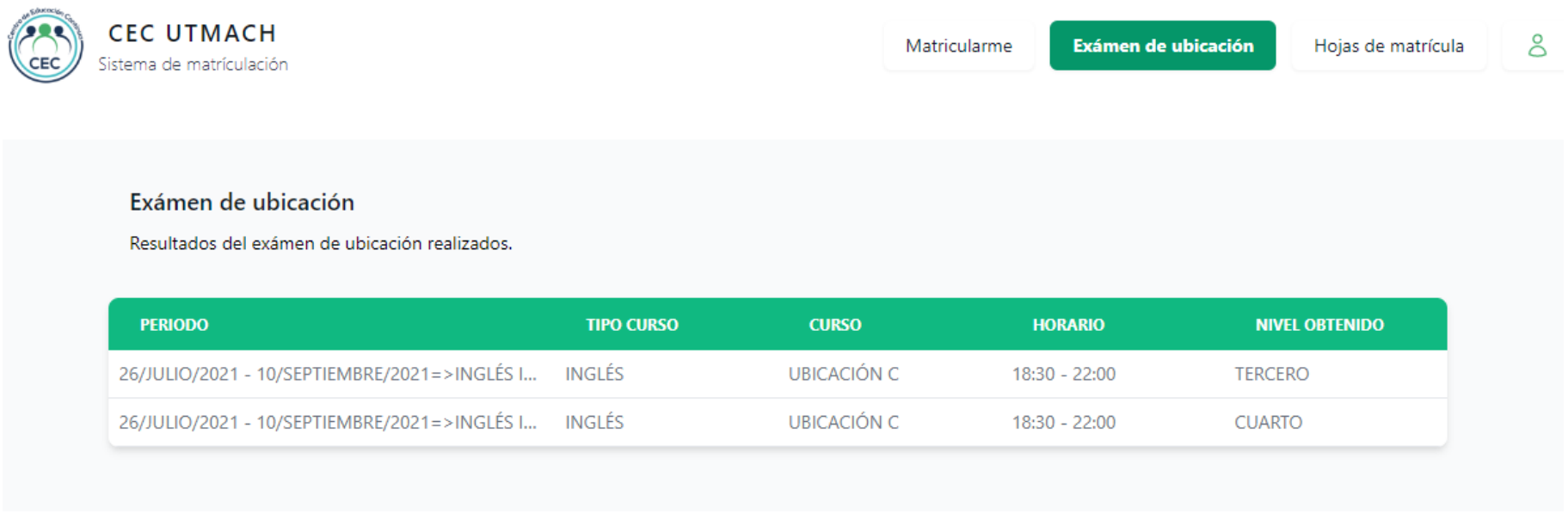
Teléfono móvil

Email

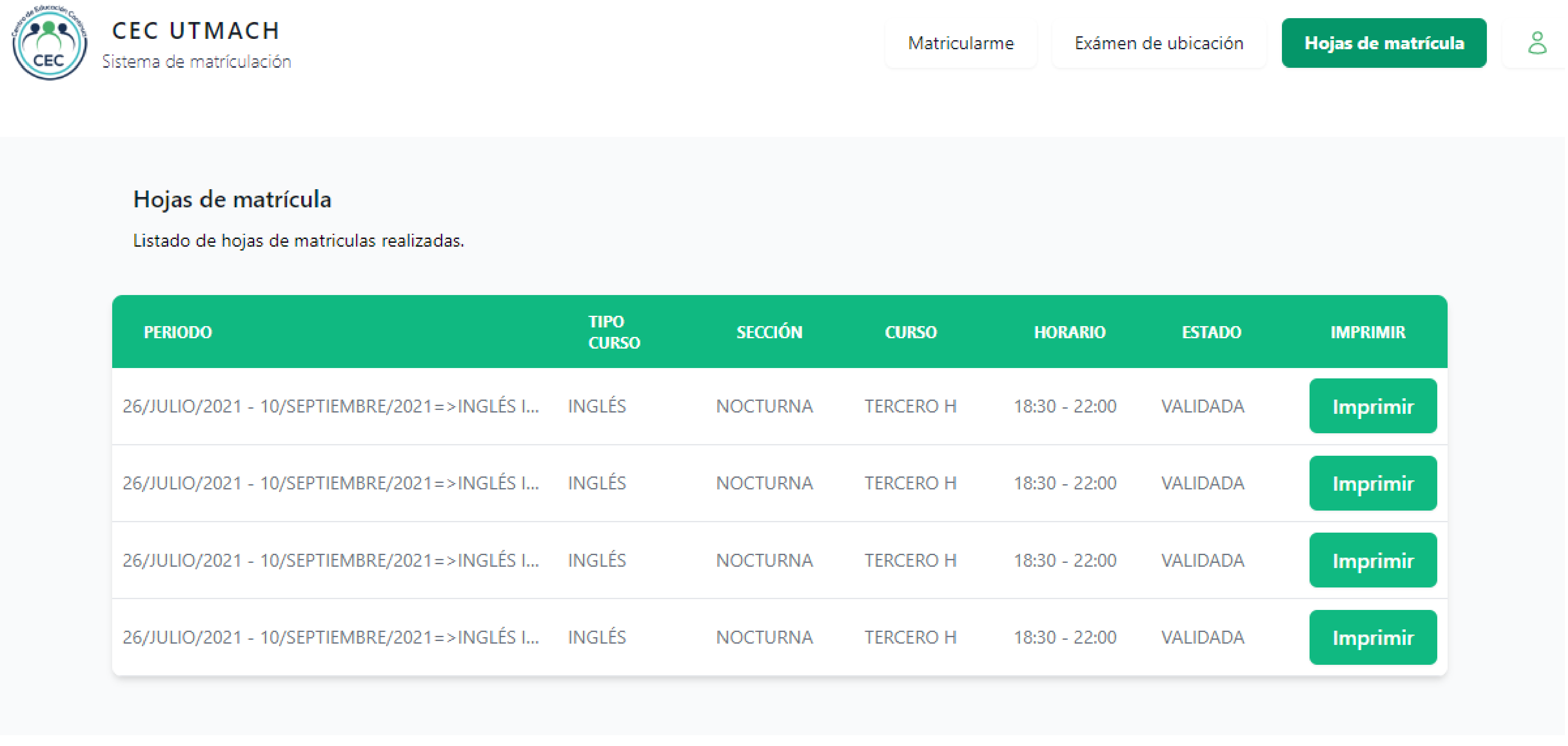
Apéndice 4.3. Interfaz de matriculación del estudiante.



