PD.TVZ.HR
POLYTECHNIC & DESIGN Vol. 12, No. 3, 2024.
Mrežna inačica: ISSN 2459-6302
Tiskana inačica: ISSN 1849-1995
DOI: 10.19279/TVZ.PD.2024-12-3-17

USPOREDBA MONOLITNIH MVC I MIKROSERVISNIH ARHITEKTURA APLIKACIJA U LARAVELU

THE COMPARISON OF MONOLITHIC MVC AND MICROSERVICES ARCHITECTURES IN LARAVEL APPLICATIONS

Sanja Brekalo, Tomislav Sedlarević

Međimurje Polytechnic in Čakovec, Bana Josipa Jelačića 21, 40000 Čakovec

SAŽETAK

Rad analizira i uspoređuje performanse monolitne MVC i mikroservisne arhitekture u razvoju aplikacija koristeći Laravel okvir. Istraživanje uključuje evaluaciju ključnih čimbenika kao što su skalabilnost, performanse, održavanje i otpornost na greške. Rad se temelji na postavljenoj hipotezi da mikroservisna arhitektura pruža značajne prednosti u pogledu skalabilnosti i otpornosti na greške, dok monolitna arhitektura ostaje prikladnija za manje, jednostavnije projekte. Originalni doprinos rada očituje se u usporedbi ovih arhitektura u kontekstu Laravel okvira, s naglaskom na praktične smjernice koje pomažu u odabiru optimalnog pristupa razvoju aplikacija. Kroz detaljne analize i primjere, rad ističe prednosti mikroservisne arhitekture u distribuiranim sustavima, poput neovisnosti pojedinih komponenti i mogućnosti horizontalnog skaliranja. Rad ukazuje na izazove implementacije mikroservisa, uključujući povećanu tehničku kompleksnost i potrebu za detaljnijom dokumentacijom. S druge strane, monolitne aplikacije pokazale su se pogodnijima za manje projekte zbog jednostavnosti razvoja i testiranja. Zaključno, rad pruža jasne preporuke za odabir optimalne arhitekture, naglašavajući važnost prilagodbe specifičnim potrebama projekta te dugoročnoj održivosti aplikacija. Rezultati istraživanja pružaju smjernice za odabir optimalne arhitekture ovisno o specifičnim potrebama aplikacije, uzimajući u obzir troškove održavanja i tehničke zahtjeve.

Ključne riječi: *monolitna arhitektura, mikroservisi, Laravel, skalabilnost, performanse*

ABSTRACT

This paper analyses and compares the performance of monolithic MVC and microservices architectures in developing applications using the Laravel framework. The research evaluates key factors such as scalability, performance, maintainability, and fault tolerance. The study is based on the hypothesis that microservices architecture provides significant advantages in terms of scalability and fault tolerance, while monolithic architecture remains more suitable for smaller, simpler projects. The original contribution of this study lies in the comparison of these architectures within the context of the Laravel framework, with an emphasis on practical guidelines to assist in choosing the optimal approach to application development. Through detailed analysis and examples, the study highlights the advantages of microservices architecture in distributed systems, such as the independence of individual components and the ability for horizontal scaling. The study highlights the challenges of implementing microservices, including increased technical complexity and the need for more detailed documentation. On the other hand, monolithic applications are found to be more suitable for smaller projects due to their simplicity in development and testing. In conclusion, the study provides clear recommendations for selecting the optimal architecture, emphasizing the importance of adapting to the specific needs of the project and ensuring the long-term sustainability of applications. The research results provide guidelines for choosing the optimal architecture depending on the specific application needs, considering maintenance costs and technical requirements.

Keywords: *monolithic architecture, microservices, Laravel, scalability, performance*

1. UVOD
1. INTRODUCTION

The aim of the paper is to investigate and compare monolithic MVC architecture and microservice architecture when developing web applications using the Laravel framework. Considering the growing trend of moving from monolithic to distributed microservice architectures [1], the purpose of the research is to determine which architecture provides better performance, scalability and ease of maintenance in different phases of application development. The paper examines how the choice of architecture can affect the long-term sustainability of applications and the optimization of resources, taking into account the costs and complexity of implementation.

