EthAIca. 2024; 3:104
doi: 10.56294/ai2024104

ORIGINAL

Low-Code Platform for Visual Creation and Automatic Generation of Microservice Architectures

Plataforma Low-Code para la Creación Visual y Generación Automática de Arquitecturas de Microservicios

Tomás Darquier[1], Pablo Alejandro Virgolini[1]

[1]Universidad Siglo 21, Licenciatura en Informática, S.C de Bariloche. Argentina.

Citar como: Darquier T, Alejandro Virgolini P. Low-Code Platform for Visual Creation and Automatic Generation of Microservice Architectures. EthAIca. 2024; 3:104. https://doi.org/10.56294/ai2024104

Enviado: 03-06-2023 **Revisado:** 10-09-2023 **Aceptado:** 30-12-2023 **Publicado:** 01-01-2024

Editor: PhD. Rubén González Vallejo

ABSTRACT

With the growing adoption of the microservices paradigm, numerous benefits have been achieved in software development. Nonetheless, this methodology also has certain drawbacks. Through various data collection techniques, one identified issue is the unnecessary repetition in the development of common components for generic systems. Developers are often required to recreate these components multiple times across different systems and must manually configure the communications between them, which is time-consuming and increases development complexity. To address this issue, a web platform was developed that, through template-based dynamic code generation, facilitates the creation and configuration of microservices architectures via an intuitive graphical interface. Through a visual process, users can select and connect generic microservices, structuring their architectures in a personalized way that suits their requirements. The interaction is straightforward: developers drag and drop elements onto a canvas and visually establish the connections between them. Upon completion, they obtain the generated code, reducing the development of a fully functional distributed system to just a few clicks.

Keywords: Microservices; Web Platform; Code Generation; Distributed Architecture.

RESUMEN

Con la creciente adopción del paradigma de microservicios, se han obtenido múltiples beneficios en el desarrollo de software. Aun así, esto no quita que la mencionada metodología también presente ciertos inconvenientes. Se conoció, mediante múltiples técnicas de recolección de datos, que uno de ellos es la repetición innecesaria en el desarrollo de componentes comunes en sistemas genéricos. Los desarrolladores se ven obligados a recrear estos componentes en múltiples ocasiones en sus diferentes sistemas y, además, deben configurar manualmente las comunicaciones entre ellos, lo que consume tiempo y aumenta la complejidad del desarrollo. En respuesta a esta problemática, se desarrolló una plataforma web que, mediante la generación dinámica de código basada en plantillas, facilita la creación y configuración de arquitecturas de microservicios mediante una interfaz gráfica intuitiva. A través de un proceso visual, se les permite a los usuarios seleccionar y conectar microservicios genéricos, estructurando sus arquitecturas de manera personalizada y adecuada a sus requerimientos. La interacción es sencilla: los desarrolladores arrastran y sueltan los elementos sobre un lienzo y establecen visualmente las conexiones entre ellos. Al finalizar, obtienen el código generado, reduciendo a unos cuantos clics el desarrollo de un sistema distribuido completamente funcional.

Palabras clave: Microservicios; Plataforma Web; Generación de Código; Arquitectura Distribuida.

INTRODUCCIÓN

El presente proyecto consistió en una plataforma web orientada a desarrolladores de software que, mediante una interfaz gráfica, permite crear microservicios personalizados con funcionalidades genéricas y, a través de la configuración de su interrelación, obtener arquitecturas funcionales mediante mínima inversión de tiempo.[1,2,3,4,5]

La adopción de arquitecturas de microservicios para el desarrollo de software en la nube por parte de grandes compañías como eBay, Amazon y Netflix impulsó una tendencia masiva en la industria.[6,7,8,9,10]

Este cambio trajo consigo múltiples beneficios en términos de escalabilidad, mantenimiento y encapsulación de código.[11,12,13,14,15] Sin embargo, también resultó en un notable incremento en la complejidad durante el desarrollo en comparación con una solución monolítica, lo que implicó un mayor requerimiento de capital humano, especialmente en las fases de codificación, pruebas e integración de los módulos.[1,2,16,17,18,19]

A su vez, el movimiento Low-code, el cual protagonizó su origen en la última década, permitió a muchas organizaciones beneficiarse de su utilidad, como Gartner, quienes comentan que para la finalización del presente año más de un 65 % de los desarrollos provendrán de este medio.[3,20,21,22]

Debido a que tanto el origen de la arquitectura de microservicios, como el movimiento Low-code son relativamente cercanos, no abundan combinaciones entre estos tópicos,[23,24,25,26,27] sin embargo, la aparición reciente de múltiples artículos académicos alrededor de este tema,[28,29,30,31] indica un interés creciente en explorar cómo la integración de enfoques Low-code con arquitecturas de microservicios puede optimizar el desarrollo de software.[32,33,34,35]

Un claro ejemplo de lo mencionado, y con cierta relación con el presente proyecto, se trata de la idea de un lenguaje específico para microservicios, como un paso hacia la integración de plataformas Low-code, propuesto por Said et al.[4].

