Automating the Generation of Microservice Architectures in Web Applications

Pavel-Cristian CRACIUN[1]
[1]Bucharest University of Economic Studies
craciunpavel18@stud.ase.ro

The advent of microservice designs, which prioritizes enhancing deployment timelines, scalability, and flexibility, marks an advancement period in software development. This article presents a tool designed to accelerate the construction of microservice architectures. Using an intuitive interface, the solution allows users to create fundamental code and graphically construct structures, which streamlines the typically laborious first coding process. Through the automation of project documentation and scaffolding, the solution reduces resource consumption and speeds up development. The proposed solution's complete approach is demonstrated by a comparison with Spring Initializr, which provides a straight path from conceptual design to deployable code. This highlights the potential of the proposed tool to revolutionize software project development.
Keywords: *Microservices, Software Development Efficiency, Automated Code Generation, Architectural Planning*

1 Introduction

The evolution of web applications demands scalable and flexible architectural solutions to handle the complexities of modern software needs. Microservices are an architectural and organisational approach as much as a technological one, having the goal of accelerating software application deployment cycles, encouraging creativity, ownership and improving a software applications maintainability and scalability. Also, productivity, agility resilience are methods that are the result factor of this shift in software development. This type of architecture represents a foundation of scalable and agile systems which supports flexibility in integrating new features, meeting dynamic demands. A microservice architecture's ability to distinguish services allows developers to concentrate on distinct components, which promotes innovation and makes maintenance easier.

The structured process used for planning, creating, and deploying an information system is SDLC (Software Development Lifecycle) this is compounded from distinct phases such as:

- Requirement analysis: identifying and documenting what is required by stakeholders.
- Design: defining the architecture of the software based on the demanded requirements.
- Implementation: Coding based on the system designed.
- Testing: verifying that the software meets all the requirements.
- Deployment: delivering the software to be used by the users.

The current paper focuses on the Design phase from the SDLC that represents how the architecture will be built.

2 From monolithic to microservice architecture

There are two models that can be approached to have a consistent architecture, these are: Monolith and Microservices. The Monolith represents a standalone architecture that highlights a linear process in which the whole logic of the application and services are dependent one of another [1]. From a management point of view is a less need for coordination between different

teams, as everything can be centralized in one place. Everything is in a single codebase thus making the testing process easier to approach. for the Deployment process is straightforward since involves only one build and one release process.

There are certain limitations and other aspects that Monolith architecture highlights:

- Scalability issues, where a notable study by G. Blinowski, et al., underscores the debate surrounding scalability in monolithic vs. microservice architectures. Their investigation into the performance dynamics on a single machine highlighted potential scalability challenges when migrating to microservices, especially for smaller businesses with limited concurrent users, posing implications for scalability strategies [2].
- Complexity in making Changes, which spotlights the revitalization of software design through microservices comes as a response to the rigidity found in monolithic architectures, where the complexity of implementing changes can stifle innovation and prolong development cycles. This is substantiated by the work of Milos Milic, et al., who developed a quality-based model emphasizing the cumbersome nature of adapting monolithic systems to evolving software needs [3].
- Slow Start-Up Times, the agility afforded by microservices directly addresses the sluggish responsiveness endemic to monolithic architectures during initialization phases. The comparison provided by Nahid Nawal Nayim, et al., evaluates performance implications in an e-commerce startup setting, shedding light on the substantial lead time reductions achievable with microservice infrastructures [4].

The addressed limitations of monolithic architectures in the design phase in the SDLC advocate for using microservices instead. Microservices architecture involves designing a system as a collection of smaller independent applications, each performing a specific business function.

Microservice architectures represents an advancement in the design and development of web applications [5].

It is evident that the transition from monolithic to microservices architecture is not merely a technical upgrade but a strategic necessity to overcome profound operational challenges [1]. This type of architectural paradigm named Microservices decomposes applications into small, independently, and deployable services with each of their own having their unique business functionality. Terdal’s work on microservices exemplifies this approach, highlighting the transition of e-commerce web applications from monolithic to microservice architecture, emphasizing the benefits of independent development, testing, and deployment facilitated by modern technologies like Docker containers and Kubernetes [6].

