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.

Cite as: 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

Submitted: 03-06-2023 **Revised:** 10-09-2023 **Accepted:** 30-12-2023 **Published:** 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.

INTRODUCTION

This project consisted of a web platform aimed at software developers that, through a graphical interface, allows them to create customized microservices with generic functionalities and, by configuring their interrelationships, obtain functional architectures with minimal time investment.(1,2,3,4,5)

The adoption of microservice architectures for cloud software development by large companies such as eBay, Amazon, and Netflix drove a massive trend in the industry.(6,7,8,9,10)

This change brought multiple benefits in terms of scalability, maintenance, and code encapsulation.(11,12,13,14,15) However, it also resulted in a notable increase in complexity during development compared to a monolithic solution, which implied a greater requirement for human capital, especially in the coding, testing, and module integration phases.(1,2,16,17,18,19)

In turn, the low-code movement, which emerged in the last decade, allowed many organizations to benefit from its usefulness. Gartner, for example, estimates that by the end of this year, more than 65 % of developments will come from this medium.(3,20,21,22)

Because both the origin of microservice architecture and the low-code movement are relatively recent, there are not many combinations between these topics.(23,24,25,26,27) However, the recent appearance of multiple academic articles on this subject,(28,29,30,31) indicates a growing interest in exploring how the integration of low-code approaches with microservice architectures can optimize software development.(32,33,34,35)

A clear example of this, and somewhat related to the present project, is the idea of a specific language for microservices as a step towards the integration of low-code platforms, proposed by Said et al.(4)

How can we facilitate the design, configuration, and integration of microservice architectures using a low-code approach, reducing complexity and development time without compromising software quality?

Objective

To develop and implement a low-code system that makes it easier for developers to design, configure, and integrate microservice architectures efficiently using visual interfaces and reusable components, optimizing time and reducing complexity compared to traditional approaches, ensuring quality through prior validation.

METHOD

Methodological Tools

During the development of this system, the guidelines established by the agile Scrum methodology were followed, a framework that facilitates collaboration between teams to deliver products iteratively and incrementally, allowing for rapid adaptation to changes and encouraging continuous improvement.(5)

In this way, at the end of each Sprint, functional portions of code were obtained, even though the complete system was not developed, learning in each iteration and applying the knowledge in the next one. Thus, the product benefited from the aforementioned characteristics, even allowing for a change in technologies, based on the knowledge obtained during the development process without any repercussions.

Development Tools

Multiple technologies were used in the development of the project, which will be explained and justified below, organized into four groups according to their specific activity: front-end tools, those intended for the back-end for basic logical authentication and exchange with the front-end embedded in the gateway, back-end technologies for automatic code generation, and finally, technologies used to facilitate system deployment and scalability.

The tools used in the development of the front-end include the now standard set of JavaScript, HTML5, and CSS, allowing for the creation of a complete visual appearance with high compatibility, as well as correct and adaptable communication logic with the back-end. In turn, these tools were enhanced with Bootstrap, which, with the help of its pre-designed components, alleviated the workload related to aesthetic design, allowing multiple functionalities to be managed within the same page without neglecting this aspect. In this way, and with the help of Thymeleaf, the front-end was coupled to the API Gateway to avoid unnecessary complexities.

The back-end was developed mostly in Java 17, using the Spring framework and its well-known libraries for correct, clean, and efficient communication between services, mostly using the REST architecture style. Security was managed by the OpenID Connect protocol and implemented using OAuth2 based on the service offered by Okta, a standard that allows websites or applications to access resources hosted on other applications, on behalf of and with prior permission from the user. Aspects requiring relational data persistence were provided by a PostgreSQL database instance, chosen for its open source environment, as well as its remarkable flexibility, data integrity, and scalability. In addition, asynchronous communications were handled for sending notifications using Apache Kafka.

The logic related to dynamic code generation was semantically managed using RDF, which allowed the combinations and decisions made by the user on the front-end canvas to be specified correctly and in a structured manner, so that they could be sent to the back-end, queried using the SPARQL query language, and managed using Apache Jena in Java services. To complete the process and generate the code specified and requested by the user, we decided to use Apache Velocity, due to its high capacity for the task required, and MinIO for storing and downloading the resulting files, thus avoiding dependence on specific cloud storage solutions.