¿Cómo facilitar el diseño, configuración e integración de arquitecturas de microservicios mediante un enfoque Low-code, reduciendo la complejidad y el tiempo de desarrollo sin comprometer la calidad del software?

Objetivo

Desarrollar e implementar un sistema Low-code que facilite a los desarrolladores diseñar, configurar e integrar arquitecturas de microservicios de forma eficiente mediante interfaces visuales y componentes reutilizables, optimizando el tiempo y reduciendo la complejidad en comparación con los enfoques tradicionales, asegurando la calidad mediante validaciones previas.

MÉTODO

Herramientas Metodológicas

Durante el desarrollo del presente sistema se siguieron los lineamientos establecidos por la metodología ágil Scrum, un marco de trabajo que facilita la colaboración entre equipos para entregar productos de manera iterativa e incremental, permitiendo adaptarse rápidamente a los cambios e incentivando la mejora continua.[5]

De esta forma, al finalizar cada Sprint, se logró obtener porciones funcionales de código, aunque el sistema completo no esté desarrollado, aprendiendo en cada iteración y aplicando el conocimiento en la siguiente a realizar. Así el producto se benefició de las características mencionadas, permitiendo incluso cambiar de tecnologías, en base al conocimiento obtenido durante el proceso de desarrollo sin represalia alguna.

Herramientas de Desarrollo

En el desarrollo del proyecto fueron utilizadas múltiples tecnologías, las cuales serán explicadas y justificadas a continuación, organizadas en 4 grupos según su actividad especifica: herramientas referidas al Front-end, aquellas destinadas al Back-end para autenticación lógica básica e intercambio con el Front-end embebido en el Gateway, por otra parte las tecnologías orientadas al Back-end para la generación automática de código y, finalmente, las tecnologías utilizadas para facilitar el despliegue y escalabilidad del sistema.

En cuanto a las herramientas empleadas en el desarrollo del Front-end se encuentra, el ya estándar conjunto de JavaScript, HTML5 y CSS, permitiendo crear un aspecto visual completo y con alta compatibilidad, además de una correcta y adaptable lógica de comunicación con el Back-end. A su vez, estas herramientas fueron potenciadas con Boostrap, que con la ayuda de sus componentes prediseñados alivianó la carga de trabajo referida al diseño estético, permitiendo gestionar múltiples funcionalidades dentro de la misma página sin descuidar este aspecto. De esta forma y con la ayuda de Thymeleaf el Front-end fue acoplado al API Gateway para evitar complejidades innecesarias.

El Back-end fue desarrollado en su gran mayoría en Java 17, haciendo uso del framework Spring y de sus conocidas librerías para una correcta, limpia y eficiente comunicación entre servicios mediante, en su mayoría, el estilo de arquitectura REST. La seguridad se administró e implementó mediante OAuth2 partiendo del servicio ofrecido por Okta, un estándar que permite a sitios web o aplicaciones acceder a recursos alojados en otras aplicaciones, en nombre y con permiso previo, del usuario. Los aspectos que requirieron persistencia de datos relacional fueron provistos de una instancia de base de datos PostgreSQL, elegida debido a su entorno

open source, además de su destacable flexibilidad, integridad de datos y escalabilidad. Además, se manejaron comunicaciones asíncronas para él envió de notificaciones mediante Apache Kafka.

La lógica referida a la generación dinámica de código se administró semánticamente mediante RDF, que permitió especificar de forma correcta y estructurada las combinaciones y decisiones realizadas por el usuario en el canvas presente en el Front- end, para poder así ser enviadas al Back-end, consultadas a través del lenguaje de consulta SPARQL y administradas mediante Apache Jena en los servicios Java. Para concluir el proceso y generar el código especificado y solicitado por el usuario se decidió utilizar Apache Velocity, debido a su alta capacidad en la tarea requerida y MinIO, para el almacenamiento y descarga de los archivos resultantes, evadiendo de esta forma, dependencia a soluciones de almacenamiento en la nube específicas.

Finalmente, la administración del despliegue y escalabilidad del sistema se llevó a cabo mediante Docker, potenciado con Kubernetes. De esta forma, al igual que otras decisiones previamente expuestas, se reduce el acoplamiento a soluciones cloud brindando al sistema una mayor versatilidad y capacidad independiente de despliegue.

Recolección de Datos

Para comprender adecuadamente el problema a abordar, se inició con el análisis de fuentes bibliográficas académicas. Este enfoque permitió identificar los principales desafíos en el desarrollo de arquitecturas distribuidas, así como los costos asociados.