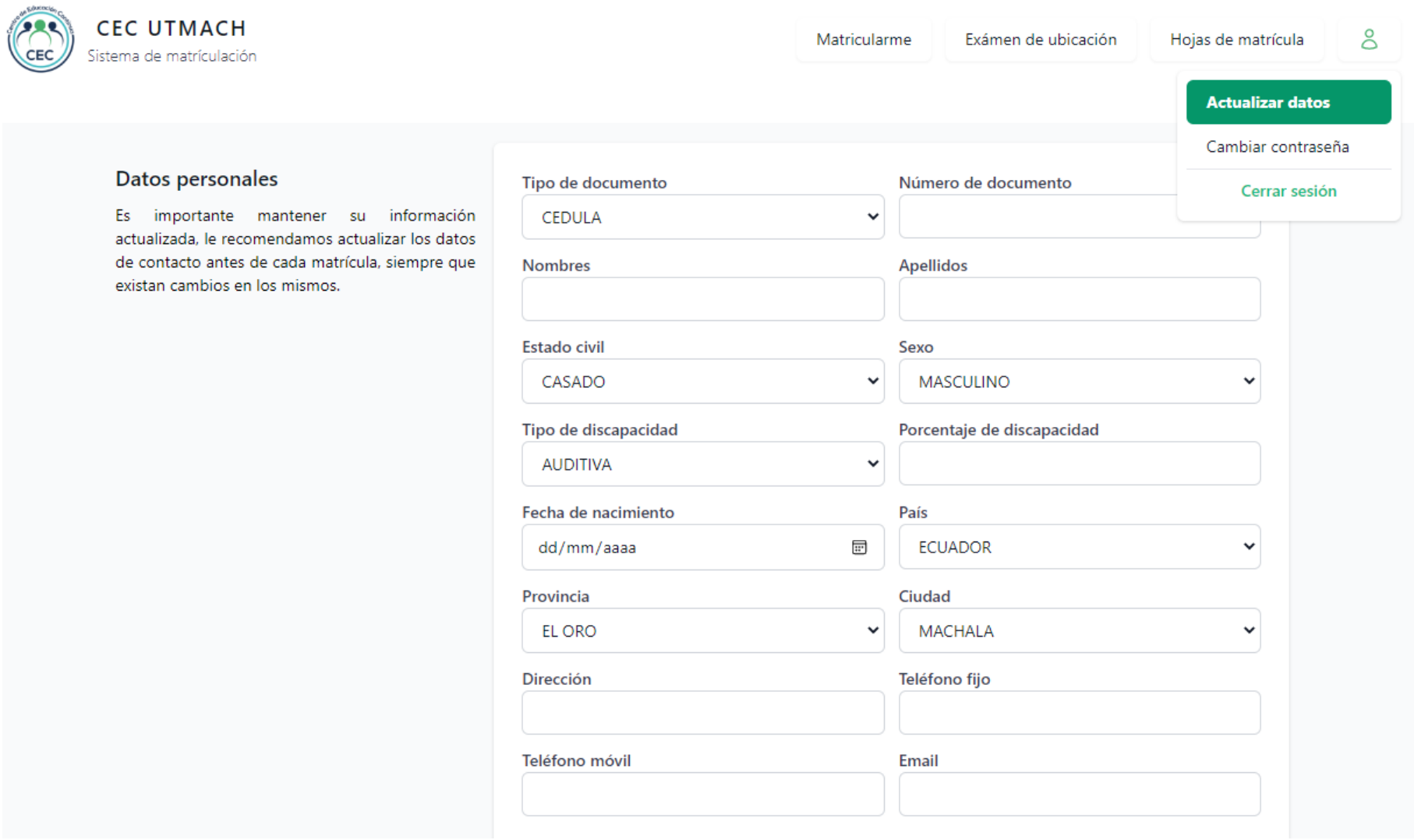
Apéndice 4.4. Interfaz de consulta de resultados de exámenes de ubicación.



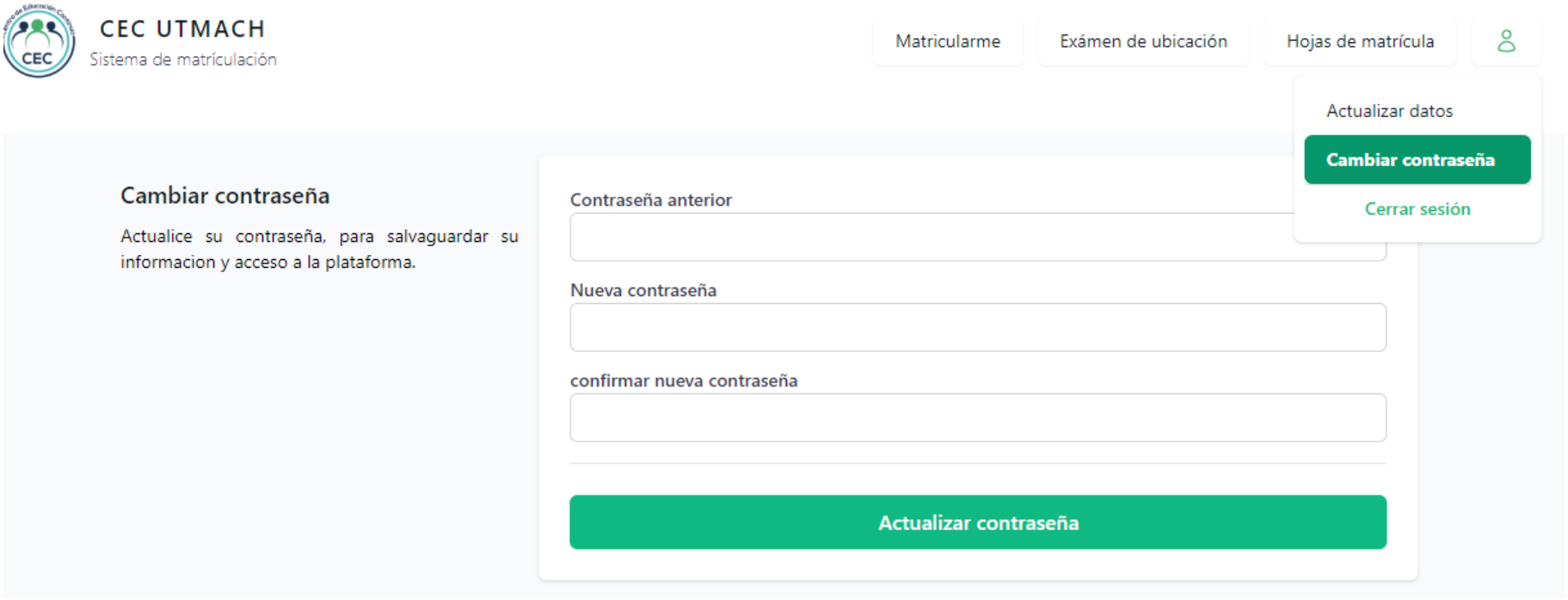
Apéndice 4.5. Interfaz de consulta de hojas de matrícula.



Apéndice 4.6. Interfaz de consulta de edición de datos personales.



Apéndice 4.7. Interfaz de consulta de cambio de contraseña.



Apéndice 5 - Código fuente de los métodos de los servicios

Apéndice 5.1. Método de iniciar sesión.

```
  async run(req: Request, res: Response) {
    try {
      const data = await this.dataRepository.login(req.body);
      const id_usuario = data.id_usuario;
      const usuario = data.usuario;
      const nombres = data.nombres;
      const ultimo_acceso = await this.getCurrentDate();
      jwt.sign({ usuario: usuario }, 'secretpassw', { expiresIn:
      '1h' },
      (error: any, token: any) => {
        const data = {
          id_usuario,
          nombres,
          ultimo_acceso,
          ip: ip.address(),
          token: token
        };
        res.status(httpStatus.OK).send(data);
      });
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
```

Apéndice 5.2. Método de recuperar contraseña.

```
  async run(req: Request, res: Response) {
    try {
      const data = req.body;
      await this.dataRepository.recoveryPassword(data);
       const message = 'Se ha enviado su nueva contraseña al
       correo electrónico.';
      res.status(httpStatus.ACCEPTED).send(message);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
}
```

Apéndice 5.3. Método de actualizar contraseña.

```
async run(req: Request, res: Response) {
    try {
      const { id_usuario, contrasena_actual, contrasena_nueva }
      = req.body;
      const userID = parseInt(id_usuario);
      if (!(await this.dataRepository.verifyUserID(userID))) {
        throw new Error('Usuario especificado no encontrado.');
      }
      if (await this.dataRepository.checkCurrentPassword(userID,
      contrasena_actual)) {
        await this.dataRepository.updatePassword(userID,
        contrasena_nueva);
        const message = 'Contraseña actualizada con éxito.';
        res.status(httpStatus.ACCEPTED).send(message);
      }
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
```

Apéndice 5.4. Método de registrarse.

```
  async run(req: Request, res: Response) {
    try {
      await this.dataRepository.createUser(req.body);
      const message = 'Creación correcta de usuario!';
      res.status(httpStatus.CREATED).send(message);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
```

Apéndice 5.5. Método de registrar datos personales.

```
async run(req: Request, res: Response) {
    try {
      const data = req.body;
      await this.dataRepository.register(data);
      const message = 'Registrado con éxito.';
      res.status(httpStatus.CREATED).send(message);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
```

Apéndice 5.6. Método de visualizar datos personales.

```
async run(req: Request, res: Response) {
    try {
      const { userID } = req.params;
      const data = await
this.dataRepository.getData(parseInt(userID));
      res.status(httpStatus.OK).send(data);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 5.7. Método de editar datos personales.

```
async run(req: Request, res: Response) {
    try {
      const { userID } = req.params;
      const data = req.body;
      await this.dataRepository.updateData(parseInt(userID),
      data);
      const message = 'Datos actualizados con éxito.';
      res.status(httpStatus.ACCEPTED).send(message);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
```

Apéndice 5.8. Método de visualizar país.