Finally, the deployment and scalability of the system was managed using Docker, powered by Kubernetes. In this way, as with other decisions outlined above, dependence on cloud solutions is reduced, giving the system greater versatility and independent deployment capabilities.

Data Collection

To properly understand the problem to be addressed, we began with an analysis of academic bibliographic sources. This approach allowed us to identify the main challenges in the development of distributed architectures, as well as the associated costs.

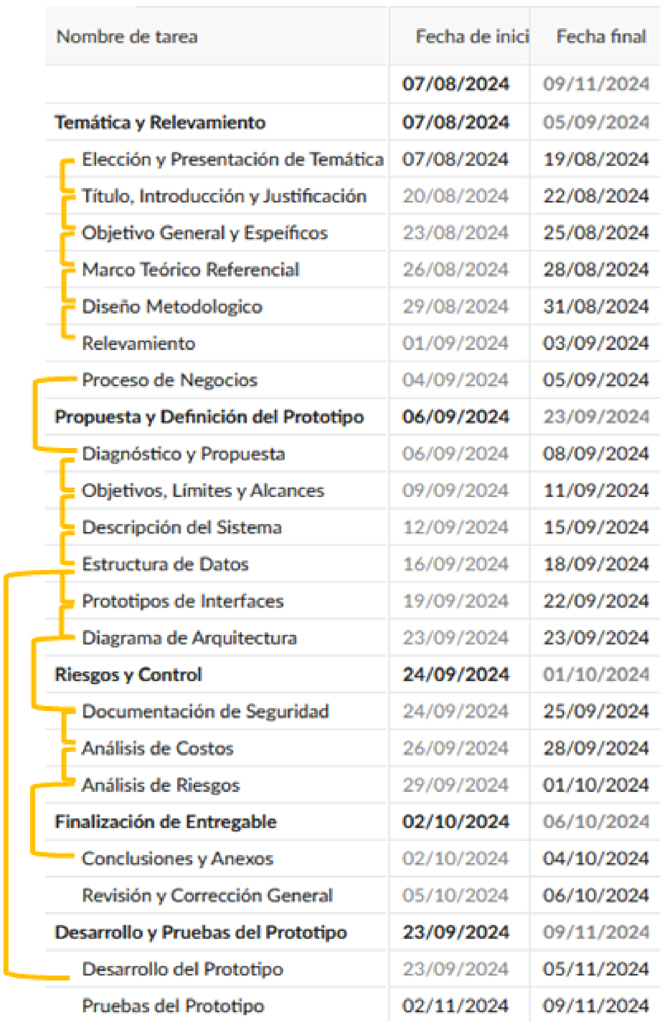
Another data collection method used was social media analysis. Due to the public nature of the project, this methodology proved highly enriching thanks to platforms such as Twitter, Reddit, and Medium, where software developers share and discuss ideas and experiences on topics and situations in the field of computing.

This resulted in two clearly antagonistic approaches, thus justifying the choice of techniques presented. These approaches are divided into strictly theoretical ones, based on academic documents, and, at the other extreme, those based on social networks, which draw on the daily experiences of developers, who, after all, are the ones affected by the problem.

Project Planning

For ease of understanding, the planning will first be presented separately, followed by the Gantt chart used to organize the objectives of this Final Graduation Project.

Next, the tasks specified are presented together with the corresponding dates, followed by the aforementioned Gantt chart.