Otro método de recolección de datos utilizado fue el análisis de redes sociales. Esta metodología, debido al potencial público del proyecto, resultó altamente enriquecedora gracias a plataformas como Twitter, Reddit y Medium, donde desarrolladores de software comparten y discuten ideas y experiencias sobre temas y situaciones del ámbito informático.

De esta forma, se obtuvieron dos enfoques ciertamente antagónicos, justificando de esta manera la elección de las técnicas presentadas. Estos enfoques se dividen en los estrictamente teóricos, basados en documentos académicos, y por el otro extremo, mediante redes sociales, los enfoques basados en las experiencias diarias y cotidianas de los desarrolladores, quienes, al fin y al cabo, son los perjudicados por la problemática.

Planificación del Proyecto

Para facilitar la comprensión, se presentará primero la planificación de forma separada, y luego se mostrará el diagrama de Gantt utilizado para organizar los objetivos del presente Trabajo Final de Graduación.

A continuación, se presentan las tareas especificadas en conjunto con las fechas correspondientes para, posteriormente presentar el diagrama de Gantt mencionado.

Nombre de tarea	Fecha de inici	Fecha final
	07/08/2024	09/11/2024
Temática y Relevamiento	**07/08/2024**	05/09/2024
Elección y Presentación de Temática	07/08/2024	**19/08/2024**
Título, Introducción y Justificación	20/08/2024	**22/08/2024**
Objetivo General y Espeíficos	23/08/2024	**25/08/2024**
Marco Teórico Referencial	26/08/2024	**28/08/2024**
Diseño Metodologico	29/08/2024	**31/08/2024**
Relevamiento	01/09/2024	**03/09/2024**
Proceso de Negocios	04/09/2024	**05/09/2024**
Propuesta y Definición del Prototipo	**06/09/2024**	23/09/2024
Diagnóstico y Propuesta	06/09/2024	**08/09/2024**
Objetivos, Límites y Alcances	09/09/2024	**11/09/2024**
Descripción del Sistema	12/09/2024	**15/09/2024**
Estructura de Datos	16/09/2024	**18/09/2024**
Prototipos de Interfaces	19/09/2024	**22/09/2024**
Diagrama de Arquitectura	23/09/2024	**23/09/2024**
Riesgos y Control	**24/09/2024**	01/10/2024
Documentación de Seguridad	24/09/2024	**25/09/2024**
Análisis de Costos	26/09/2024	**28/09/2024**
Análisis de Riesgos	29/09/2024	**01/10/2024**
Finalización de Entregable	**02/10/2024**	06/10/2024
Conclusiones y Anexos	02/10/2024	**04/10/2024**
Revisión y Corrección General	05/10/2024	**06/10/2024**
Desarrollo y Pruebas del Prototipo	**23/09/2024**	09/11/2024
Desarrollo del Prototipo	23/09/2024	**05/11/2024**
Pruebas del Prototipo	**02/11/2024**	**09/11/2024**