```
async run(req: Request, res: Response) {
    try {
      const data = await this.dataRepository.getCountries();
      res.status(httpStatus.OK).send(data);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 5.9. Método de visualizar provincia de un país.

```
async run(req: Request, res: Response) {
    try {
      const { countryID } = req.params;
      const data = await
      this.dataRepository.getStates(parseInt(countryID));
      res.status(httpStatus.OK).send(data);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 5.10. Método de visualizar ciudades de una provincia.

```
async run(req: Request, res: Response) {
    try {
      const { stateID } = req.params;
      const data = await
      this.dataRepository.getCities(parseInt(stateID));
      res.status(httpStatus.OK).send(data);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 5.11. Método de visualizar programas ofrecidos.

```
async getPrograms(req: Request, res: Response) {
    try {
      const projection = { _id: false };
      const query = await this.mongoRepository.getFilter({}, {
      projection });
      const programs = await query.toArray();
      if (programs.length == 0) throw new Error('No se
      encontraron datos de los programas.');
      res.status(httpStatus.OK).json(programs);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 1.12. Método de visualizar los tipos de periodos.

```
async getPeriodsByProgram(req: Request, res: Response) {
    try {
      const programID = String(req.query.programID);
      if (req.query.programID === undefined || !programID) throw
      new Error('ID del programa no especificado!');
      const filters = {
        'programa.id': { $eq: parseInt(programID) }
      };
      const projection = { _id: false, programa: false };
      const query = await this.getPeriods(filters, projection);
      const periods = await query.toArray();
      if (periods.length == 0) throw new Error('No se
      encontraron datos de los períodos con el programa;
       especificado.')
       res.status(httpStatus.OK).json(periods);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 1.13. Método de visualizar los tipos de cursos.

```
async getCoursesTypesByPeriod(req: Request, res: Response) {
  try {
    const periodID = String(req.query.periodID);
    if (req.query.periodID === undefined || !periodID) throw
    new Error('ID del período no especificado!');
    const filters = {
      'periodo.id': { $eq: parseInt(periodID) }
    };
    const projection = { _id: false, periodo: false };
    const query = await this.getCoursesTypes(filters,
    projection);
    const courses = await query.toArray();
    if (courses.length == 0)
      throw new Error('No se encontraron datos de los tipos de
      cursos con el período especificado.');
      res.status(httpStatus.OK).json(courses);
  } catch (error) {
    if (error instanceof Error) {
      const message = error.message;
      res.status(httpStatus.NOT_FOUND).send(message);
    }
  }
}
```

Apéndice 1.14. Método de visualizar las secciones.

```
async getSectionsByCourseType(req: Request, res: Response) {
  try {
    const courseTypeID = String(req.query.courseTypeID);
    if (req.query.courseTypeID === undefined || !courseTypeID)
      throw new Error('ID del tipo de curso no
      especificado.');
      const filters = {
      'tipo_curso.id': { $eq: parseInt(courseTypeID) }
    };
    const projection = { _id: false, tipo_curso: false };
    const query = await this.getSections(filters, projection);
    const sections = await query.toArray();
    if (sections.length == 0)
      throw new Error('No se encontraron datos de las
      secciones con el tipo de curso especificado.');
      res.status(httpStatus.OK).json(sections);
  } catch (error) {
    if (error instanceof Error) {
      const message = error.message;
      res.status(httpStatus.NOT_FOUND).send(message);
    }
  }
}
```

Apéndice 1.15. Método de visualizar los niveles.

```
async getLevelsBySection(req: Request, res: Response) {
    try {
      const sectionID = String(req.query.sectionID);
      if (req.query.sectionID === undefined || !sectionID) throw
       new Error('ID de la sección no especificada.');
      const filters = {
        'seccion.id': { $eq: parseInt(sectionID) }
      };
      const projection = { _id: false, seccion: false };
      const query = await this.getLevels(filters, projection);
      const levels = await query.toArray();
      if (levels.length == 0) throw new Error('No se encontraron
       datos de los niveles con la sección especificada.');
      res.status(httpStatus.OK).json(levels);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 1.16. Método de difusión (te enteraste del curso por).

```
  async getBroadcastTypes(req: Request, res: Response) {
    try {
      const projection = { _id: false };
      const query = await this.mongoRepository.getFilter({}, {
      projection });
      const broadcastTypes = await query.toArray();
      if (broadcastTypes.length == 0) throw new Error('No se
       encontraron datos de los tipos de difunsión.');
      res.status(httpStatus.OK).json(broadcastTypes);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
```

Apéndice 1.17. Método de visualizar los cursos.

```
async getCoursesByLevel(req: Request, res: Response) {
    try {
      const levelID = String(req.query.levelID);
      if (req.query.levelID === undefined || !levelID) throw new
      Error('ID del nivel no especificado.');
      const filters = {
        'nivel.id': { $eq: parseInt(levelID) }
      };
      const projection = {
        _id: false,
        informacion_curso: false,
        programa: false,
        tipo_curso: false,
        nivel: false,
        seccion: false
      };
      const query = await this.getCourses(filters, projection);
      const courses = await query.toArray();
      if (courses.length == 0) throw new Error('No se
       encontraron datos de los cursos con el nivel
       especificado.');
       res.status(httpStatus.OK).json(courses);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 1.18. Método de matricularse.

```
async enroll(req: Request, res: Response) {
    try {
      const { curso, usuario } = req.body;
      if (!curso) throw new Error('ID del curso no
      especificado.');
       if (!usuario) throw new Error('ID del estudiante no
      especificado.');
       const courseID = parseInt(curso);
      const userID = parseInt(usuario);
      const course = await this.getCourse(courseID);
      const user = await this.getUser(userID);
      if (course.ubicacion) {
        await this.locationExamEnrollment.enroll(course, user);
      } else {
        await this.regularEnrollment.enroll(course, user);
      }
      res.status(httpStatus.CREATED).send('Matriculado con
       éxito.');
     } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.BAD_REQUEST).send(message);
      }
    }
  }
```

Apéndice 1.19. Método de visualizar los datos de la matricula.

```
  async getRegularEnrollmentsById(req: Request, res: Response) {
    try {
      const { enrollmentID } = req.params;
      if (!enrollmentID) throw new Error('ID de la matriculada
      no especificada.');
      const enrollment = await
      this.regularEnrollment.getEnrollmentsById(
      parseInt(enrollmentID));
      res.status(httpStatus.OK).json(enrollment);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 1.20. Método de resultados de examen de ubicación.

```
async getLocationExamsEnrollmentByUser(req: Request, res:
Response) {
    try {
      const userID = String(req.query.userID);
      if (req.query.userID === undefined || !userID) throw new
      Error('ID del usuario no especificado.');
      const locationExams = await
      this.locationExamEnrollment.getLocationExamsByUser(
      parseInt(userID));
      res.status(httpStatus.OK).json(locationExams);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 1.21. Método de visualizar lista de matrículas.

```
async getRegularEnrollmentsByUser(req: Request, res: Response) {
    try {
      const userID = String(req.query.userID);
      if (req.query.userID === undefined || !userID) throw new
      Error('ID del estudiante no especificado.');
      const enrollments = await
      this.regularEnrollment.getEnrollmentsByUser(
      parseInt(userID));
      res.status(httpStatus.OK).json(enrollments);
    } catch (error) {
      if (error instanceof Error) {
        const message = error.message;
        res.status(httpStatus.NOT_FOUND).send(message);
      }
    }
  }
```