The architectural approach of microservices also offers notable financial advantages, particularly in terms of cost-effectiveness when developing large-scale systems [7]. The perspectives of Nada Salaheddin Elgheriani et. al. [8] highlight even more of the many advantages that microservice designs offer, including improved agility, developer productivity, resilience, scalability, dependability, maintainability, and ease of deployment. All these qualities work together to create a strong structure that can handle future demands in addition to meeting company development's present needs. Considering the above listed academic achievements, software development processes might be greatly streamlined by automating the creation and implementation of microservice architectures. Tools like as "Mono2Micro"

from IBM [9], which were put out by A. Kalia et al., show how AI may help ease the shift from monolithic to microservice architectures by providing an automated and effective way forward [10]. Developers may quickly prototype and implement services by including automatic code generation approaches, as covered by Gaetanino Paolone et al. [11]. Although, the solution represents more of an aid in transforming existing codebases and primary focuses on migration rather than the initial creation of microservices from scratch.

The proposed tool offers a novel solution by enabling the rapid setup and deployment of microservices, streamlining the development cycle, and reducing the time and resources traditionally required for architectural planning and documentation. Essentially, the revolutionary potential of automating the creation and implementation of microservice architectures is fully captured by this suggested approach. It holds the potential to redefine the standard for software development, particularly in terms of project conception and execution. As we continue to explore this novel technique, our research delves into the theoretical foundations, real-world applications, and potential impacts of the proposed web application on the effectiveness, scalability, and resilience of web application development frameworks. This investigation aims to illuminate how the proposed solution can fundamentally enhance the architectural landscape of software engineering, promising a transformative shift in how developers approach project development.

3 Efficient Automation of Microservices in Web Applications

In contrast, this paper proposes a solution designed to pioneer the development of microservice architectures from the ground up. As a web application with a user-friendly UI, the solution proposes a solution for creating and interconnecting microservices through the interface. Also, another important objective would be to generate a comprehensive foundational project that includes all necessary dependencies and microservices code. This tool reduces the need for extensive design and planning meetings, allowing developers to directly construct and visualize their architecture through an intuitive UI diagram. Each microservice can be customized based on the goal that it serves and can be documented directly in the interface, where notes and details can be added to enhance clarity and maintainability. This approach not only streamlines the development process but also ensures that all aspects of the system architecture are well-documented from the outset, significantly enhancing the efficiency and effectiveness of project development.

The approach is characterized as empirical research aimed at identifying ways to automate microservices. It was intended to thoroughly analyse the automation of microservices architectures in comparison to with other existing tool, Spring Initialzr. Various strategies were employed to focus on analyse and demonstrate the efficiency of automation in the microservice architecture development process. The research was conducted with the aim of identifying, testing, and documenting the most effective practices to reduce time and resources in the software development lifecycle. This involved a combination of theoretical study and practical application to assess the impact of automation on project efficiency and error minimization.

3.1 Beyond existing solutions

The proposed solution represents a new step in the software development by simplifying the creation of microservice architectures. Through an intuitive and friendly user-interface, developers can visualize and design their system using a diagrammatic approach, which fosters a clear visual understanding of service interconnectivity. The application streamlines the process of

setting up microservices by allowing users to add dependencies directly within the UI, thus eliminating the traditional back-and-forth between planning and execution. Going beyond mere diagramming, the proposed tool generates the corresponding microservices code, complete with specified dependencies, thus significantly accelerating the development workflow.

There are multiple solutions that share the same purpose as this paper such as:

- Jhipster [12], a development platform, that generates, develops, and deploys Spring Boot web applications and Spring Microservices.
- Telosys [13], a lightweight code generator capable of creating various types of applications (Web, CLI, etc.) from a simple model defined with text files or database schema.While flexible, Telosys might not offer as much depth in handling complex microservices architectures.
- Jmix [14], an instrument that provides high-level tools for enterprise development with Spring Boot. Specialised in rapid development capabilities but might not offer the same level of microservices-focused functionalities.