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

Figure 1. Development Plan

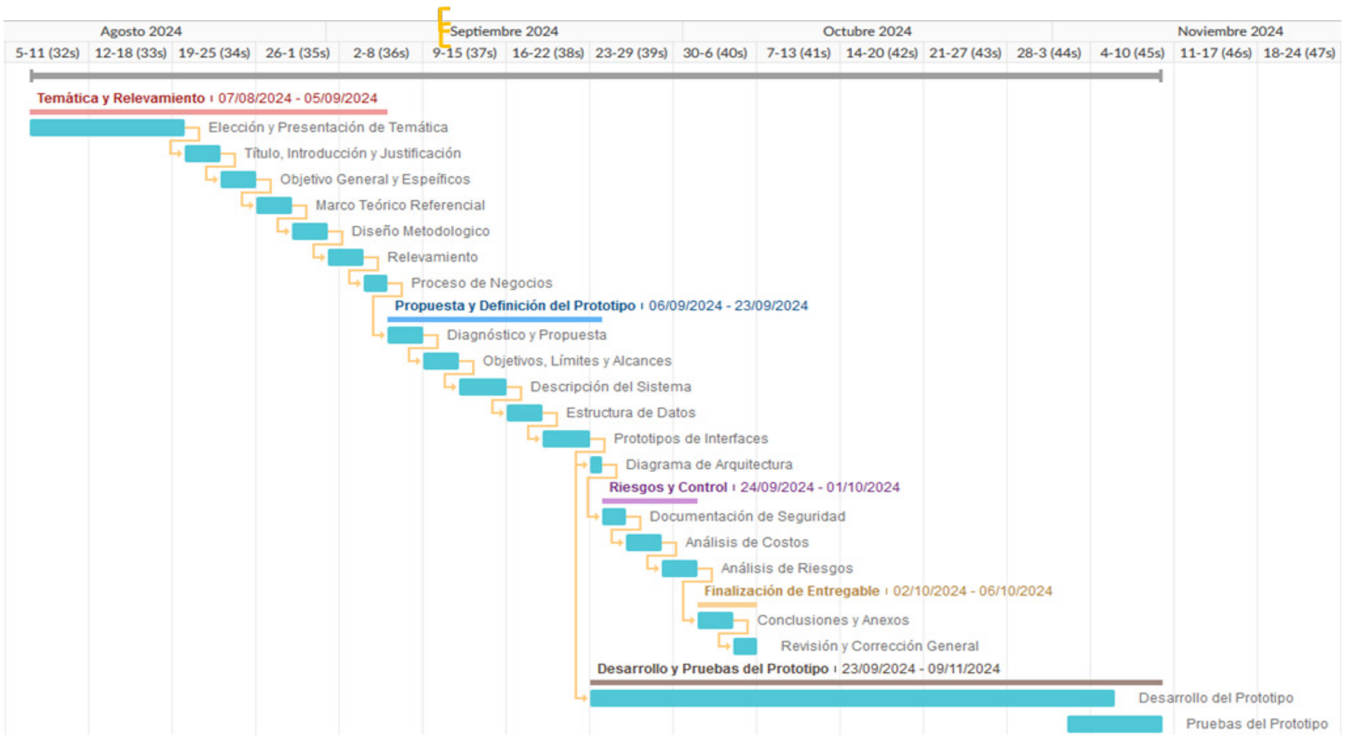


Figure 2. Gantt chart

RESULTS

Survey

Structural Survey

Given that the project developed was a web platform aimed at software developers, both employees and freelancers, it is not possible to establish a specific location, as this aspect depends on each individual user who accesses the platform.

However, using the data collection methods specified above, it has been identified that most developers use their own or their employer’s laptops and, to a lesser extent, desktop computers to carry out their professional tasks in the IT field.

Functional Survey

Hierarchical Structure

Below is a generic organizational chart of a small or medium-sized software development company, given the nature of the users targeted by the project.

The sectors covered by the developed platform are colored.