Figura 1. Plan de Desarrollo

https://doi.org/10.56294/ai2024104 ISSN: 3072-7952

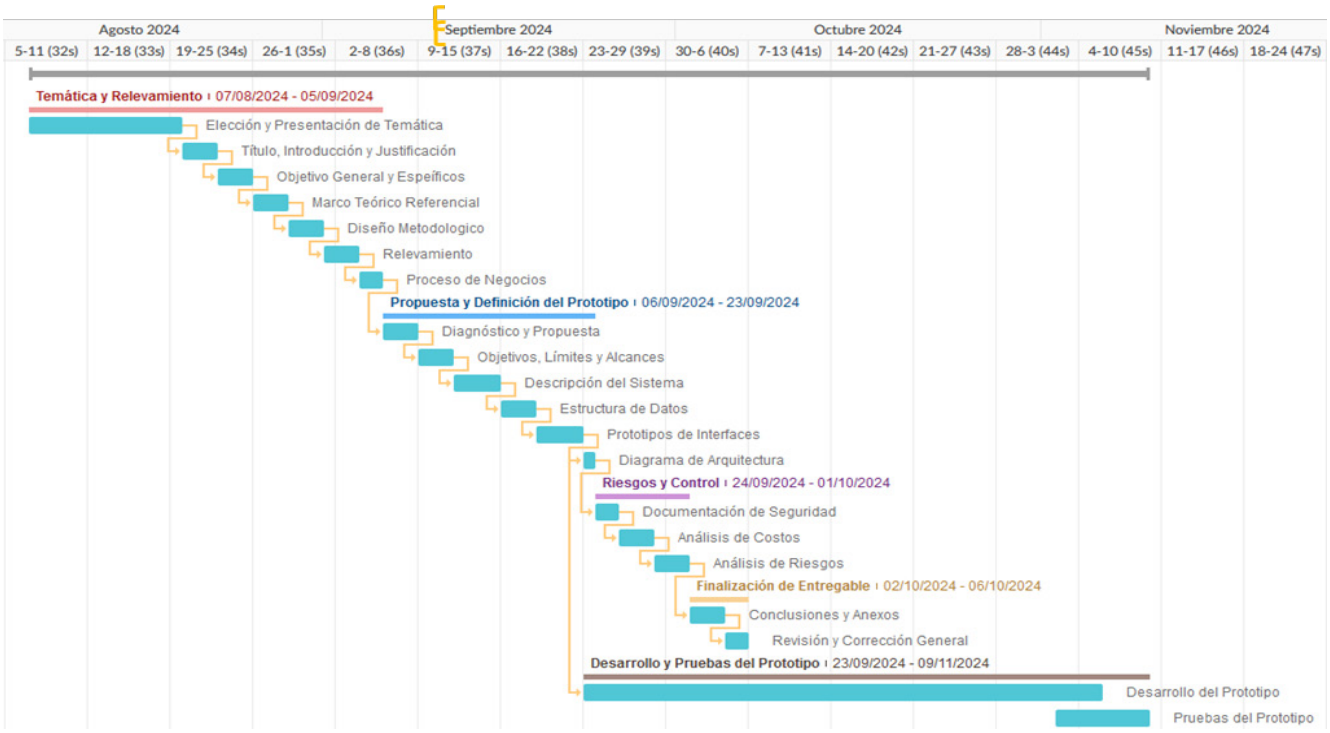

Figura 2. Diagrama de Gantt

RESULTADOS

Relevamiento

Relevamiento Estructural

Dado que el proyecto desarrollado se trató de una plataforma web dirigida a desarrolladores de software, tanto empleados como freelancers, no es posible establecer una localización específica, ya que este aspecto depende de cada usuario particular que acceda a la plataforma.

Sin embargo, se ha identificado mediante los métodos de recolección de datos especificados previamente, que la mayoría de los desarrolladores utilizan, propias o de la empresa empleadora, computadoras portátiles, y en menor medida, equipos de escritorio, a través de los cuales realizan sus tareas profesionales en el ámbito informático.

Relevamiento Funcional

Estructura Jerárquica

A continuación, se presenta el organigrama genérico de una empresa pequeña o mediana de desarrollo de software, debido a la naturaleza de usuarios a los que apunta el proyecto.

Se encuentran coloreados los sectores alcanzados por la plataforma desarrollada.

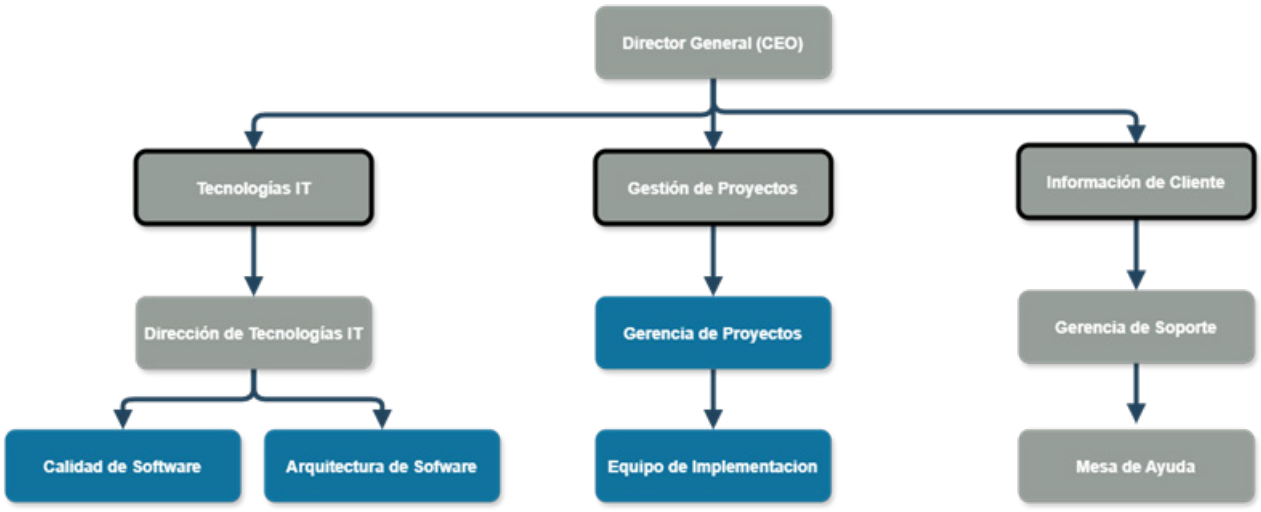

Figura 3. Organigrama Modelo de Empresa de Desarrollo de Software

Funciones de las Áreas

Calidad de Software: se encarga de asegurar que el software cumple con los requisitos previamente especificados y que este no posee errores. También implementa pruebas y controla la calidad del producto de forma continua durante el ciclo de vida del desarrollo.