Apéndice 6 - Resultados de las pruebas con la arquitectura de microservicios

Apéndice 6.1. Resultados completos de la prueba con la arquitectura de microservicios configurados 1000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-1000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	2909	3363	6	27898	3280.49	0.00%	12.2/sec	6.90	2.35	581.0
GET /enroll...	24679	4381	11	46428	3970.09	0.00%	102.1/sec	150.13	20.45	1505.0
GET /periods	2863	3424	5	27328	3376.48	0.00%	11.9/sec	6.35	2.43	547.0
GET /course-...	2813	3448	5	23137	3397.65	0.00%	11.7/sec	4.06	2.43	356.0
GET /locatio...	24185	4266	7	47473	3891.36	0.00%	100.5/sec	95.67	20.41	975.0
GET /sections	2751	3397	5	18384	3271.39	0.00%	11.5/sec	4.77	2.38	426.0
GET /levels	2715	3513	4	33988	3402.18	0.00%	11.3/sec	5.74	2.30	519.0
GET /courses	2663	3513	6	22636	3449.20	0.00%	11.1/sec	10.34	2.24	956.0
GET /courses...	2610	3662	4	19323	3415.90	0.00%	10.9/sec	10.34	2.12	973.0
GET /broadc...	2553	3588	5	24942	3337.85	0.00%	10.7/sec	7.46	2.14	715.0
POST /login	2660	23443	1808	61020	15596.51	0.00%	11.0/sec	6.38	2.96	593.7
POST /enroll...	2514	26655	184	95278	21714.60	1.03%	10.0/sec	3.35	2.51	342.3
POST /enroll...	2260	28004	85	112130	21506.79	0.75%	9.1/sec	3.04	2.28	341.9
GET /enroll...	1964	4876	6	35830	4161.21	0.00%	8.4/sec	11.99	1.65	1470.0
PUT /update...	2522	55186	1955	116267	23380.21	0.00%	10.0/sec	3.45	3.18	354.0
POST /register	3730	59123	4798	87089	17671.86	0.00%	14.6/sec	4.86	10.34	340.0
GET /data/{u...	2733	36	3	1262	90.37	0.00%	12.1/sec	8.62	2.25	728.0
PUT /data/{u...	2731	52	4	1629	103.77	0.00%	12.1/sec	4.13	8.54	349.0
GET /countries	2730	35	4	1383	96.03	0.00%	12.2/sec	97.20	2.29	8178.0
GET /states/{...	2730	35	3	1144	85.46	0.00%	12.2/sec	17.95	2.31	1507.0
GET /cities/{...	2730	157	18	1457	174.99	0.00%	12.2/sec	9.87	2.32	828.0
POST /recov...	1756	10757	3342	36734	6033.14	0.00%	7.4/sec	2.74	2.14	377.0
TOTAL	101801	8538	3	116267	16063.93	0.04%	399.7/sec	441.05	96.16	1130.0

Apéndice 6.2. Resultados de solicitudes atendidas de la prueba con la arquitectura de microservicios configurados 1000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-1000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	2909	3363	6	27898	3280.49	0.00%	12.2/sec	6.90	2.35	581.0
GET /enroll...	24679	4381	11	46428	3970.09	0.00%	102.1/sec	150.13	20.45	1505.0
GET /periods	2863	3424	5	27328	3376.48	0.00%	11.9/sec	6.35	2.43	547.0
GET /course-...	2813	3448	5	23137	3397.65	0.00%	11.7/sec	4.06	2.43	356.0
GET /locatio...	24185	4266	7	47473	3891.36	0.00%	100.5/sec	95.67	20.41	975.0
GET /sections	2751	3397	5	18384	3271.39	0.00%	11.5/sec	4.77	2.38	426.0
GET /levels	2715	3513	4	33988	3402.18	0.00%	11.3/sec	5.74	2.30	519.0
GET /courses	2663	3513	6	22636	3449.20	0.00%	11.1/sec	10.34	2.24	956.0
GET /courses...	2610	3662	4	19323	3415.90	0.00%	10.9/sec	10.34	2.12	973.0
GET /broadc...	2553	3588	5	24942	3337.85	0.00%	10.7/sec	7.46	2.14	715.0
POST /login	2660	23443	1808	61020	15596.51	0.00%	11.0/sec	6.38	2.96	593.7
POST /enroll...	2488	26412	184	95278	21625.23	0.00%	9.9/sec	3.30	2.48	341.0
POST /enroll...	2243	27818	85	112130	21386.87	0.00%	9.0/sec	3.01	2.27	341.0
GET /enroll...	1964	4876	6	35830	4161.21	0.00%	8.4/sec	11.99	1.65	1470.0
PUT /update...	2522	55186	1955	116267	23380.21	0.00%	10.0/sec	3.45	3.18	354.0
POST /register	3730	59123	4798	87089	17671.86	0.00%	14.6/sec	4.86	10.34	340.0
GET /data/{u...	2733	36	3	1262	90.37	0.00%	12.1/sec	8.62	2.25	728.0
PUT /data/{u...	2731	52	4	1629	103.77	0.00%	12.1/sec	4.13	8.54	349.0
GET /countries	2730	35	4	1383	96.03	0.00%	12.2/sec	97.20	2.29	8178.0
GET /states/{...	2730	35	3	1144	85.46	0.00%	12.2/sec	17.95	2.31	1507.0
GET /cities/{...	2730	157	18	1457	174.99	0.00%	12.2/sec	9.87	2.32	828.0
POST /recov...	1756	10757	3342	36734	6033.14	0.00%	7.4/sec	2.74	2.14	377.0
TOTAL	101758	8520	3	116267	16038.49	0.00%	399.5/sec	440.98	96.11	1130.3

Apéndice 6.3. Resultados completos de la prueba con la arquitectura de microservicios configurados 3000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-3000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	4089	7859	7	74680	10290.08	0.00%	14.7/sec	8.36	2.85	581.0
GET /periods	3898	7372	6	75139	9403.69	0.00%	14.0/sec	7.49	2.86	547.0
GET /enroll...	29808	11535	11	84919	12237.55	0.00%	107.2/sec	157.20	21.46	1502.0
GET /course-...	3771	7626	5	87961	9102.87	0.00%	13.6/sec	4.72	2.82	356.0
GET /locatio...	28278	11581	12	78863	12370.16	0.00%	101.8/sec	96.92	20.68	975.0
GET /sections	3655	7653	7	74793	8785.76	0.00%	13.2/sec	5.49	2.75	426.0
GET /levels	3561	8175	6	74789	9080.60	0.00%	12.9/sec	6.55	2.62	519.0
GET /courses	3483	8516	7	81310	8718.39	0.00%	12.7/sec	11.82	2.56	956.0
GET /courses...	3406	8986	14	75027	9071.88	0.00%	12.6/sec	11.96	2.45	973.0
GET /broadc...	3346	9473	6	74490	9491.51	0.00%	12.3/sec	8.58	2.46	715.0
POST /login	4113	138993	6265	217548	57645.78	0.00%	14.6/sec	8.43	3.94	590.7
POST /enroll...	3283	85096	284	269057	65665.40	0.00%	10.4/sec	3.47	2.60	341.0
POST /enroll...	2300	81291	545	250926	56790.73	0.00%	7.4/sec	2.45	1.85	341.0
GET /enroll...	1279	17096	19	77408	15946.19	0.00%	4.7/sec	6.79	0.93	1467.0
POST /register	4815	149018	39085	228635	57804.16	0.00%	16.9/sec	5.62	11.95	340.0
GET /data/{u...	1829	66	5	2706	180.15	0.00%	9.6/sec	6.81	1.78	728.0
PUT /data/{u...	1829	162	5	2469	290.04	0.00%	9.6/sec	3.26	6.73	349.0
GET /countries	1826	83	4	2358	222.64	0.00%	9.6/sec	76.38	1.80	8178.0
GET /states/{...	1826	76	3	2264	190.48	0.00%	9.6/sec	14.06	1.81	1507.0
GET /cities/{...	1821	345	20	2789	403.73	0.00%	9.5/sec	7.72	1.82	828.0
PUT /update...	2275	48089	8063	83144	16290.11	0.00%	8.2/sec	2.83	2.61	354.0
POST /recov...	1209	14357	3696	36184	7998.14	0.00%	5.9/sec	2.18	1.71	377.0
TOTAL	115700	24334	3	269057	45172.23	0.00%	361.4/sec	368.77	85.88	1045.0