Spring Initializr [15] was specifically chosen for comparison due to its widespread recognition and focused utility in bootstrapping Spring-based projects efficiently, making it highly relevant to developers familiar with Spring ecosystems. This context helps highlight the proposed solution's enhancements over traditional methods like those provided by Spring Initializr. A straightforward through the Internet interface is offered by Spring Initializr to create Spring-based projects using pre-made templates. Through the creation of build files and project structures, it provides developers with a rapid start. However, as Table 1 illustrates, its usefulness, is limited to the initial setup and requires human implementation of the comprehensive architectural design and inter-service communication.

Conversely, the proposed solution encompasses a wider range of functions by not just starting projects but also streamlining the complete architecture design process. The suggested approach emphasises the complete perspective of the system architecture by enabling the thorough designing of interconnected services, in contrast to Spring Initializr, which mainly provides project scaffolding. It bridges the gap between initial setup and full-fledged architectural development by providing teams with a more integrated solution.

Table 1. Comparison Solutions Table

Feature	Spring Initialzr	Proposed Solution
Primary Function	Bootstrapping Spring-based projects	Comprehensive microservice architecture creation
User Interface	Simple web-based interface	Intuitive and diagrammatic user interface
Automation Level	Generates basic project structures and build files	Automates microservice code and dependency management

Developing strong software architectures requires meticulous attention to detail, starting with the creation of diagrams and documentation of architectural designs and each microservice's functionality. Research indicates that these preliminary tasks are naturally slow paced and resource-intensive, even though they are essential for comprehending and growing applications [16][17]. Developers expend substantial effort not only in conceptualizing the structure and interaction of services but also in translating these conceptualizations into actionable code which represents difficult, redundant procedure that demands for creativity. While, Spring Initialzr provides a solid foundation for initiating Spring-based

projects, the proposed solution, extends far beyond the initial setup. It integrates the entire design and development, saves time and enhances quality which makes it superior.

3.2 Web prototype of the proposed solution

The solution proposed is designed to assist development teams, to avoid the usual obstacles related to the architectural foundation stage. Conventional methods, such as lengthy design and planning meetings with iterative cycles to finalize an architecture, often delay project kick-offs. But these chores would become much simpler when the offered solution would be used. As illustrated in Figure 1, the solution does more than just visualisation of the project architecture: it also converts the visualisation into deployable code. This guarantees that teams leave the design and planning session with more than just conceptual schematics. Instead, they have a workable code foundation that can be expanded upon. The tool is essentially removing the need for fundamental work by automating the creation of microservices infrastructure, freeing up teams to focus on product development and business logic execution. Leaping ahead of the early setup stages results in increased productivity, a shorter time-to-market, and more efficient use of resources.

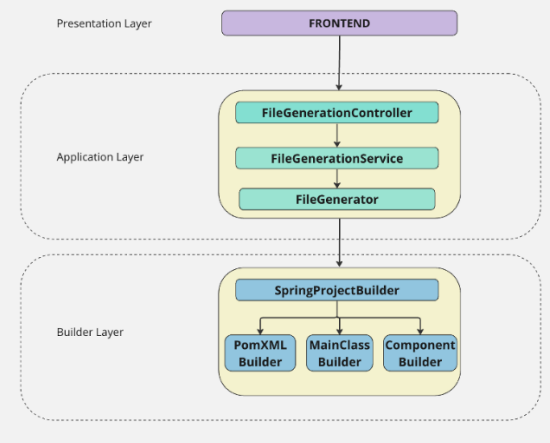


Fig. 1. Solution Architecture

The application is built using Java 17 and employs a microservices architecture. One of the microservices is designed to process JSON inputs, storing templates that represent snippets of code such as classes, records, and interfaces. These templates are then overlaid with additional templates, such as annotations, and transformed into strings. These strings are subsequently modified with parameters derived from the JSON input, utilizing a builder pattern to construct these values and inject them into the codebase. This process allows for dynamic code generation based on the HTTP Request parameters.