Arquitectura de Software: tiene el objetivo de definir la estructura general del sistema, incluyendo las tecnologías, patrones de diseño y las interacciones entre los diferentes servicios que componen el producto final. A su vez se enfoca en asegurar escalabilidad y eficiencia en el software.

Equipo de Implementación: desarrolla, implementa y mantiene el producto de software. Realiza la tarea de escribir el código, solucionar errores y participar en la creación de nuevas funcionalidades indicadas por los requerimientos y especificaciones de diseño.

Gerencia de Proyectos: establece que recursos se necesitan para realizar el proyecto y supervisa la ejecución y desarrollo del mismo, a su vez, asigna las tareas correspondientes y define sus plazos.

Procesos Relevados

Nombre de proceso: control de Calidad del Software.

Roles: calidad de Software, Equipo de Implementación, Arquitectura de Software, Gerencia de Proyectos.

Pasos: el departamento de Calidad de Software establece y documenta los estándares y criterios de calidad que deben cumplir los desarrollos de software, de forma coordinada con el departamento de Arquitectura de Software para asegurar la alineación técnica y de diseño.

Luego, el equipo de Calidad de Sofware desarrolla un plan de pruebas detallado, el cual es compartido con el Equipo de Implementación para garantizar que todas las funcionalidades sean cubiertas y probadas mediante las pruebas especificadas.

Finalmente, el equipo de Calidad de Software realiza las pruebas y evalúa los resultados para, si es necesario realizar una retroalimentación al Equipo de Implementación. En caso contrario la Gerencia de Proyectos recibe el producto y se encarga de su liberación final.

Nombre de proceso: diseño de Arquitectura de Software.

Roles: Arquitectura de Software, Calidad de Software, Gerencia de Proyectos.

Pasos: la Gerencia de Proyectos, trabaja para comprender y documentar los diferentes requisitos del proyecto a tratar.

Una vez definidos los requisitos, el Departamento de Arquitectura de Software diseña y documenta la arquitectura de forma completa, incluyendo patrones de diseño, frameworks y tecnologías. Una vez realizado se comunica el resultado al Equipo de Calidad.

El Departamento de Calidad verifica los estándares de calidad de la arquitectura desarrollada, invitando a realizar ajustes en caso de ser necesario.

Nombre de proceso: Desarrollo de Software.

Roles: equipo de Implementación, Gerencia de Proyectos, Arquitectura de Software, Calidad de Software.

Pasos: la Gerencia de Proyectos distribuye tareas y asigna recursos al Equipo de Implementación según el cronograma y las prioridades que hayan sido asignadas a cada tarea del proyecto.

El equipo de Implementación desarrolla las funcionalidades según el diseño arquitectónico y sus especificaciones detalladas por el departamento de Arquitectura de Software, estableciendo comunicación mutua de forma constante para garantizar la comprensión y cumplimiento de los lineamientos.

Previamente a enviar los módulos desarrollados a Calidad de Software, el Equipo de Implementación realiza pruebas internas para detectar y corregir errores. Una vez solucionados se proceden a enviar los módulos completos al departamento mencionado, quienes procederán con la realización de pruebas formales.

Cualquier defecto identificado por el Equipo de Calidad es informado, mediante un informe de pruebas, al Equipo de Implementación, quienes realizan las correcciones pertinentes. Este paso se trata de un ciclo que finaliza cuando se considera que los estándares de calidad especificados han sido cumplidos.

Relevamiento de la Documentación

A continuación, se mencionan los documentos relevados:

- Especificación de Estándares de Calidad: documento que describe los criterios y estándares de calidad que deben cumplir los desarrollos de software según lo definido por el equipo de Calidad de Software.
- Plan de Pruebas: documento que detalla el plan de pruebas, incluyendo tipos de pruebas, casos de prueba, y recursos asignados. Este plan es desarrollado por el equipo de Calidad de Software.
- Relevamiento de Requisitos: documento que lista los requisitos recopilados por la Arquitectura de Software en colaboración con la Gerencia de Proyectos. Es la base para el diseño de la arquitectura del software.
- Documentación de Arquitectura: documento que describe en detalle la arquitectura del software, incluyendo diagramas de componentes, interacciones y tecnologías utilizadas.

- Informe de Pruebas: documento generado por el equipo de Calidad de Software que resume los resultados de las pruebas realizadas, incluyendo defectos encontrados y acciones correctivas.