Previous research has provided insight into the general advantages and disadvantages of microservices architecture, such as better scaling and fault tolerance, but also highlights challenges related to complexity and requirements for technical expertise ekspertizom [2] [3]. Monolithic applications are still a popular choice for small and medium-sized projects due to their simplicity and ease of management [4]. Most of the research does not provide concrete recommendations for the choice of architecture in the context of Laravel applications, which opens up space for more detailed research on this topic.

The basic hypothesis of this research is that based on the analysis of the features of both architectures, guidelines can be developed that clearly define their optimal use. A monolithic architecture will prove more suitable for simpler projects with smaller requirements and in the early stages of development, while a microservice architecture will be a better choice for larger, complex systems that require high scalability, flexibility and fault tolerance. This hypothesis will be confirmed through the analysis of advantages and disadvantages of each architecture in different stages of development and specific usage scenarios, providing clear guidelines for choosing the optimal architecture depending on the characteristics of the project.

Laravel is one of the most popular PHP frameworks today, providing a set of tools and resources for building applications based on monolithic and microservice architectures. With the development of this programming framework, numerous possibilities have been designed for simple configuration and creation of projects, enabling scalability, rapid development and secure authentication of users. The advantages of using the Laravel programming framework would be in a highly scalable programming framework, the ability to quickly develop applications, a simple and understandable syntax, robust authentication, extensive libraries and available documentation [5].

The choice of application architecture plays a key role in the long-term success of the project. A monolithic architecture allows the application to be developed as a single unit, while a microservice architecture divides the application into smaller, independent parts that communicate via APIs. These two approaches offer different advantages and challenges, and the choice depends on the specific requirements of the project.

This paper will thoroughly compare monolithic and microservice architectures in the context of Laravel applications, analysing their strengths, weaknesses and suitability for different types of projects.

2. IZRADA LARAVEL APLIKACIJA
2. DEVELOPMENT OF A LARAVEL APPLICATION

The recommended way to create a Laravel project is through Composer commands, which require local installation of a supported version of PHP [6], [7] and Composer [8]. Composer is a tool for managing project dependencies in PHP. It is inspired by the NPM registry as a public database for JavaScript programs and Bundler which is a Ruby library management tool. It enables declaration of dependencies, finds the versions that need to be installed and installs them, and updates all dependencies with a single command [9]. Creating a project through Composer also creates a database. The basic settings of the Composer command create a SQLite database. SQLite is a lightweight, self-contained database that stores data in a single file, making it

practical for development purposes and smaller applications [10].

After creating a project through Composer, the web application can be viewed through a Laravel local development server powered by Artisan. PHP Artisan Serve is an embedded PHP development server that runs a Laravel application. Artisan enables automation of code generation, database migration, dependency management, testing and other administrative tasks. [11]

A Laravel project can also be created using Laravel Sail and Docker. Laravel Sail is a command-line interface for interacting with the Docker development environment [12]. Docker is a platform that makes it possible to create, distribute and run applications in isolated environments called containers. Containers contain all the necessary components to run an application, allowing for consistency across environments. Docker containers allow applications to be portable and run consistently on any system [13]. Containers are very often used in microservices application architecture to ensure their independence from the operating environment.

Laravel Breeze is a starter package for authentication in Laravel applications [14]. Laravel Breeze is often used in monolithic Laravel applications to quickly and securely set up user authentication.