Upon accessing the suggested application, developers are prompted to create a new project. Also, the user will have the possibility to edit in the future the project that was created.

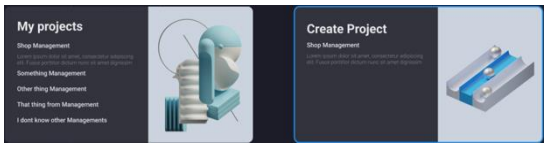


Fig. 2. Main Menu

What can be seen in Figure 2 is an approach of the application to set the stage for a customized microservice architecture. Upon project creation, users will articulate essential metadata, including project's title, the preferred programming language, and an enumeration of global dependencies that will underpin the subsequent architecture. This initial stage is critical as it establishes the project's core framework. Ensuring that a sensible technological and structural baseline is followed by all upcoming microservices.

After creation, the user is redirected to a dashboard that takes the stage for a future placeholder of the microservices that will be connected through diagramming. This represents a centralized core and an orchestrator of the project, which grants the user an oversight over their microservice ecosystem. As shown in Figure 3, the dashboard provides sections for adding new microservices or templates. This is the place where the design, management and

configurational utilities come together to provide users with the resources they need so that at the end could have a fully functional project. As new microservices are created, users can attach detailed notes to each one, documenting specific functionalities and architectural decisions. This ensures that each microservice's role and configuration are clearly understood and maintained through the project lifecycle.

Fig. 3. Dashboard diagram

The "Add Service" option on the dashboard takes users to a **customised design area** where they may access the microservice architecture's granularity. Illustrated in Figure 4, users have the freedom to choose from creating a service from scratch, fully customized by them or to choose from an existing template such as Login Service, User Details service and many more.

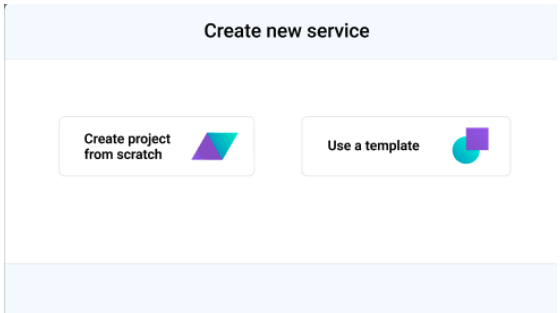


Fig. 4. Create service

These templates are services that are usually present in every application. If the users select to create a new service, the application will grant them freedom to determine the characteristics of the services, such as programming language, nomenclature, and whether to build a custom entity from scratch or instantiate the service from a desired template. Because of its adaptability, every microservice guaranteed to be both precisely functional and to be in harmony with the project's overall architectural philosophy.

One other example of the proposed solution's strength is its ability to specify deep services. As a component of a broader architectural process, every microservice requires careful description of classes, methods, and their interactions. This is made possible by the user-friendly, diagrammatic interface of the suggested solution, displayed in Figure 5, which gives developers the ability to see and control the operational dynamics of any service component.

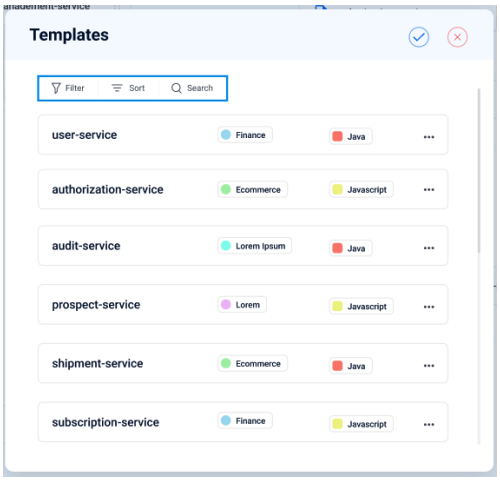


Fig. 5. Create template

One important aspect that the tool provides is the possibility to incorporate dependencies for each microservice, ensuring that each has the necessary tools and libraries for optimal functionality, as seen in Figure 6.