Proceso de Negocios

A continuación, se presenta el diagrama de flujo, referido al proceso integral llevado a cabo al momento de desarrollar un producto de software.

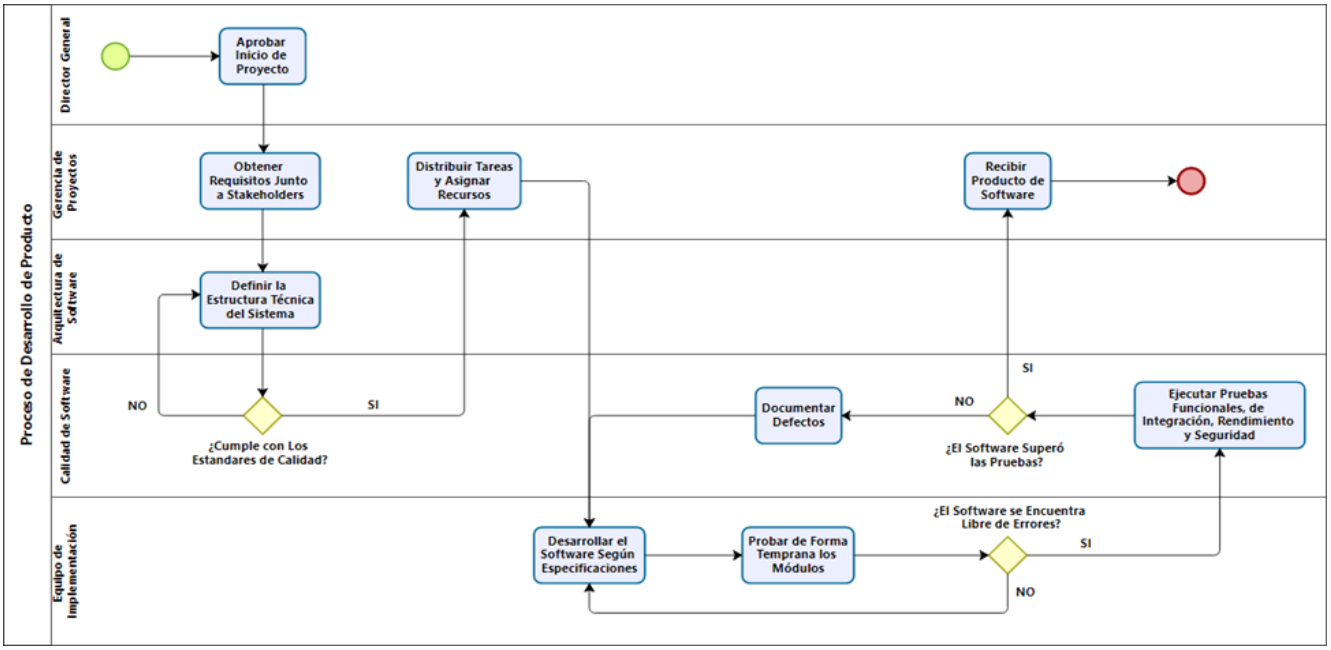

Figura 4. Diagrama de Flujo referido al Desarrollo de Producto

Diagnóstico y Propuesta

Tabla 1. Diagnóstico de proceso Control de Calidad de Software	
Nombre del Proceso: Control de Calidad de Software	
Problemas	**Causas**
Dependencia de pruebas manuales.	Presencia de componentes incorrectamente desacoplados, provocando la imposibilidad de automatizar pruebas de componentes aislados. Escasa documentación o ausencia de la misma referida a la composición y lógica interna de los componentes pertenecientes al producto de software.

Tabla 2. Diagnóstico de proceso Diseño de Arquitectura de Software	
Nombre del Proceso: Diseño de Arquitectura de Software	
Problemas	**Causas**
Dificultad para adaptar la arquitectura a nuevos requisitos del proyecto.	Los cambios en los requisitos pueden provocar que la arquitectura original, o la mayoría de la misma, quede obsoleta o no cubra las necesidades emergentes. La presencia de una arquitectura monolítica o poco modular limita la flexibilidad para adaptar partes específicas del sistema sin afectar otras. Falta de la documentación de la arquitectura o inaccesibilidad a la misma.
Falta de coordinación en la definición de la arquitectura entre equipos.	Los distintos departamentos no siempre están correctamente alineados con las decisiones arquitectónicas.

Tabla 3. Diagnóstico de proceso de Desarrollo de Software

Nombre del Proceso: Desarrollo de Software	
Problemas	**Causas**
Sobrecarga del equipo de desarrollo por falta de reutilización software.	No se cuenta con una estrategia clara o un repositorio centralizado para gestionar componentes reutilizables.
Identificación tardía de errores en el código antes de la entrega a calidad.	Debido a la falta de recursos de prueba continua, los errores suelen detectarse tarde en el ciclo de desarrollo.