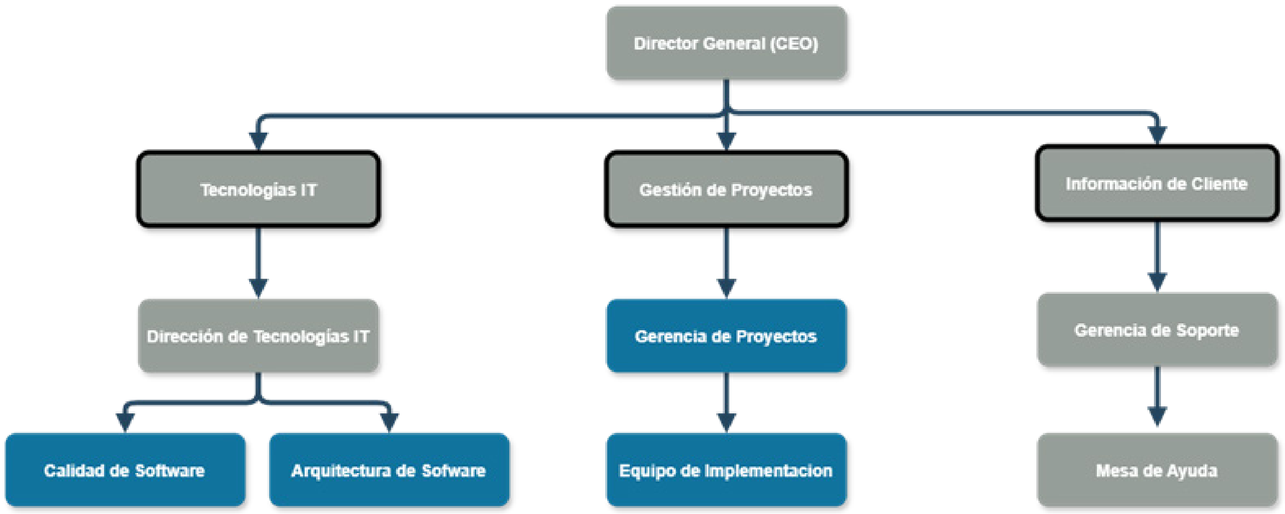


Figure 3. Organizational Chart of a Software Development Company

Functions of the Areas

Software Quality: responsible for ensuring that the software meets the previously specified requirements and that it is free of errors. It also implements tests and continuously monitors product quality throughout the development life cycle.

Software Architecture: aims to define the overall structure of the system, including technologies, design patterns, and interactions between the different services that make up the final product. It also focuses on ensuring scalability and efficiency in the software.

Implementation Team: develops, implements, and maintains the software product. Performs the task of writing code, fixing bugs, and participating in the creation of new features indicated by design requirements and specifications.

Project Management: establishes the resources needed to carry out the project and supervises its execution and development, while assigning the corresponding tasks and defining their deadlines.

Relevant Processes

Process name: Software Quality Control.

Roles: Software Quality, Implementation Team, Software Architecture, Project Management.

Steps: the Software Quality department establishes and documents the quality standards and criteria that software developments must meet, in coordination with the Software Architecture department to ensure technical and design alignment.

The Software Quality team then develops a detailed test plan, which is shared with the Implementation Team to ensure that all functionalities are covered and tested through the specified tests.

Finally, the Software Quality team performs the tests and evaluates the results to provide feedback to the Implementation Team, if necessary. Otherwise, Project Management receives the product and is responsible for its final release.

Process name: Software Architecture Design

Roles: Software Architecture, Software Quality, Project Management.

Steps: Project Management works to understand and document the different requirements of the project in question.

Once the requirements have been defined, the Software Architecture Department designs and documents the architecture in its entirety, including design patterns, frameworks, and technologies. Once this has been done, the results are communicated to the Quality Team.

The Quality Department verifies the quality standards of the developed architecture, inviting adjustments to be made if necessary.

Process name: Software Development.

Roles: Implementation Team, Project Management, Software Architecture, Software Quality.

Steps: Project Management distributes tasks and assigns resources to the Implementation Team according to the schedule and priorities assigned to each project task.

The Implementation Team develops the functionalities according to the architectural design and detailed specifications provided by the Software Architecture department, establishing constant communication to ensure understanding and compliance with the guidelines.

Before sending the developed modules to Software Quality, the Implementation Team performs internal tests to detect and correct errors. Once resolved, the complete modules are sent to the aforementioned department, which will proceed with formal testing.

Any defects identified by the Quality Team are reported in a test report to the Implementation Team, who make the necessary corrections. This step is a cycle that ends when the specified quality standards are considered to have been met.

Documentation Survey

The following documents are surveyed:

- Quality Standards Specification: document describing the quality criteria and standards that software developments must meet as defined by the Software Quality team.
- Test Plan: document detailing the test plan, including test types, test cases, and assigned resources. This plan is developed by the Software Quality team.
- Requirements Survey: document listing the requirements gathered by the Software Architecture team in collaboration with Project Management. It is the basis for the design of the software architecture.
- Architecture Documentation: document that describes in detail the software architecture, including diagrams of components, interactions, and technologies used.
- Test Report: document generated by the Software Quality team summarizing the results of the tests performed, including defects found and corrective actions.