Apéndice 6.4. Resultados de solicitudes atendidas de la prueba con la arquitectura de microservicios configurados 3000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-3000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	4089	7859	7	74680	10290.08	0.00%	14.7/sec	8.36	2.85	581.0
GET /periods	3898	7372	6	75139	9403.69	0.00%	14.0/sec	7.49	2.86	547.0
GET /enroll...	29808	11535	11	84919	12237.55	0.00%	107.2/sec	157.20	21.46	1502.0
GET /course-...	3771	7626	5	87961	9102.87	0.00%	13.6/sec	4.72	2.82	356.0
GET /locatio...	28278	11581	12	78863	12370.16	0.00%	101.8/sec	96.92	20.68	975.0
GET /sections	3655	7653	7	74793	8785.76	0.00%	13.2/sec	5.49	2.75	426.0
GET /levels	3561	8175	6	74789	9080.60	0.00%	12.9/sec	6.55	2.62	519.0
GET /courses	3483	8516	7	81310	8718.39	0.00%	12.7/sec	11.82	2.56	956.0
GET /courses...	3406	8986	14	75027	9071.88	0.00%	12.6/sec	11.96	2.45	973.0
GET /broadc...	3346	9473	6	74490	9491.51	0.00%	12.3/sec	8.58	2.46	715.0
POST /login	4113	138993	6265	217548	57645.78	0.00%	14.6/sec	8.43	3.94	590.7
POST /enroll...	3283	85096	284	269057	65665.40	0.00%	10.4/sec	3.47	2.60	341.0
POST /enroll...	2300	81291	545	250926	56790.73	0.00%	7.4/sec	2.45	1.85	341.0
GET /enroll...	1279	17096	19	77408	15946.19	0.00%	4.7/sec	6.79	0.93	1467.0
POST /register	4815	149018	39085	228635	57804.16	0.00%	16.9/sec	5.62	11.95	340.0
GET /data/{u...	1829	66	5	2706	180.15	0.00%	9.6/sec	6.81	1.78	728.0
PUT /data/{u...	1829	162	5	2469	290.04	0.00%	9.6/sec	3.26	6.73	349.0
GET /countries	1826	83	4	2358	222.64	0.00%	9.6/sec	76.38	1.80	8178.0
GET /states/{...	1826	76	3	2264	190.48	0.00%	9.6/sec	14.06	1.81	1507.0
GET /cities/{...	1821	345	20	2789	403.73	0.00%	9.5/sec	7.72	1.82	828.0
PUT /update...	2275	48089	8063	83144	16290.11	0.00%	8.2/sec	2.83	2.61	354.0
POST /recov...	1209	14357	3696	36184	7998.14	0.00%	5.9/sec	2.18	1.71	377.0
TOTAL	115700	24334	3	269057	45172.23	0.00%	361.4/sec	368.77	85.88	1045.0

Apéndice 6.5. Resultados completos de la prueba con la arquitectura de microservicios configurados 5000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-5000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	35854	16539	20	81491	12020.78	0.00%	119.2/sec	170.25	23.87	1462.0
GET /progra...	5322	9058	25	70849	8047.02	0.00%	17.8/sec	10.12	3.45	581.0
GET /periods	5218	8937	16	72021	7135.72	0.00%	17.6/sec	9.39	3.59	547.0
GET /locatio...	33335	16958	19	95276	12091.30	0.00%	103.2/sec	98.31	20.97	975.0
POST /login	5752	215600	742	270030	68364.96	0.00%	19.3/sec	11.20	5.21	593.9
GET /course-...	5157	9446	12	71841	6638.91	0.00%	17.3/sec	6.02	3.60	356.0
GET /sections	5095	9932	17	71261	6400.34	0.00%	17.1/sec	7.10	3.55	426.0
GET /levels	5048	10674	77	69549	5982.59	0.00%	16.9/sec	8.58	3.44	519.0
GET /courses	5019	11570	86	53323	5832.50	0.00%	18.7/sec	17.43	3.77	956.0
GET /courses...	5006	12670	336	68095	6055.80	0.00%	17.6/sec	16.74	3.42	973.0
GET /broadc...	5002	14477	254	51538	7180.28	0.00%	20.5/sec	14.35	4.11	715.0
POST /enroll...	4997	151449	2078	300181	67285.82	0.58%	12.2/sec	4.06	3.05	340.9
PUT /update...	874	39131	9654	84688	20434.76	0.00%	3.5/sec	1.22	1.12	354.0
POST /enroll...	1740	134826	28692	275466	56103.29	0.00%	4.3/sec	1.42	1.07	341.0
GET /enroll...	411	27317	6360	72049	13424.81	0.00%	1.6/sec	2.25	0.32	1427.0
POST /recov...	794	19560	4617	53143	10403.26	0.00%	3.4/sec	1.24	0.97	377.0
POST /register	5559	233392	53881	275465	48776.03	0.00%	18.7/sec	6.20	13.18	340.0
GET /data/{u...	559	88	5	1529	185.42	0.00%	3.6/sec	2.53	0.66	728.0
PUT /data/{u...	559	134	5	1361	206.97	0.00%	3.6/sec	1.21	2.51	349.0
GET /countries	559	93	5	1705	175.25	0.00%	3.6/sec	28.52	0.67	8178.0
GET /states/{...	559	89	4	830	145.40	0.00%	3.6/sec	5.27	0.68	1507.0
GET /cities/{...	559	261	21	1463	234.62	0.00%	3.6/sec	2.90	0.68	828.0
TOTAL	132978	39033	4	300181	68321.17	0.02%	320.4/sec	296.50	74.10	947.6

Apéndice 6.6. Resultados de solicitudes atendidas de la prueba con la arquitectura de microservicios configurados 5000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-5000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	35854	16539	20	81491	12020.78	0.00%	119.2/sec	170.25	23.87	1462.0
GET /progra...	5322	9058	25	70849	8047.02	0.00%	17.8/sec	10.12	3.45	581.0
GET /periods	5218	8937	16	72021	7135.72	0.00%	17.6/sec	9.39	3.59	547.0
GET /locatio...	33335	16958	19	95276	12091.30	0.00%	103.2/sec	98.31	20.97	975.0
POST /login	5752	215600	742	270030	68364.96	0.00%	19.3/sec	11.20	5.21	593.9
GET /course-...	5157	9446	12	71841	6638.91	0.00%	17.3/sec	6.02	3.60	356.0
GET /sections	5095	9932	17	71261	6400.34	0.00%	17.1/sec	7.10	3.55	426.0
GET /levels	5048	10674	77	69549	5982.59	0.00%	16.9/sec	8.58	3.44	519.0
GET /courses	5019	11570	86	53323	5832.50	0.00%	18.7/sec	17.43	3.77	956.0
GET /courses...	5006	12670	336	68095	6055.80	0.00%	17.6/sec	16.74	3.42	973.0
GET /broadc...	5002	14477	254	51538	7180.28	0.00%	20.5/sec	14.35	4.11	715.0
POST /enroll...	4968	150582	2078	299984	66514.34	0.00%	12.1/sec	4.04	3.03	341.0
PUT /update...	874	39131	9654	84688	20434.76	0.00%	3.5/sec	1.22	1.12	354.0
POST /enroll...	1740	134826	28692	275466	56103.29	0.00%	4.3/sec	1.42	1.07	341.0
GET /enroll...	411	27317	6360	72049	13424.81	0.00%	1.6/sec	2.25	0.32	1427.0
POST /recov...	794	19560	4617	53143	10403.26	0.00%	3.4/sec	1.24	0.97	377.0
POST /register	5559	233392	53881	275465	48776.03	0.00%	18.7/sec	6.20	13.18	340.0
GET /data/{u...	559	88	5	1529	185.42	0.00%	3.6/sec	2.53	0.66	728.0
PUT /data/{u...	559	134	5	1361	206.97	0.00%	3.6/sec	1.21	2.51	349.0
GET /countries	559	93	5	1705	175.25	0.00%	3.6/sec	28.52	0.67	8178.0
GET /states/{...	559	89	4	830	145.40	0.00%	3.6/sec	5.27	0.68	1507.0
GET /cities/{...	559	261	21	1463	234.62	0.00%	3.6/sec	2.90	0.68	828.0
TOTAL	132949	38976	4	299984	68219.76	0.00%	320.3/sec	296.48	74.08	947.8