Propuesta

El sistema desarrollado mejoró la eficiencia y claridad en el diseño y desarrollo de componentes de software. Permite al usuario crear arquitecturas de componentes altamente desacoplados y obtener el código de cada componente con documentación centralizada, accesible y clara. Esto facilita la coordinación entre equipos y asegura la alineación técnica en todas las fases del proyecto. La plataforma incorpora pruebas automáticas desde el inicio de la creación de componentes, reduciendo significativamente la dependencia de pruebas manuales y mejorando los procesos de testing y calidad. Además, el desacople lógico de los componentes fomenta su reutilización, acelerando el desarrollo y mejorando la adaptabilidad de las arquitecturas resultantes.

Objetivos, Límites y Alcance del Prototipo

Objetivos del Prototipo

Desarrollar un prototipo de sistema que permita el diseño de arquitecturas distribuidas mediante la combinación de componentes predefinidos, generando automáticamente el código ajustado a las especificaciones del usuario para su posterior descarga.

Límites

Desde la configuración y combinación de componentes predefinidos hasta la generación y descarga automática del código correspondiente según las especificaciones del usuario.

Alcances

A continuación, se listan los procesos comprendidos por el prototipo tecnológico:

- Registro e ingreso de usuarios.
- Diseño de arquitecturas mediante la combinación de microservicios.
- Gestión de composición de microservicios.
- Guías de usuario.
- Generación dinámica de componentes.
- Generación de documentación técnica.
- Validación y pruebas de la arquitectura generada.
- Descarga de la arquitectura resultante

CONCLUSIONES

A partir de la identificación de un patrón recurrente en el desarrollo de aplicaciones de microservicios, basado en que se suelen programar servicios con funcionalidades y lógica similares, se ideó, diseñó e implementó un sistema con el objetivo de ofrecer una alternativa que mejore la productividad del desarrollo, aportando múltiples beneficios para el desarrollador y por consecuencia, para el cliente y/o empleador de este.

El proyecto resultó en una plataforma web con una interfaz simple e intuitiva, construida con múltiples tecnologías y basando su núcleo lógico en plantillas para la generación dinámica de código. Esta herramienta logró brindar a los usuarios la capacidad de diseñar arquitecturas distribuidas de microservicios basadas en componentes comunes, arrastrando y conectando piezas sobre un canvas interactivo. De manera sencilla, se pueden generar miles de líneas de código, con el respaldo de documentación adecuada. Al descargar el código resultante de su diseño sobre el canvas, el usuario obtiene componentes totalmente compatibles entre sí, con configuraciones correctas, y una comunicación fluida y libre de errores entre los servicios.

A lo largo del desarrollo del sistema y de la elaboración de este documento, tuve la oportunidad de aplicar y profundizar en los conocimientos adquiridos durante mi carrera, aprendiendo de manera práctica en cada etapa del proceso, desde la concepción del proyecto hasta la documentación final. Este trabajo también me permitió enfrentar el verdadero desafío que implica construir un sistema desde cero, combinando diversas tecnologías de forma coherente y funcional. En este proceso, no solo consolidé lo aprendido, sino que también comprendí las complejidades que implican lograr una solución robusta y eficiente, y la satisfacción que genera tangibilidad en un producto final de software lo que, en un comienzo, no fue más que una idea.

REFERENCIAS BIBLIOGRÁFICAS

1. Elgheriani NS, Ahmed NA. Microservices vs. monolithic architectures. Int J Appl Sci Technol. 2022;4:501-14. doi:10.47832/2717-8234.12.47

2. Lopez BM, Garcia JL. Impacto de arquitecturas de microservicios en el desarrollo web [Tesis de maestría]. Madrid: Universidad Politécnica de Madrid; 2019. https://oa.upm.es/55917/1/TESIS_MASTER_BRUNO_MARTIN_LOPEZ.pdf

3. Vincent P, Lijima K, Driver M, Wong J, Natis Y. Gartner magic quadrant for enterprise low-code application platforms. Stamford: Gartner, Inc.; 2019. https://www.gartner.com/en/documents/3956079

4. Said M, Ezzati A, Arezki S. Microservice-specific language, a step to the low-code platforms. In: Lecture Notes in Networks and Systems. 2023;637:817-28. doi:10.1007/978-3-031-26384-2_72

5. Schwaber K, Sutherland J. Scrum: The art of doing twice the work in half the time. Houston: Crown Business; 2010.

6. Amazon Web Services. What is Docker? 2023. https://aws.amazon.com/es/docker/

7. Amazon Web Services. What is Java? 2023. https://aws.amazon.com/es/what-is/java/

8. Apache Software Foundation. What is Velocity? 2020. https://velocity.apache.org/

9. Apache Software Foundation. About Jena. 2024. https://jena.apache.org/about_jena/about.html

10. Apache Software Foundation. Apache Kafka. 2024. https://kafka.apache.org/

11. Banco Central de la República Argentina (BCRA). Cotizaciones por fecha. 2024. http://www.bcra.gob.ar/PublicacionesEstadisticas/Cotizaciones_por_fecha_2.asp

12. Brown T, Smith L. The impact of Low-code platforms and intuitive interfaces on software development efficiency. J Softw Innov. 2023;18:75-89.