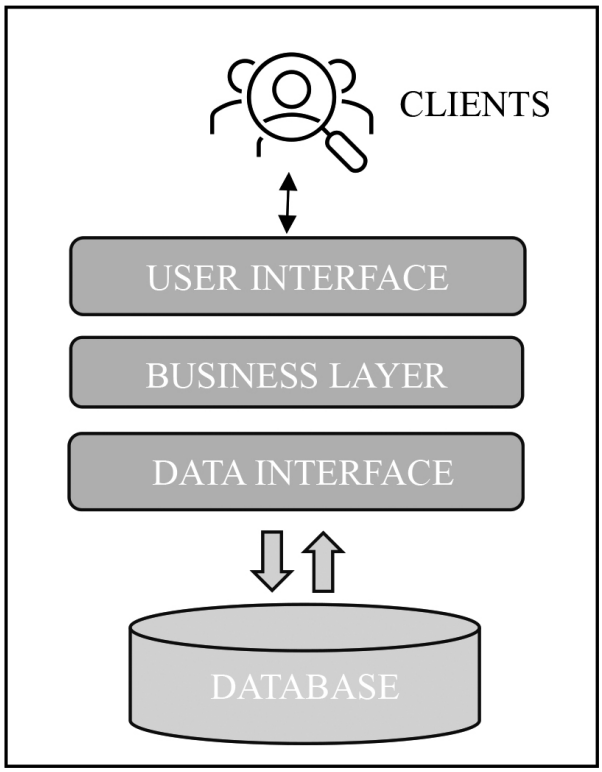
Among other Laravel plugins that can be used in both monolithic and microservice architectures, it is important to mention Blade Templates. Blade is a templating system used within a Laravel application to render HTML. Vite is a development server that will automatically compile CSS and refresh the browser when changes are made to Blade templates [15].

Laravel also includes Eloquent, a tool for mapping objects and relations (ORM - Object-Relational Mapping) that enables the mapping of data from the database to objects in the programming language, thus facilitating work with databases. Eloquent simplifies working with the database. Eloquent allows each database table to have its own "Model" to interact with [16]. Migrations in Laravel make it easy to create and modify database tables, ensuring a consistent database structure across environments. They work like database versioning, allowing the database definition to be shared.[17]

2.1. MONOLITNA MVC ARHITEKTURA U LARAVELU

2.1. MONOLITHIC MVC ARCHITECTURE IN LARAVEL

Monolithic architectures represent the traditional approach to application development as one large unit. They are characterized by one central database and an interconnected complex structure that can ultimately represent challenges arising from the development and maintenance of large and complex applications. Figure 1 shows a monolithic architecture with different application layers.

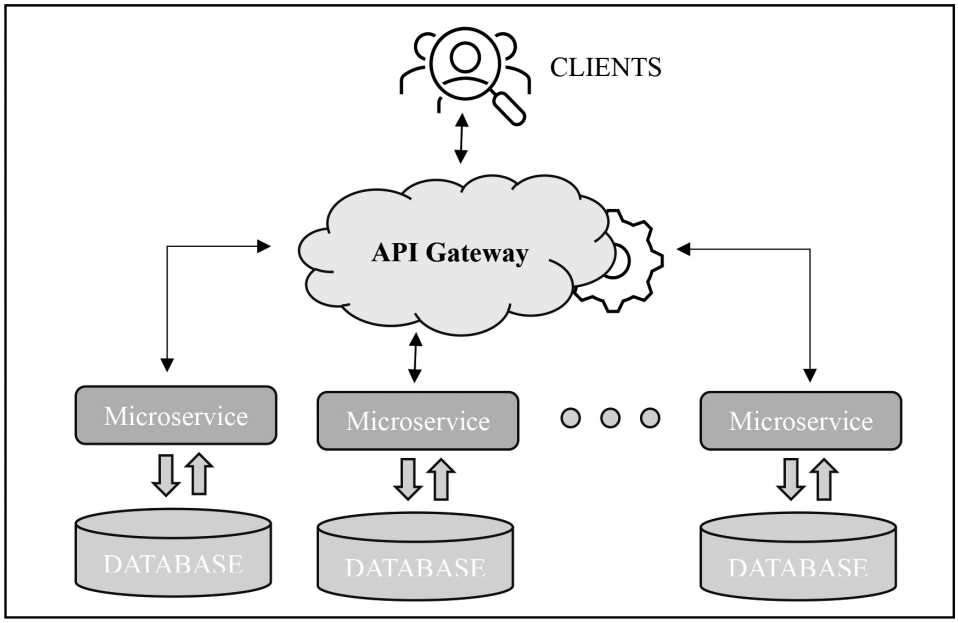


Slika 1 *Monolitna arhitektura aplikacije*

Figure 1 *Monolithic Architecture of the Application*

MVC (Model-View-Controller) is an architectural pattern that separates the application into three interconnected components. Each letter in MVC has its own meaning and function:

1. M-Model is the component that manages data,



Slika 2 *Mikroservisna arhitektura aplikacije*

Figure 2 *Microservices Architecture of the Application*

business logic and application rules. It represents the data and business layer of the application.

2. V-View: View is a component responsible for displaying data to users, it is used to display data from the model in the user interface.

3. C-Controller: The controller is a component that acts as an intermediary between the model and the view. The controller handles user input, processes requests, and returns appropriate responses to display to the user.

2.2. MIKROSERVISNA ARHITEKTURA
2.2. MICROSERVICES ARCHITECTURE

Microservices are a way of structuring network applications that are built using small, independent units. A microservice is an architectural style in which an application is built on top of separate components. Services are connected via API (Application Programming Interface) as REST (Representational State Transfer) or other methods. An API is a set of rules and protocols that enable different software applications to communicate with each other. An API defines the ways in which applications can send requests and receive responses, allowing them to exchange data and functionality without needing direct access to the code or internal details of another application). REST is an architectural style for the design of network services that enables communication between a client and a server via the HTTP protocol. REST is based on using standard HTTP methods such as GET, POST, PUT and DELETE to interact with resources, which are usually represented as URLs. [18] Figure 2 shows an example of how the microservice architecture of the application works.

Microservice architecture is a way of designing software systems in which services are responsible for part of the functionality. There is no standard, so basically a microservice can perform complex tasks maintained by the whole team or it can be a service that performs some trivial tasks.

Microservices in Laravel are often narrowly targeted, allowing them to be smaller and run faster. Laravel microservices are most often accessed through an API so that the user calls the API gateway URL, which then authorizes the user and calls the corresponding service. The API gateway maintains a service repository, a singleton class that contains service metadata and host names, which helps determine which service needs to be invoked.

There are different ways to create microservices. Various development frameworks, programming languages and data storage systems can be combined. One way to create microservices is to use the Lumen project created in Laravel. There

are other frameworks (e.g., Slim, Silex) similar to Lumen that can be used for microservices. Lumen is a good choice for creating microservices because it is fast and contains all the features of Laravel. Lumen is a micro-framework with specific capabilities in the development of microservices in Laravel. A new Lumen project can be created using the Composer command. It's easy to scale, easy to maintain, and good for building APIs. [19]

3. USPOREDBA PREDNOSTI I NEDOSTATKA ARHITEKTURA
3. COMPARISON OF THE ADVANTAGES AND DISADVANTAGES OF ARCHITECTURES

The different architectures used in the creation of network applications have their advantages, disadvantages and recommended use cases. In this paper, monolithic and microservice architectures are compared with different key aspects that can influence the choice of architecture when creating a network application.

3.1. RAZVOJ APLIKACIJE
3.1. APPLICATION DEVELOPMENT

A monolithic application is easier to build and maintain in the early stages of development because all components are in one place, which simplifies development, communication between application parts, and upgrades. Centralization of resources (database, cache, API, processes) facilitates performance monitoring and simplifies infrastructure upgrades. It is simpler for small projects, but becomes more complex as the application grows.

Microservices architecture enables faster development as teams can work on separate components in parallel or integrate ready-made services from external third parties, thus increasing productivity. Although microservices initially bring greater complexity, they make it easier to later manage large systems. The disadvantage is more complex interconnection and code complexity.

3.2. PERFORMANSE
3.2. PERFORMANCE

As an application gets bigger, the number of users and requests can cause performance degradation in a monolithic application because all components share the same resources. Scaling a monolithic application usually requires adding more resources to the same server (vertical scalability). The use of microservices provides a key advantage in increasing system performance as it enables easier management and scaling of the application to cope with increased demands. Microservices can be optimized independently, which increases performance in contrast to monolithic applications where it is more difficult to optimize the performance of large applications.