Apéndice 6.7. Resultados completos de la prueba con la arquitectura de microservicios configurados 8000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-8000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	8611	17350	31	66772	10950.34	13.00%	32.8/sec	25.63	5.51	801.3
GET /periods	8348	16298	78	61288	9636.10	4.91%	31.6/sec	19.72	6.13	638.9
GET /enroll...	34618	26188	256	92637	14025.17	34.14%	123.1/sec	214.53	16.23	1784.7
POST /login	8716	258188	665	317678	96397.24	83.39%	24.0/sec	13.71	5.83	585.0
GET /course-...	8195	18650	314	68892	8799.06	2.87%	31.3/sec	12.53	6.32	410.2
GET /locatio...	31279	27286	321	102555	12446.77	15.11%	110.3/sec	129.43	19.02	1201.6
GET /sections	8083	20476	344	75463	9202.94	25.20%	31.1/sec	29.64	4.84	975.3
GET /levels	8020	23896	431	75562	11481.30	26.11%	31.5/sec	31.39	4.73	1019.5
GET /courses	7997	26030	614	72478	10859.64	18.09%	31.3/sec	37.37	5.18	1224.2
GET /courses...	7810	27987	1205	76473	11542.96	22.52%	30.0/sec	38.28	4.52	1306.2
GET /broadc...	6725	29328	1826	75227	12316.94	29.12%	26.2/sec	31.36	3.72	1224.5
PUT /update...	1366	58202	6148	145207	42501.24	47.51%	5.1/sec	6.98	0.85	1399.5
POST /register	8926	255145	5252	317696	102096.20	89.05%	24.6/sec	15.83	14.74	659.6
POST /enroll...	4695	70596	6123	189679	43461.51	30.48%	15.0/sec	14.18	2.61	967.6
GET /data/{u...	1352	19207	401	50206	5202.25	68.71%	7.2/sec	14.14	0.42	2005.6
POST /recov...	1083	49930	6859	108248	22705.43	28.07%	4.4/sec	4.05	0.92	934.8
POST /enroll...	1932	73549	5625	176069	42906.86	28.67%	7.4/sec	6.65	1.32	922.5
PUT /data/{u...	1330	18603	562	59605	9127.10	34.06%	7.8/sec	7.37	3.63	967.3
GET /countries	1274	16879	998	40549	7184.14	29.51%	8.0/sec	51.07	1.06	6558.7
GET /states/{...	1178	15907	961	64824	8229.12	6.54%	7.4/sec	11.43	1.32	1572.4
GET /enroll...	822	28128	6148	72715	10881.59	22.63%	4.2/sec	6.96	0.65	1680.7
GET /cities/{...	1035	17824	1129	62076	8348.24	7.83%	6.8/sec	6.19	1.19	932.2
TOTAL	163395	51696	31	317696	80466.60	29.29%	449.7/sec	526.94	85.51	1199.8

Apéndice 6.8. Resultados completos de la prueba con la arquitectura de microservicios configurados 8000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\microservice-sep-service-summary-8000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	7492	16827	31	66772	11486.51	0.00%	28.5/sec	16.17	5.51	581.0
GET /periods	7938	16203	78	61288	9799.53	0.00%	30.1/sec	16.05	6.13	547.0
GET /enroll...	22800	28099	256	92637	16368.36	0.00%	81.1/sec	115.83	16.23	1463.0
POST /login	1448	198488	665	317678	95144.09	0.00%	4.0/sec	2.30	1.07	590.8
GET /course-...	7960	18535	314	68892	8839.74	0.00%	30.4/sec	10.56	6.32	356.0
GET /locatio...	26552	28233	321	102555	12977.34	0.00%	93.6/sec	89.15	19.02	975.0
GET /sections	6046	20049	344	75463	10332.66	0.00%	23.3/sec	9.68	4.84	426.0
GET /levels	5926	23961	431	75562	12539.23	0.00%	23.3/sec	11.81	4.73	519.0
GET /courses	6550	26323	614	72478	11483.42	0.00%	25.6/sec	23.90	5.18	956.0
GET /courses...	6051	29236	1205	76473	12383.88	0.00%	23.3/sec	22.09	4.52	973.0
GET /broadc...	4767	32248	1826	75227	13068.29	0.00%	18.6/sec	12.98	3.72	715.0
PUT /update...	717	92218	20707	145207	31383.00	0.00%	2.7/sec	0.93	0.85	354.0
POST /enroll...	3264	91862	17359	189679	34844.88	0.00%	10.4/sec	3.47	2.61	341.0
GET /data/{u...	423	16079	401	50206	6568.75	0.00%	2.3/sec	1.65	0.43	728.0
POST /recov...	779	60199	22341	108248	17679.53	0.00%	3.2/sec	1.18	0.92	377.0
PUT /data/{u...	877	17467	562	59605	10472.87	0.00%	5.1/sec	1.75	3.63	349.0
GET /countries	898	15108	998	40549	7850.09	0.00%	5.6/sec	44.88	1.06	8178.0
POST /register	977	251018	91352	317696	54439.60	0.00%	2.7/sec	0.89	1.90	340.0
GET /states/{...	1101	15713	961	64824	8356.64	0.00%	7.0/sec	10.24	1.32	1507.0
GET /enroll...	636	29716	7692	72715	11078.65	0.00%	3.3/sec	4.76	0.65	1476.0
POST /enroll...	1378	94775	22443	176069	31520.20	0.00%	5.3/sec	1.75	1.32	341.0
GET /cities/{...	954	17010	1129	62076	7940.69	0.00%	6.3/sec	5.06	1.19	828.0
TOTAL	115534	32224	31	317696	36696.67	0.00%	318.0/sec	286.66	67.92	923.1

Apéndice 7 - Resultados de las pruebas con la arquitectura monolítica.

Apéndice 7.1. Resultados completos de la prueba con la arquitectura monolítica configurados 1000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-1000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	6913	16564	86	27286	6710.30	0.00%	27.2/sec	39.98	4.95	1503.0
GET /progra...	1025	13187	239	23826	7476.68	0.00%	4.4/sec	2.52	0.76	582.0
GET /locatio...	6375	16371	128	26752	6646.26	0.00%	25.4/sec	24.25	4.70	976.0
GET /periods	1019	15823	127	26263	9199.19	0.00%	4.1/sec	2.20	0.75	548.0
GET /course-...	1005	16722	175	27214	8802.93	0.00%	4.0/sec	1.41	0.75	357.0
GET /sections	1000	17794	473	26756	8343.28	0.00%	7.5/sec	3.14	1.40	427.0
GET /levels	1000	14910	1158	24022	7146.77	0.00%	6.8/sec	3.47	1.24	520.0
GET /courses	1000	14291	1137	22531	5964.07	0.00%	6.1/sec	5.73	1.10	957.0
GET /courses...	1000	15073	1230	22030	5466.55	0.00%	5.6/sec	5.32	0.96	974.0
GET /broadc...	1000	15237	500	19494	4955.30	0.00%	5.1/sec	3.56	0.91	716.0
POST /login	1520	127059	7454	185809	49686.96	0.00%	4.9/sec	2.82	1.23	594.0
PUT /update...	1080	82521	2185	151193	55384.62	0.00%	3.1/sec	1.07	0.93	355.0
POST /recov...	530	3860	627	10197	2626.67	2.26%	2.5/sec	0.94	0.69	377.5
POST /enroll...	1000	115277	30814	139396	15502.56	0.00%	3.4/sec	1.13	0.77	342.0
POST /register	1200	246443	39889	321866	81760.94	0.17%	3.3/sec	1.09	2.25	341.0
GET /data/{u...	270	6981	1996	10685	2643.38	0.00%	1.4/sec	1.00	0.24	729.0
PUT /data/{u...	260	5865	1297	10176	2012.26	2.31%	1.4/sec	0.48	0.96	350.1
GET /countries	238	7938	931	13418	3057.06	6.72%	1.3/sec	9.81	0.23	7652.8
GET /states/{...	200	6916	901	12889	3338.46	0.00%	1.7/sec	2.43	0.29	1508.0
GET /cities/{...	200	5448	523	10092	2274.24	0.00%	1.8/sec	1.42	0.30	829.0
POST /enroll...	200	120151	114894	133711	4303.56	0.00%	45.7/min	0.25	0.17	342.0
GET /enroll...	85	21769	15474	25124	2645.47	0.00%	1.0/sec	1.48	0.18	1468.0
TOTAL	28120	38107	86	321866	60258.72	0.13%	76.6/sec	71.07	16.82	950.5