13. Chaudhary HAA, Ahmed T. Integration of micro-services as components in modeling environments for low code development. ISP RAS. 2021;33(4):19-30. doi:10.15514/ISPRAS-2021-33(4)-2

14. Consejo Profesional de Ciencias Informáticas de la Provincia de Córdoba (CPCIPC). Honorarios Recomendados. 2024. https://cpcipc.org.ar/honorarios-recomendados/

15. Dhoke P, Lokulwar P. Evaluating the Impact of No-Code/Low-Code Backend Services on API Development and Implementation: A Case Study Approach. In: 14th International Conference on Computing Communication and Networking Technologies (ICCCNT); 2023 Jul 11-13; Chennai, India. Piscataway: IEEE; 2023. p. 1-5. doi:10.1109/ICCCNT56998.2023.10306945

16. DigitalOcean. DigitalOcean Managed Kubernetes. 2024. https://www.digitalocean.com/products/kubernetes

17. GitHub. About GitHub and Git. 2024. https://docs.github.com/es/get-started/start-your-journey/about-github-and-git

18. IBM. Minio. 2021. https://www.ibm.com/docs/es/cloud-private/3.2.x?topic=private-minio

19. IBM. PostgreSQL. 2024. https://www.ibm.com/mx-es/topics/postgresql

20. JetBrains. IntelliJ IDEA features. 2024. https://www.jetbrains.com/es-es/idea/features/

21. Kubernetes. ¿Qué es Kubernetes? 2022. https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

22. Lewis J, Fowler M. Microservices: a definition of this new architectural term. 2014. https://martinfowler.com/articles/microservices.html

23. Misic B, Novkovic M, Ramac R, Mandic V. Do the microservices improve the agility of software development teams? In: International Scientific Conference on Industrial Systems; 2017. 17:170-5.

24. Mozilla Developer Network. CSS. 2023. https://developer.mozilla.org/es/docs/Glossary/CSS

25. Mozilla Developer Network. HTML5. 2023. https://developer.mozilla.org/es/docs/Glossary/HTML5

26. Mozilla Developer Network. What is JavaScript? 2024. https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript

27. Newman S. Building microservices: Designing fine-grained systems. Newton: O'Reilly Media; 2015.

28. Postman. What is Postman? 2024. https://www.postman.com/product/what-is-postman/

29. Richardson C. Microservices patterns: With examples in Java. New York: Manning Publications; 2019.

30. Rock Content. What is Bootstrap? 2020. https://rockcontent.com/es/blog/bootstrap/

31. Spring. Spring Framework. 2024. https://spring.io/projects/spring-framework

32. The Thymeleaf Team. Thymeleaf. 2024. https://www.thymeleaf.org/

33. Trello. Trello Tour. 2023. https://trello.com/es/tour

34. World Wide Web Consortium. RDF 1.1 Concepts and Abstract Syntax. 2014. https://www.w3.org/TR/rdf11-concepts/

35. World Wide Web Consortium. SPARQL 1.1 Query Language. 2013. https://www.w3.org/TR/sparql11-query/

FINANCIACIÓN

Ninguna.

CONFLICTO DE INTERESES

Los autores declaran que no existe conflicto de intereses.

CONTRIBUCIÓN DE AUTORÍA

Conceptualización: Tomás Darquier, Pablo Alejandro Virgolini.
Curación de datos: Tomás Darquier, Pablo Alejandro Virgolini.
Análisis formal: Tomás Darquier, Pablo Alejandro Virgolini.
Investigación: Tomás Darquier, Pablo Alejandro Virgolini.
Metodología: Tomás Darquier, Pablo Alejandro Virgolini.
Administración del proyecto: Tomás Darquier, Pablo Alejandro Virgolini.
Recursos: Tomás Darquier, Pablo Alejandro Virgolini.
Software: Tomás Darquier, Pablo Alejandro Virgolini.
Supervisión: Tomás Darquier, Pablo Alejandro Virgolini.
Validación: Tomás Darquier, Pablo Alejandro Virgolini.
Visualización: Tomás Darquier, Pablo Alejandro Virgolini.
Redacción – borrador original: Tomás Darquier, Pablo Alejandro Virgolini.
Redacción – revisión y edición: Tomás Darquier, Pablo Alejandro Virgolini.