Business Process

The flowchart below shows the comprehensive process carried out when developing a software product.

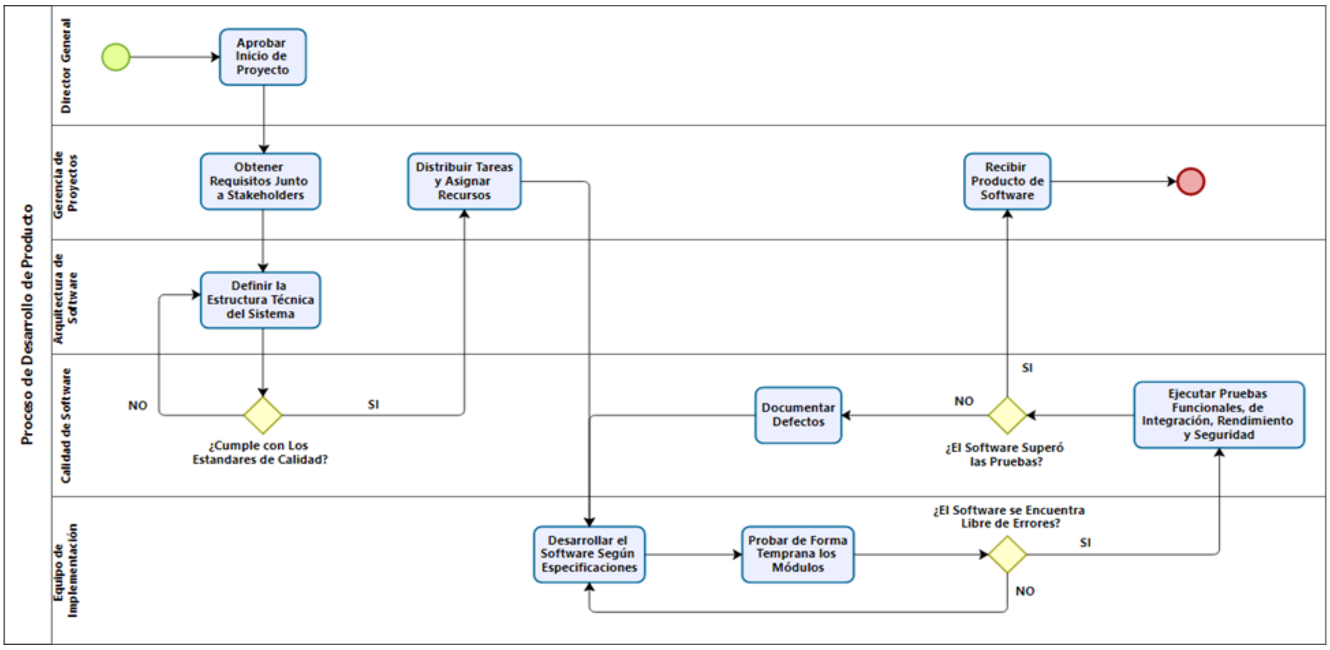

Proceso de Desarrollo de Producto
Director General
Gerencia de Proyectos
Arquitectura de Software
Calidad de Software
Equipo de Implementación
Aprobar Inicio de Proyecto
Obtener Requisitos Junto a Stakeholders
Distribuir Tareas y Asignar Recursos
Recibir Producto de Software
Definir la Estructura Técnica del Sistema
NO
SI
¿Cumple con Los Estandares de Calidad?
Documentar Defectos
NO
SI
¿El Software Superó las Pruebas?
Ejecutar Pruebas Funcionales, de Integración, Rendimiento y Seguridad
¿El Software se Encuentra Libre de Errores?
Desarrollar el Software Según Especificaciones
Probar de Forma Temprana los Módulos
SI
NO

Figure 4. Flowchart referring to Product Development

Diagnosis and Proposal

Table 1. Diagnosis of the Software Quality Control Process

Process Name: Software Quality Control	
Problems	**Causes**
Dependence on manual testing.	Presence of incorrectly decoupled components, making it impossible to automate isolated component tests. Poor documentation or lack thereof regarding the composition and internal logic of the components belonging to the software product.