Apéndice 7.2. Resultados de solicitudes atendidas de la prueba con la arquitectura monolítica configurados 1000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-1000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	6913	16564	86	27286	6710.30	0.00%	27.2/sec	39.98	4.95	1503.0
GET /progra...	1025	13187	239	23826	7476.68	0.00%	4.4/sec	2.52	0.76	582.0
GET /locatio...	6375	16371	128	26752	6646.26	0.00%	25.4/sec	24.25	4.70	976.0
GET /periods	1019	15823	127	26263	9199.19	0.00%	4.1/sec	2.20	0.75	548.0
GET /course-...	1005	16722	175	27214	8802.93	0.00%	4.0/sec	1.41	0.75	357.0
GET /sections	1000	17794	473	26756	8343.28	0.00%	7.5/sec	3.14	1.40	427.0
GET /levels	1000	14910	1158	24022	7146.77	0.00%	6.8/sec	3.47	1.24	520.0
GET /courses	1000	14291	1137	22531	5964.07	0.00%	6.1/sec	5.73	1.10	957.0
GET /courses...	1000	15073	1230	22030	5466.55	0.00%	5.6/sec	5.32	0.96	974.0
GET /broadc...	1000	15237	500	19494	4955.30	0.00%	5.1/sec	3.56	0.91	716.0
POST /login	1520	127059	7454	185809	49686.96	0.00%	4.9/sec	2.82	1.23	594.0
PUT /update...	1080	82521	2185	151193	55384.62	0.00%	3.1/sec	1.07	0.93	355.0
POST /recov...	518	3848	627	10197	2648.75	0.00%	2.5/sec	0.92	0.67	378.0
POST /enroll...	1000	115277	30814	139396	15502.56	0.00%	3.4/sec	1.13	0.77	342.0
POST /register	1198	246701	39889	321866	81584.66	0.00%	3.3/sec	1.09	2.25	341.0
GET /data/{u...	270	6981	1996	10685	2643.38	0.00%	1.4/sec	1.00	0.24	729.0
PUT /data/{u...	254	5910	1297	10176	2010.15	0.00%	1.4/sec	0.47	0.94	350.0
GET /countries	222	8236	931	13418	2934.33	0.00%	1.2/sec	9.78	0.21	8179.0
GET /states/{...	200	6916	901	12889	3338.46	0.00%	1.7/sec	2.43	0.29	1508.0
GET /cities/{...	200	5448	523	10092	2274.24	0.00%	1.8/sec	1.42	0.30	829.0
POST /enroll...	200	120151	114894	133711	4303.56	0.00%	45.7/min	0.25	0.17	342.0
GET /enroll...	85	21769	15474	25124	2645.47	0.00%	1.0/sec	1.48	0.18	1468.0
TOTAL	28084	38145	86	321866	60283.94	0.00%	76.5/sec	71.03	16.79	951.2

Apéndice 7.3. Resultados completos de la prueba con la arquitectura monolítica configurados 3000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-3000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	6473	57467	74	126072	22435.61	2.49%	24.6/sec	36.40	4.35	1518.4
GET /progra...	3000	66404	155	111401	23395.45	0.30%	20.5/sec	11.72	3.49	586.5
GET /locatio...	5007	59818	658	103949	18560.70	0.00%	19.0/sec	18.10	3.51	976.0
GET /periods	3000	71412	611	99076	11868.05	0.00%	14.8/sec	7.94	2.70	548.0
POST /login	3490	239351	14696	293412	70539.77	0.29%	10.9/sec	6.32	2.74	595.3
GET /course-...	3000	54328	18091	96711	10454.07	0.00%	12.3/sec	4.30	2.29	357.0
PUT /update...	490	15726	1584	29788	7550.07	0.00%	2.4/sec	0.83	0.72	355.0
POST /recov...	490	3203	722	9728	1484.81	0.00%	2.5/sec	0.91	0.67	378.0
POST /register	3109	360367	16752	429352	62770.85	0.35%	6.8/sec	2.31	4.69	346.6
GET /data/{u...	109	4318	1609	12659	2480.69	0.00%	56.6/min	0.67	0.16	729.0
PUT /data/{u...	109	3704	1719	5194	995.21	0.00%	1.0/sec	0.35	0.70	350.0
GET /countries	109	3356	869	9577	2163.83	0.00%	1.0/sec	8.14	0.18	8179.0
GET /states/{...	109	6651	1426	10127	1893.30	0.00%	1.1/sec	1.57	0.18	1508.0
GET /cities/{...	109	7029	4036	8875	1268.00	0.00%	1.1/sec	0.92	0.20	829.0
GET /sections	2063	40514	33141	74754	8415.12	0.00%	8.4/sec	3.51	1.56	427.0
GET /levels	329	39327	33321	72107	6007.84	0.00%	1.9/sec	0.94	0.34	520.0
GET /courses	29	37251	33728	57725	7059.23	0.00%	16.6/min	0.26	0.05	957.0
GET /courses...	3	37151	37140	37160	8.38	0.00%	4.8/min	0.08	0.01	974.0
TOTAL	31028	106786	74	429352	108875.78	0.62%	68.1/sec	53.73	16.65	807.4

Apéndice 7.4. Resultados de solicitudes atendidas de la prueba con la arquitectura monolítica configurados 3000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-3000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	6312	58371	74	126072	21954.66	0.00%	23.9/sec	35.16	4.35	1504.0
GET /progra...	2991	66560	155	111401	23257.78	0.00%	20.4/sec	11.60	3.49	582.0
GET /locatio...	5007	59818	658	103949	18560.70	0.00%	19.0/sec	18.10	3.51	976.0
GET /periods	3000	71412	611	99076	11868.05	0.00%	14.8/sec	7.94	2.70	548.0
POST /login	3480	239988	16959	293412	69631.87	0.00%	10.8/sec	6.26	2.74	591.0
GET /course-...	3000	54328	18091	96711	10454.07	0.00%	12.3/sec	4.30	2.29	357.0
PUT /update...	490	15726	1584	29788	7550.07	0.00%	2.4/sec	0.83	0.72	355.0
POST /recov...	490	3203	722	9728	1484.81	0.00%	2.5/sec	0.91	0.67	378.0
GET /data/{u...	109	4318	1609	12659	2480.69	0.00%	56.6/min	0.67	0.16	729.0
PUT /data/{u...	109	3704	1719	5194	995.21	0.00%	1.0/sec	0.35	0.70	350.0
GET /countries	109	3356	869	9577	2163.83	0.00%	1.0/sec	8.14	0.18	8179.0
POST /register	3098	361555	67618	429352	59620.93	0.00%	6.8/sec	2.27	4.69	341.0
GET /states/{...	109	6651	1426	10127	1893.30	0.00%	1.1/sec	1.57	0.18	1508.0
GET /cities/{...	109	7029	4036	8875	1268.00	0.00%	1.1/sec	0.92	0.20	829.0
GET /sections	2063	40514	33141	74754	8415.12	0.00%	8.4/sec	3.51	1.56	427.0
GET /levels	329	39327	33321	72107	6007.84	0.00%	1.9/sec	0.94	0.34	520.0
GET /courses	29	37251	33728	57725	7059.23	0.00%	16.6/min	0.26	0.05	957.0
GET /courses...	3	37151	37140	37160	8.38	0.00%	4.8/min	0.08	0.01	974.0
TOTAL	30837	107313	74	429352	109003.88	0.00%	67.7/sec	52.88	16.64	799.5

Apéndice 7.5. Resultados completos de la prueba con la arquitectura monolítica configurados 5000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-5000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	8682	81525	496	127172	24831.03	0.00%	29.8/sec	43.78	5.41	1504.0
GET /progra...	5000	86500	925	129999	29969.94	0.00%	29.2/sec	16.60	4.99	582.0
POST /login	5353	354915	26282	393347	75938.34	0.00%	12.5/sec	7.24	3.16	594.0
GET /locatio...	5500	88040	26029	119087	12561.36	0.00%	19.0/sec	18.07	3.50	976.0
PUT /update...	409	16072	2473	36582	9412.69	0.00%	1.9/sec	0.66	0.57	355.0
GET /periods	5000	88433	31637	119089	11394.04	0.00%	19.9/sec	10.67	3.62	548.0
POST /recov...	353	3681	1291	9776	2120.05	0.00%	1.9/sec	0.71	0.52	378.0
GET /course-...	3805	76523	61054	118344	10944.57	0.00%	14.7/sec	5.12	2.72	357.0
POST /register	5005	546515	174902	632053	83177.66	0.00%	7.6/sec	2.52	5.21	341.0
GET /data/{u...	10	5783	4777	7355	1012.10	0.00%	13.7/min	0.16	0.04	729.0
PUT /data/{u...	10	7125	4910	11077	2186.58	0.00%	13.7/min	0.08	0.16	350.0
GET /countries	10	6390	3732	9645	2662.80	0.00%	15.4/min	2.05	0.04	8179.0
GET /states/{...	10	7776	3987	12299	3177.46	0.00%	16.5/min	0.40	0.05	1508.0
GET /cities/{...	5	7736	6609	8055	565.21	0.00%	28.2/min	0.38	0.08	829.0
GET /sections	500	72915	61368	92734	9132.38	0.00%	3.1/sec	1.29	0.57	427.0
GET /levels	41	61921	61760	62120	130.24	0.00%	34.6/min	0.29	0.10	520.0
TOTAL	39693	177366	496	632053	173868.74	0.00%	59.8/sec	45.54	15.37	779.6