3.3. SKALIRANJE
3.3. SCALING

Scaling an application corresponds to an increase in the volume of work, users or data. Microservice architecture enables independent scaling of services, which is a highly scalable approach. In a Laravel application, different parts (e.g., authentication, user system) can be separated into microservices and scaled according to needs. If one part, such as authentication, requires more resources, only that service is scaled, without affecting the rest of the system, thus increasing stability. Each microservice can use different technologies and be optimized for a specific task, which provides greater flexibility in architectural decisions.

Horizontal scalability allows adding microservice instances instead of increasing server power, thereby distributing the load. A key advantage of microservices is to scale only the parts under traffic pressure instead of having to scale the entire application (as would be the case with monolithic applications) thus reducing costs. This kind of architecture is ideal for cloud environments, where scaling can be automated using orchestration tools like Kubernetes or Docker Swarm, which allows for easy addition or removal of microservice instances according to current needs. [20]

3.4. OTPORNOST NA GREŠKE I KVAROVE
3.4. ERRORS AND FAILURES RESISTANCE

Each microservice operates independently, so if one microservice fails due to a bug, performance issue, or code error, the other microservices can still run smoothly. In a microservice architecture, a failure in one microservice will not necessarily cause the entire application to crash, making the system more resilient to individual failures. If one microservice fails, additional instances can be quickly added. This is a significant advantage over monolithic applications, where a failure in one part of the application could affect the entire system.

3.5. OTKLANJANJE POGREŠAKA APLIKACIJE
3.5. DEBUGGING THE APPLICATION

Debugging microservices is challenging due to their distributed nature, lack of centralized data logging, and increased complexity.

3.6. TESTIRANJE APLIKACIJE
3.6. APPLICATION TESTING

With monolithic applications, testing the entire application is simpler, while microservice architectures makes testing parts simpler.

3.7. ODRŽAVANJE I AŽURIRANJE APLIKACIJE
3.7. APPLICATION MAINTENANCE AND UPDATING

Microservice architectures are advantageous in cases where each component requires separate updates and releases or the application needs to adapt to rapid changes. Each service is independent and can be modified without affecting the overall functioning of the application. Updating is easier because microservices are smaller in code size, so updating can be performed for separate code parts. With monolithic applications, it is more difficult to make changes because a change in any one part of the code can affect the entire application. If one part of the application faces a problem, it can affect the entire application. As the application gets bigger, it becomes more difficult to maintain and implement changes, because each change can affect the entire system.

3.8. TEHNIČKA EKSPERTIZA
3.8. TECHNICAL EXPERTISE

Microservice applications require greater technical expertise, more documentation for all services, and knowledge of the Laravel framework requires additional technical knowledge about microservices.

3.9. ODABIR TEHNOLOGIJA
3.9. TECHNOLOGY SELECTION

With monolithic applications, restrictions are typically set on the use of the same technology for the entire system, while with microservice architecture, there is a greater flexibility in selecting technology for each microservice.

3.10. BAZA PODATAKA
3.10. DATABASE

Monolithic applications often use a single database while microservices typically use separate databases, which can cause additional challenges in data management and consistency between different services. In a microservice architecture, data isolation means that each microservice manages its own data and database, meaning that different microservices do not have direct access to the data of other microservices. This practice ensures the independence of microservices and security and privacy. Each microservice can function independently, without the need to share data with other microservices. Microservices have access to their own data only, which reduces the risk of access to unauthorized information. Isolated data reduces interdependencies between microservices, which enables better scalability and maintainability.

Isolated data can also lead to microservice data inconsistencies due to their distributed nature.

Factors such as network latency and concurrent updates contribute to inconsistencies among microservices, which can affect system reliability and data integrity.