Table 2. Process diagnosis: Software Architecture Design

Process Name: Software Architecture Design	
Problems	**Causes**
Difficulty adapting the architecture to new project requirements.	Changes in requirements can cause the original architecture, or most of it, to become obsolete or unable to meet emerging needs. The presence of a monolithic or poorly modular architecture limits flexibility to adapt specific parts of the system without affecting others. Lack of documentation of the or inaccessibility to it.
Lack of coordination in defining the architecture between teams.	The different departments are not always properly aligned with architectural decisions.

Table 3. Software Development Process Diagnosis

Process Name: Software Development	
Problems	**Causes**
Overloading of the development team due to lack of software reuse.	No clear strategy or a centralized repository for managing reusable components.
Late identification of errors in the code before quality delivery.	Due to a lack of continuous testing resources, errors are often detected late in the development cycle.

Proposal

The developed system improved efficiency and clarity in the design and development of software components. It allows users to create highly decoupled component architectures and obtain the code for each component with centralized, accessible, and clear documentation. This facilitates coordination between teams and ensures technical alignment in all phases of the project. The platform incorporates automatic testing from the start of component creation, significantly reducing dependence on manual testing and improving testing and quality processes. In addition, the logical decoupling of components encourages reuse, accelerating development and improving the adaptability of the resulting architectures.

Objectives, Limits, and Scope of the Prototype

Prototype Objectives

Develop a prototype system that allows the design of distributed architectures through the combination of predefined components, automatically generating code tailored to user specifications for subsequent download.

Limitations

From the configuration and combination of predefined components to the automatic generation and download of the corresponding code according to user specifications.

Scope

The processes included in the technological prototype are listed below:

- User registration and login.
- Architecture design through the combination of microservices.
- Microservice composition management.
- User guides.
- Dynamic component generation.
- Generation of technical documentation.
- Validation and testing of the generated architecture.
- Downloading the resulting architecture.

CONCLUSIONS

Based on the identification of a recurring pattern in the development of microservice applications, namely that services with similar functionalities and logic are often programmed, a system was conceived, designed, and implemented with the aim of offering an alternative that improves development productivity, providing multiple benefits for the developer and, consequently, for the client and/or employer.

The project resulted in a web platform with a simple and intuitive interface, built with multiple technologies and based on templates for dynamic code generation. This tool provided users with the ability to design distributed microservice architectures based on common components by dragging and connecting pieces on an interactive canvas. Thousands of lines of code can be generated easily, backed by adequate documentation. By downloading the code resulting from their design onto the canvas, users obtain fully compatible components with correct configurations and smooth, error-free communication between services.

Throughout the development of the system and the preparation of this document, I had the opportunity to apply and deepen the knowledge I acquired during my career, learning in a practical way at each stage of the process, from project conception to final documentation. This work also allowed me to face the real challenge of building a system from scratch, combining different technologies in a coherent and functional way. In this process, I not only consolidated what I had learned, but also understood the complexities involved in achieving a robust and efficient solution, and the satisfaction that comes from seeing a final software product that was initially nothing more than an idea.

BIBLIOGRAPHIC REFERENCES

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/

FINANCING
None

CONFLICT OF INTEREST
The authors declare that there is no conflict of interest.

AUTHORSHIP CONTRIBUTION
Conceptualization: Tomás Darquier, Pablo Alejandro Virgolini.
Data curation: Tomás Darquier, Pablo Alejandro Virgolini.
Formal analysis: Tomás Darquier, Pablo Alejandro Virgolini.
Research: Tomás Darquier, Pablo Alejandro Virgolini.
Methodology: Tomás Darquier, Pablo Alejandro Virgolini.
Project management: Tomás Darquier, Pablo Alejandro Virgolini.
Resources: Tomás Darquier, Pablo Alejandro Virgolini.
Software: Tomás Darquier, Pablo Alejandro Virgolini.
Supervision: Tomás Darquier, Pablo Alejandro Virgolini.
Validation: Tomás Darquier, Pablo Alejandro Virgolini.
Visualization: Tomás Darquier, Pablo Alejandro Virgolini.
Writing - original draft: Tomás Darquier, Pablo Alejandro Virgolini.
Writing - review and editing: Tomás Darquier, Pablo Alejandro Virgolini.