Apéndice 7.6. Resultados de solicitudes atendidas de la prueba con la arquitectura monolítica configurados 5000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-5000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /enroll...	8682	81525	496	127172	24831.03	0.00%	29.8/sec	43.78	5.41	1504.0
GET /progra...	5000	86500	925	129999	29969.94	0.00%	29.2/sec	16.60	4.99	582.0
POST /login	5353	354915	26282	393347	75938.34	0.00%	12.5/sec	7.24	3.16	594.0
GET /locatio...	5500	88040	26029	119087	12561.36	0.00%	19.0/sec	18.07	3.50	976.0
PUT /update...	409	16072	2473	36582	9412.69	0.00%	1.9/sec	0.66	0.57	355.0
GET /periods	5000	88433	31637	119089	11394.04	0.00%	19.9/sec	10.67	3.62	548.0
POST /recov...	353	3681	1291	9776	2120.05	0.00%	1.9/sec	0.71	0.52	378.0
GET /course-...	3805	76523	61054	118344	10944.57	0.00%	14.7/sec	5.12	2.72	357.0
POST /register	5005	546515	174902	632053	83177.66	0.00%	7.6/sec	2.52	5.21	341.0
GET /data/{u...	10	5783	4777	7355	1012.10	0.00%	13.7/min	0.16	0.04	729.0
PUT /data/{u...	10	7125	4910	11077	2186.58	0.00%	13.7/min	0.08	0.16	350.0
GET /countries	10	6390	3732	9645	2662.80	0.00%	15.4/min	2.05	0.04	8179.0
GET /states/{...	10	7776	3987	12299	3177.46	0.00%	16.5/min	0.40	0.05	1508.0
GET /cities/{...	5	7736	6609	8055	565.21	0.00%	28.2/min	0.38	0.08	829.0
GET /sections	500	72915	61368	92734	9132.38	0.00%	3.1/sec	1.29	0.57	427.0
GET /levels	41	61921	61760	62120	130.24	0.00%	34.6/min	0.29	0.10	520.0
TOTAL	39693	177366	496	632053	173868.74	0.00%	59.8/sec	45.54	15.37	779.6

Apéndice 7.7. Resultados completos de la prueba con la arquitectura monolítica configurados 8000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-8000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	8839	103529	890	269548	85822.25	20.48%	25.8/sec	22.40	3.50	890.4
GET /enroll...	21295	65428	183	271670	85982.29	57.18%	62.0/sec	111.62	4.82	1844.4
GET /periods	6915	45911	656	215584	64278.73	61.95%	20.2/sec	29.90	1.40	1511.9
GET /locatio...	16598	41974	37	226692	64319.60	71.85%	48.6/sec	84.79	2.52	1786.5
GET /course-...	5548	37921	1497	224249	61237.59	75.63%	16.3/sec	26.70	0.74	1676.3
POST /login	11655	358363	1494	609024	267001.10	36.75%	18.7/sec	20.86	2.99	1144.0
GET /sections	4651	34053	1539	226695	59959.82	81.75%	13.9/sec	24.38	0.47	1796.4
PUT /update...	4286	20819	1493	152150	34211.46	86.26%	14.2/sec	25.77	0.59	1862.0
POST /recov...	3775	10630	1196	79059	6252.54	83.84%	14.8/sec	26.32	0.65	1820.2
POST /register	10875	520450	860	952348	388891.71	36.19%	11.3/sec	10.79	4.98	976.1
GET /data/{u...	3812	8830	37	79081	5571.58	88.72%	16.0/sec	30.25	0.30	1938.3
PUT /data/{u...	3664	9978	231	79062	5965.84	83.49%	15.4/sec	27.13	1.75	1809.3
GET /countries	3504	11322	1017	79092	7543.35	79.14%	14.8/sec	48.56	0.53	3365.7
GET /states/{...	3306	12718	790	51398	8569.49	76.62%	16.1/sec	30.77	0.65	1961.9
GET /levels	3834	31174	1540	226635	55667.87	85.47%	12.0/sec	22.01	0.32	1873.7
GET /courses	3242	33236	1493	217857	59244.73	84.98%	11.2/sec	21.04	0.30	1929.9
GET /cities/{...	3049	10451	1531	78924	5894.37	78.16%	13.1/sec	23.22	0.50	1817.6
GET /courses...	2578	32299	1580	224238	57758.37	85.65%	9.2/sec	17.36	0.23	1939.4
GET /broadc...	2104	30916	1569	209823	55105.33	85.93%	7.6/sec	14.21	0.19	1906.3
POST /enroll...	1709	35604	1645	279546	67267.26	85.84%	6.0/sec	10.96	0.19	1855.5
POST /enroll...	1403	39215	1561	268866	70981.84	84.25%	5.2/sec	9.20	0.19	1825.9
GET /enroll...	1090	31828	1494	208226	54175.99	84.59%	4.4/sec	8.59	0.12	2014.7
TOTAL	127732	112479	37	952348	216486.45	63.43%	132.7/sec	214.94	13.40	1659.2

Apéndice 7.8. Resultados de solicitudes atendidas de la prueba con la arquitectura monolítica configurados 8000 hilos concurrentes.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename C:\Users\TEC02-MAQ14\Desktop\resultados\monolith-sep-endpoint-summary-8000.csv Browse... Log/Display Only: Errors Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/...	Sent KB/sec	Avg. Bytes
GET /progra...	7029	122268	890	269548	83304.14	0.00%	20.5/sec	11.64	3.50	582.0
GET /enroll...	9118	136231	889	271670	88519.63	0.00%	26.5/sec	38.97	4.82	1504.0
GET /periods	2631	107134	1405	215584	69168.38	0.00%	7.7/sec	4.12	1.40	548.0
GET /locatio...	4672	126000	1401	226692	69441.31	0.00%	13.7/sec	13.04	2.52	976.0
GET /course-...	1352	128060	5369	224249	67768.62	0.00%	4.0/sec	1.39	0.74	357.0
POST /login	7372	554018	13636	609024	87660.52	0.00%	11.8/sec	6.82	2.99	591.0
GET /sections	849	146680	15930	226695	64112.28	0.00%	2.5/sec	1.06	0.47	427.0
PUT /update...	589	96332	5956	152150	42619.63	0.00%	1.9/sec	0.68	0.59	355.0
POST /recov...	610	18789	1878	79059	8323.29	0.00%	2.4/sec	0.88	0.65	378.0
GET /data/{u...	430	17818	7810	79081	6870.23	0.00%	1.8/sec	1.28	0.30	729.0
PUT /data/{u...	605	16454	2579	79062	7523.89	0.00%	2.5/sec	0.87	1.75	350.0
GET /countries	731	21832	4525	79092	8490.77	0.00%	3.1/sec	24.89	0.54	8179.0
GET /states/{...	773	24594	3867	51398	8575.57	0.00%	3.8/sec	5.54	0.65	1508.0
GET /cities/{...	666	16834	6917	78924	6924.03	0.00%	2.9/sec	2.38	0.51	829.0
GET /levels	557	160815	69769	226635	39564.14	0.00%	1.7/sec	0.89	0.32	520.0
GET /courses	487	170946	114140	217857	30938.74	0.00%	1.7/sec	1.57	0.30	957.0
GET /courses...	370	170431	114137	224238	29317.15	0.00%	1.3/sec	1.25	0.23	974.0
GET /broadc...	296	164251	114132	209823	28301.68	0.00%	1.1/sec	0.77	0.19	716.0
GET /enroll...	168	156205	114685	208226	25520.26	0.00%	40.5/min	0.97	0.12	1469.0
POST /enroll...	242	196872	129855	279546	39544.96	0.00%	51.4/min	0.29	0.19	342.0
POST /enroll...	221	199634	129871	268866	36753.49	0.00%	48.8/min	0.27	0.19	342.0
POST /register	6939	804949	595184	952348	112518.84	0.00%	7.2/sec	2.40	4.98	341.0
TOTAL	46707	288480	889	952348	280735.12	0.00%	48.5/sec	42.36	13.40	894.3