3.11. KOMUNIKACIJA IZMEĐU MIKROSERVISA
3.11. COMMUNICATION BETWEEN MICROSERVICES

Communication between application components differs significantly between monolithic and microservice architectures. In a monolithic architecture, communication between modules takes place within the application itself, resulting in lower latency and simpler implementation. On the other hand, microservice architecture relies on network communication (e.g., via REST API or gRPC) which introduces additional complexity, latency and the need to achieve network security. Although the microservice approach allows for greater flexibility and module independence, it requires the implementation of additional mechanisms for performance monitoring, authentication, and error handling. This difference in communication mode can have a significant impact on system performance and complexity, depending on project requirements.

3.12. POPREČNA PITANJA
3.12. CROSS-CUTTING CONCERNS

Examples of cross-cutting concerns include security (authentication and authorization), logging (logging), error handling, caching, performance monitoring, and transactions, which must be implemented consistently throughout the application. In monolithic applications, Laravel usually centralizes them through middleware or service providers. In a monolithic architecture, it is easier to manage these issues because everything is located in one place and within a single framework.

On the other hand, in a microservice architecture with Laravel, cross-cutting issues become more complex because each microservice can be a separate entity that requires its own implementation of these functionalities. For example, each microservice must provide consistent logging, error handling, and security measures, which can increase system complexity. In order to facilitate cross-question management in microservices, external tools such as centralized logging systems (e.g., ELK stack) or API gateways for security and authentication are often used.

In either case, proper cross-issue management is critical to maintaining application scalability, security, and reliability.

3.13. MOGUĆNOSTI IMPLEMENTACIJE MODERNIH TEHNOLOGIJA
3.13. POSSIBILITIES OF IMPLEMENTING MODERN TECHNOLOGIES

As part of the implementation of modern trends in application development, technologies such as serverless architecture and edge computing are becoming increasingly important. Serverless solutions, such as AWS Lambda, enable the execution of code without the need for infrastructure management, with automatic horizontal scaling of the application according to needs. Edge computing provides the ability to process data closer to end users, reducing latency and increasing performance. Locally processed data remain closer to the user, which reduces the risk of misuse. At the same time, the amount of data sent over the Internet is reduced, thus saving resources and reducing costs. In order to improve the performance of the application in the context of Laravel, it is possible to use tools like Octana, which can greatly speed up the work and ensure high responsiveness of the application. In Laravel projects, it is possible to incorporate support for event-driven architecture, which can improve the scalability and efficiency of task processing. In terms of using new technologies, the integration of GraphQL is also possible as an alternative to traditional REST APIs due to better control and flexibility in retrieving data.

Artificial intelligence (AI), blockchain and distributed systems will play an increasingly important role in the future. AI brings adaptability to applications, enables a personalized user experience, automates routine tasks, can optimize resource usage, predict problems and help systems run faster and more efficiently. Blockchain can provide greater security to microservices thanks to decentralized data storage and encryption. To

keep up with these trends, Laravel may include additional tools to support these technologies in the future, allowing applications to be more adaptable and meet increasingly complex industry requirements.

4. SMJERNICE ZA OLAKŠANI ODABIR OPTIMALNE ARHITEKTURE OVISNO O SPECIFIČNIM POTREBAMA APLIKACIJE

4. GUIDELINES FOR EASIER SELECTION OF OPTIMAL ARCHITECTURE DEPENDING ON SPECIFIC APPLICATION NEEDS

Table 1, which evaluates the advantages of monolithic and microservice architectures, was created based on consideration of the advantages and disadvantages of individual architectures for building applications in Laravel.

5. ZAKLJUČAK

5. CONCLUSION

Based on the conducted research, key conclusions were obtained. A monolithic Laravel application is a better choice for smaller projects or projects in the early stages of development due to simple implementation, lower initial costs and easier maintenance. However, as the application grows, scalability becomes more challenging, and performance may decrease due to the limitations of a monolithic architecture. In such situations, managing the application becomes more complex, and changes in the code can cause problems with the consistency and stability of the system.