With the proposed solution, developers may examine the complete project layout after a thorough architecture has been built and documented. Users can iterate and revise their design during this comprehensive review phase, which is crucial for assuring alignment with project objectives and functional needs. The suggested tool stands

out for being able to provide a codebase that can be used. This essential feature eliminates the need for manual coding by converting the graphically represented architecture straight into an organised, deployable project structure that includes all necessary microservices and dependencies.

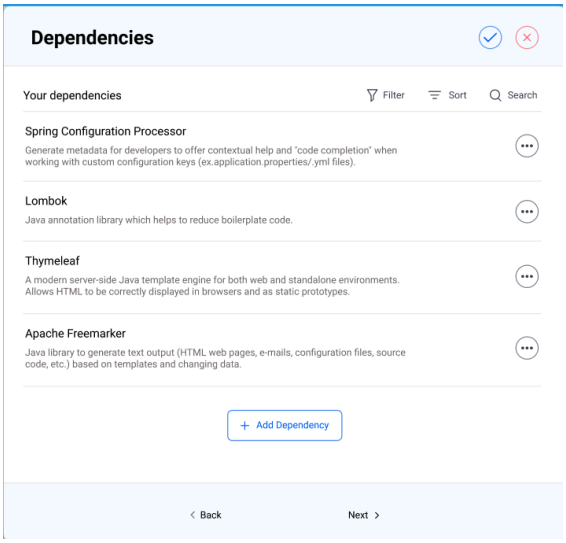


Fig. 6. Add dependencies

At the end, the tool delivers the users a fully deployable code that will only need focus on writing the business logic of the application/product. Through this integration of design, documentation and automated code generation, development teams will help other teams of management and business analysts to understand and have a broader view on the technical landscape which will reversely make the development team be more focused on the product and business side.

4 Conclusion

All things considered, the proposed solution rethinks the architectural foundation of microservice-based apps, transforming a labour-intensive procedure into one that is automated, effective and minimises errors. Looking through the prism of studies that highlights how time- and resources-consuming traditional design methods are, the suggested solution becomes more than simply a tool, and it's a paradigm change. By removing the barrier that separates design from implementation, it enables development teams to jump straight to the process of integrating business logic into precisely created, thoroughly documented code bases. For future enhancements, the proposed solution aims to further boost efficiency and reduce costs by incorporating additional automation across the Software Development Lifecycle, including:

- Infrastructure Automation: Enhancing the solution to automate infrastructure generation could significantly improve system portability through advanced containerization techniques. This would align with modern development practices and address scalability and deployment challenges [18].
- Monitoring Automation: Extending automation to include the monitoring of previously created microservices. This would facilitate proactive management and maintenance, ensuring high availability and performance consistency [19].

In contrast to Spring Initialzr, which mainly facilitates project initialization, the offered solution advances the concepts of agility, speed, and accuracy that form the foundation of contemporary software engineering, therefore capturing the spirit of innovation in software development. The proposed solution facilitated the structured creation of microservices architectures, providing extended functionalities for visualization and code generation. These elements have significantly enhanced the efficiency of the project development process, which confirms the achievement of the established objectives.

References

[1] Kalske, M., Mäkitalo, N., Mikkonen, T. (2018). Challenges When Moving from Monolith to Microservice Architecture. In: Garrigós, I.,

Wimmer, M. (eds) Current Trends in Web Engineering. ICWE 2017. Lecture Notes in Computer Science, vol 10544. Springer, Cham. https://doi.org/10.1007/978-3-319-74433-9_3

[2] G. Blinowski, A. Ojdowska and A. Przybyłek, "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation," in IEEE Access, vol. 10, pp. 20357-20374, 2022, doi: https://doi.org/10.1109/ACCESS.2022.3152803.