Tablica 1 *Procjena prednosti monolitne i mikroservisne arhitekture*

Table 1 *Evaluation of the Advantages of Monolithic and Microservices Architecture*

	Monolitne	Mikroservisne
Development of simple applications	+ easier	
Development of complex applications		+ easier
Application development in teams		+ easier
Integration of ready-made services		+ easier
Connecting parts of the application	+ alleviated	
Overall code complexity	+ lower	
Performance under higher application load		+ better
Application scaling		+ easier
Resistance to errors and malfunctions		+ better
Debugging application development	+ easier	
Polyglot programming (using different programming languages and technologies when creating program code)		+ easier
Choice in the selection of technologies		+ bigger
Testing parts of the application		+ easier
Testing the entire application	+ easier	
Independent supply of parts		+ advantage
Installation of new technologies		+ advantage
Independence of changing parts of the application		+ advantage
Reuse in different applications or pieces of code		+ advantage
Technical expertise required	+ lower	
Data isolation increases security and privacy		+ advantage
Consistency of records in the database	+ advantage	
Communication between pieces of code	+ easier	
Solving cross-cutting issues	+ easier	

On the other hand, Laravel applications based on microservices represent a more scalable option in the long term, especially for larger systems with many users and different business requirements. Although the initial implementation of a microservice architecture is more complex, this architecture provides significant flexibility and better allocation of resources as the application grows. The advantages of microservices stand out in cases where it is necessary to dynamically scale certain system components, update them independently, or when the system has specific industrial and technical requirements that require the use of multiple technologies and programming languages.

From the research, it can be concluded that the choice between monolithic and microservice architecture will largely depend on the specific needs of the project, but also on the experience and technical expertise of the development team. Microservices are especially useful when organizational scaling challenges arise or when an application becomes too large for a team to easily understand and maintain. In such conditions, microservices enable a faster reaction to changes and easier management of resources. The analysis showed that microservice architecture brings significant advantages in terms of fault tolerance and flexibility in managing complex systems, but it should be noted that these advantages come with increased technical requirements and more complex implementation. Although microservices architecture is more cost-effective in the long run for larger projects, monolithic applications have their role in smaller and medium-sized systems and are often more cost-effective for projects that do not require significant scalability.

This paper provides an original contribution to the research of Laravel application architectures, comparing monolithic and microservice approaches, and has offered new guidelines for choosing the optimal architecture, based on the real needs and characteristics of applications. The conclusions are based on a detailed analysis of the specifics of the Laravel framework, which makes the work innovative in the context of existing research.

In order to adapt Laravel applications for future challenges, a modular development approach is recommended, which facilitates the transition from a monolithic to a microservice architecture. When building applications, you should consider using tools like Laravel Octane to improve performance, GraphQL for more advanced API requests, and container technologies like Docker for easier application scaling and management. Adaptability to these technologies can ensure the long-term sustainability and competitiveness of the application in an increasingly dynamic technological environment.

6. REFERENCE

6. REFERENCES

[1.] Razzaq A.; Ghayyur S. A. K.; A systematic mapping study: The new age of software architecture from monolithic to microservice architecture-awareness and challenges; Computer Applications in Engineering Education, svez. 2, pp. 421-451, ožujak 2023.

[2.] Gos K.; Zabierowski W.; Comparison of Monolithic and Microservice Architectures on the Example of a Web Application in Java; 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), Lviv, Ukraine, 2020.

[3.] Freitas F.; Ferreira A.; Cunha J.; Refactoring Java Monoliths into Executable Microservice-Based Applications; 25th Brazilian Symposium on Programming Languages (SBLP 2021), ELECTR NETWORK, 2021.

[4.] Blinowski G.; Ojdowska A.; Przybylek A.; Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation; IEEE Access, pp. 20357-20374, 2022..

[5.] Yadav N.; Rajpoot D. S.; Dhakad S. K.; LARAVEL: A PHP Framework for E-Commerce Website; 2019 Fifth International Conference on Image Information Processing (ICIIP 2019), Waknaghat, India, 2019.

[6.] Supported Versions; The PHP Group, [Mrežno]. Dostupno: https://www.php.net/supported-versions.php. [Pokušaj pristupa 6. 6. 2024.].