[3] Milić, Miloš, and Dragana Makajić-Nikolić. 2022. "Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures" *Symmetry* 14, no. 9: 1824. https://doi.org/10.3390/sym14091824

[4] N. N. Nayim, A. Karmakar, M. R. Ahmed, M. Saifuddin and M. H. Kabir, "Performance Evaluation of Monolithic and Microservice Architecture for an E-commerce Startup," *2023 26th International Conference on Computer and Information Technology (ICCIT)*, Cox's Bazar, Bangladesh, 2023, pp. 1-5, doi: https://doi.org/10.1109/ICCIT60459.2023.10441241.

[5] A. Jan, "What is Microservices Architecture," Sumerge, 09-Nov-2020. [Online]. Available: https://www.sumerge.com/what-is-microservices-architecture/.

[6] Dr. Sujata Terdal, Prasad R G, Vikas Mahajan, Vishal S K, "Microservices Enabled E-Commerce Web Application", International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538 Volume 10 Issue VII July 2022, doi: https://doi.org/10.22214/ijraset.2022.45791

[7] I. Papakonstantinou, S. Kalafatidis and L. Mamatas, "A Techno-Economic Assessment of Microservices," 2020 16th International Conference on Network and Service Management (CNSM), Izmir, Turkey, 2020, pp. 1-5, doi: 10.23919/CNSM50824.2020.9269114.

[8] Nada Salaheddin ELGHERIANI, Nuredin D Ali Salem AHME, "MICROSERVICES VS. MONOLITHIC ARCHITECTURES [THE DIFFERENTIAL STRUCTURE BETWEEN TWO ARCHITECTURES]", *MINAR International Journal of Applied Sciences and Technology, 2022, doi:* https://doi.org/10.47832/2717-8234.12.47

[9] https://www.ibm.com/products/cloud-pak-for-applications/mono2micro

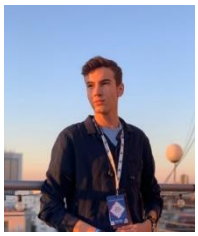
[10] Anup K. Kalia, Jin Xiao, Chen Lin, Saurabh Sinha, John Rofrano, Maja Vukovic, and Debasish Banerjee. 2020. Mono2Micro: an AI-based toolchain for evolving monolithic enterprise applications to a microservice architecture. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). Association for Computing Machinery, New York, NY, USA, 1606–1610. https://doi.org/10.1145/3368089.3417933

[11] Paolone, Gaetanino, Martina Marinelli, Romolo Paesani, and Paolino Di Felice. 2020. "Automatic Code Generation of MVC Web Applications" *Computers* 9, no. 3: 56. https://doi.org/10.3390/computers9030056

[12] https://www.jhipster.tech

[13] https://www.telosys.org

[14] https://www.jmix.io
[15] https://start.spring.io
[16] Przemysław Jatkiewicz and Szymon Okrój. 2023. Differences in performance, scalability, and cost of using microservice and monolithic architecture. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23). Association for Computing Machinery, New York, NY, USA, 1038–1041. https://doi.org/10.1145/3555776.3578725
[17] R. Mishra, N. Jaiswal, R. Prakash and P. N. Barwal, "Transition from Monolithic to Microservices Architecture: Need and proposed pipeline," 2022 International Conference on Futuristic Technologies (INCOFT), Belgaum, India, 2022, pp. 1-6, doi: 10.1109/INCOFT55651.2022.10094556. keywords:
[18] P.-C. Craciun and S.-C. Necula, "Theoretical and Applied in Automating Kubernetes Resources", Informatica Economică vol. 27, no. 2/2023, 2023, pp. [36-45], doi: https://doi.org/10.24818/issn14531305/27.2.2023.04
[19] D. Boncea, M. Zamfiroiu, and I. Bacivarov, " A scalable architecture for automated monitoring of microservices", Economy Informatics vol. 18, no. 1/2018, 2018, pp. [13-22].



Pavel-Cristian Craciun completed his undergraduate studies at the Faculty of Economic Computation and Economic Cybernetics in 2021 and graduated from the master's program in Informatics Systems for the Management of Economic Resources at the Academy of Economic Studies. He is currently enrolled as a first-year PhD student in Economic Informatics. Professionally, he excels as a DevOps engineer, focusing on developing, maintaining, and monitoring software applications.