[7.] PHP: Hypertext Preprocessor - Downloads; PHP.net, [Mrežno]. Dostupno: https://www.php.net/downloads. [Pokušaj pristupa 06 06 2024].

[8.] Composer; GetComposer.org, [Mrežno]. Dostupno: https://getcomposer.org/. [Pokušaj pristupa 2024. 6. 6.].

[9.] Introduction - Composer; GetComposer.org, [Mrežno]. Dostupno: https://getcomposer.org/doc/00-intro.md. [Pokušaj pristupa 2024. 6. 6.].

[10.] SQLite Home Page; SQLite.org, [Mrežno]. Dostupno: https://sqlite.org/. [Pokušaj pristupa 6. 6. 2024.].

[11.] Artisan Console; Laravel.com, [Mrežno]. Dostupno: https://laravel.com/docs/11.x/artisan. [Pokušaj pristupa 6. 6. 2024.].

[12.] Introduction - Sail; Laravel.com, [Mrežno]. Dostupno: https://laravel.com/docs/11.x/sail. [Pokušaj pristupa 6. 6. 2024.].

[13.] What is a Container?; Docker.com, [Mrežno]. Dostupno: https://www.docker.com/resources/what-container/. [Pokušaj pristupa 6. 6. 2024.].

[14.] Starter Kits - Laravel Breeze; Laravel Documentation, 11.x, [Mrežno]. Dostupno: https://laravel.com/docs/11.x/starter-kits#laravel-breeze. [Pokušaj pristupa 10. 6. 2024.].

[15.] Vite; Laravel Documentation, 11.x, [Mrežno]. Dostupno: https://laravel.com/docs/11.x/vite. [Pokušaj pristupa 10. 6. 2024.].

[16.] Laravel, Eloquent: Getting Started - Laravel 11.x - The PHP Framework For Web Artisans; [Mrežno]. Dostupno: https://laravel.com/docs/11.x/eloquent. [Pokušaj pristupa 11. 07. 2024.].

[17.] Migrations in Laravel 11.x Documentation; Laravel , [Mrežno]. Dostupno: https://laravel.com/docs/11.x/migrations. [Pokušaj pristupa 11. 07. 2024.].

[18.] A. Avdhoot, A Complete Laravel Microservices Guide for Beginners; eLuminous Technologies, 22 08 2024. [Mrežno]. Dostupno: https://eluminoustechnologies.com/blog/laravel-microservices-guide/. [Pokušaj pristupa 02 09 2024].

[19.] Laravel Microservices: Everything You Need to Know; Bacancy Technology, 6 6 2024. [Mrežno]. Dostupno: https://www.bacancytechnology.com/blog/laravel-microservices. [Pokušaj pristupa 4 9 2024].

[20.] R. Powell, Docker Swarm vs. Kubernetes: A Comparison; CircleCI Blog, 08. 04. 2024.. [Mrežno]. Dostupno: https://circleci.com/blog/docker-swarm-vs-kubernetes/. [Pokušaj pristupa 10. 09. 2024.].

AUTORI · *AUTHORS*

• Sanja Brekalo - lecturer at the Međimurje Polytechnic in Čakovec, she received her doctorate from the Faculty of Graphic Arts in Zagreb. She specialized in the fields of web design, programming, multimedia and digital marketing. Her scientific and research interest is focused on the analysis and optimization of software architectures and their application in modern technologies.

Korespondencija · *Correspondence*

sanja.brekalo@gmail.com

• Tomislav Sedlarević - completed his studies in Computer Science at the Međimurje Polytechnic in Čakovec, where he acquired basic knowledge in the field of programming, database and software engineering. He is particularly interested in creating web applications, with an emphasis on working with the Laravel framework, through which he develops advanced skills in the design and implementation of modern application solutions. He is actively involved in the research of new technologies and best practices in the development of software architectures, with a focus on performance optimization and user experience.

Korespondencija · *Correspondence*

tomi.sedla@gmail.com