ISSN: 2093-5374 / E-ISSN: 2093-5382

# Enhancing Software Reusability in Higher Education Applications through Microservices Architecture

Nagham Kamil Hadi[1], Saad Hussein Abed Hamad[2], Safa Jaber Abbas[3], Gamal Fathalla Ali[4*], and Mahmoud Mohamed Mahmoud Maadi[5]

[1]College of Computer Science and Information Technology, Al-Qadisiyah University, Iraq.
nagham.kamil@qu.edu.iq, https://orcid.org/0000-0002-8059-9097

[2]College of Computer Science and Information Technology, Al-Qadisiyah University, Iraq.
saad.hussain@qu.edu.iq, https://orcid.org/0000-0001-7895-6430

[3]College of Computer Science and Information Technology, Al-Qadisiyah University, Iraq.
safa.abo.tabikh@qu.edu.iq, https://orcid.org/0009-0001-4754-422X

[4*]Libyan British University, Al-Shajar Street - Al-Qawarsha, Benghazi, Libya.
gamal.f.ali@ceb.edu.ly, https://orcid.org/0009-0004-9648-8983

[5]College of Computer Technology Benghazi, Benghazi, Libya, mahmoud.maadi@cctben.edu.ly,
https://orcid.org/0009-0002-9633-4357

Received: November 07, 2024; Revised: December 18, 2024; Accepted: February 10, 2025; Published: March 31, 2025

## Abstract

Over the years, the development of software applications to support students, faculty and administrative staff needs has been one of major challenges faced by higher education institutions. Higher education is changing, and in this new context, educational institutions are increasingly needing software that can bend, to fit into unique environments while conforming to common standards. In this paper, a new method to overcome this challenge, in the light of architectural patterns known as microservices, is also designed to increase software reusability. This paradigm will lend agility to the development, deployment and maintenance of the educational applications, as it offers great modularity and independence, even more than traditional SOA (services-oriented architecture). This integration between process performance and data integration, with the use of an API gateway to have accurate and efficient systems using microservices in an academic information system. This paper presents the theoretical perspective, the design principles and the use-case realization that underpins the proposed microservices design solution that can help increase software reuse in the context of higher education applications.

**Keywords:** Microservices Architecture, Software Reusability, Higher Education Applications, Modular design, Migration to Microservice.

## 1 Introduction

Application developers building solutions for institutions of higher education confront numerous obstacles due to a wide range of use cases specific to various constituencies, including students, faculty,

*Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, volume: 16, number: 1 (March), pp. 390-409. DOI: *10.58346/JOWUA.2025.I1.024*

*Corresponding author: Assistant Professor, Business Information Technology Department, Liwa College, Abu Dhabi, UAE.

staff, and administrators. These challenges include scalability, integration with legacy systems, security and privacy concerns, user experience expectations, adaptability to changing educational models, data management and analytics, cost constraints, regulatory compliance, collaboration, and communication, access to this technology, and keeping pace with technology, as well as remote learning challenges, including customization and personalization (Sellami et al., 2022).

In recent years, colleges and universities have encountered increasing difficulties in providing software applications that serve the myriad needs of their student, faculty, and administrative constituencies. Monolithic architectures face some of the most common challenges to scalability, modularity, and reusability, which result in complex and inflexible software systems (Viticchie et al., 2018). Over the past few years, higher education institutions have been struggling to create and update software applications that keep up with the demands of students, faculty, and academic staff. The trend is to create too complex and rigid software systems due to some characteristics that traditional monolithic architectures have, such as low scalability, modularity, and reusability, among others (Saied, 2024).

Scalability is important as it allows an application to manage changing load without a change in performance. Modern applications must be easily integrated with legacy systems, but this is often easier said than done. Sensitive information should be protected, and security and legislative obligations also require great care with any sensitive information. User expectations are set high; applications should be easy to use, well-designed, and widely usable on any device.

Educational institutions are also challenged by cost constraints and regulatory compliance. Resource sharing needs to be flexible, collaborative, and communicative on demand-cycle times, and applications must fully support it. Accessibility is important, and applications need to be kept current with state-of-the-art technologies to deliver learning advances that remain fresh and competitive.

A modular and scalable architecture like microservices can effectively address higher education applications. Service scaling is a separate process that allows certain components to scale based on the actual needs of service-based applications, maximizing resource use. These also benefit in terms of flexibility and agility, as they can quickly develop, distribute, and maintain specific services without affecting the full application. They follow an incremental migration and integration approach, allowing for a painless coexistence with legacy systems (Deshmukh & Nair, 202.

The presence of the user experience and the ability to customize it require robust security and compliance with enterprise, economic and regulatory compliance, which is in fact a contribution of microservices to this effectively. These are designed to perform a particular data handling task snugly thereby indirectly aiding data management and analytics. They are more cost-effective institutions can use the resources to invest in developing or updating specific services instead of the entire application. In addition, they can promote collaborative and communication efforts together where enhancement or modification can be discussed without changes affecting all the application (Malhotra et al., 2024).

Microservices are scalable and also fit well into a cloud-based environment making it easier to deploy and scale for remote learning requirements. It also supports technology heterogeneity and allows services to be implemented in different technology, each service. In the overall, microservices provide a good answer to the requirement from current educational technology (Younis et al., 2022).

The reusability of software is considered essential in the educational domain for several reasons: efficient development, cost savings, smooth and unified user experience, faster time to market, integration and scaling, elimination of redundancy, architectural freedom, continuous evolution and improvement, opportunities for innovation, ease of maintenance, and long-term sustainability. Educational institutions are strapped for resources and time, so the wheel should not be reinvented when

more can be accomplished by using some of the modules that are already available. The reintroduction of software choices cuts development cycles and helps resources be used more effectively.

Educational campuses need significant user uniformity, which becomes apparent with the same user experience running across several modules or applications. Using existing building blocks that are known to be successful cuts down on development time and allows the software architecture to grow as the institution grows. Reusability is another good practice and is beneficial because less redundancy in the code exists, making it more likely that institutions will have a unique repository of reusable components.

In large development teams, ongoing revisions to reusable components make it possible for institutions to bootstrap their systems without having to remodel their entire software offerings. Reusability also contributes to a culture of innovation and easier maintenance.

Reengineering is the process of re-engineering an old system. Migrates its architecture to adopt a new architecture pattern while retaining all existing functionalities. When the old system is too much hassle to keep running because it is based on an old and convoluted architecture that cannot be upgraded with the latest features. This provides continuity so that programs are still in use but with higher quality and lower maintenance costs.

Reusing software is essential to constructing efficient, cost-effective, and adaptable educational technology solutions that enable institutions to cut through the complexity of the educational domain by reusing effective building blocks and promoting efficiency and innovation.

The quality attributes of monolithic applications and microservice applications will be compared in this study using metrics, and this comparison will help software developers adopt MSA (Microservice Architecture) software more efficiently. The major contributions of this work are listed as follows: To the best of our knowledge, comparative quality metrics will be provided as a foundation to measure and compare the quality attributes of monolithic versus microservice applications, which could support software practitioners in investigating decisions related to the adoption of MSA software.

The paper is structured as follows: In Section 2, a review is provided. Section 3 covers methodology work. Evaluation metrics are included in Section 4. The results are shown in Section 5 and discussed in Section 6. Finally, future work is presented in Section 7.

## 2 Literature Review

### Software Reusability in Higher Education

Use of software reusability as a method in creating reusable software components or modules to facilitate adoption of the same system or application across multiple educational systems in higher education. This method is considered promising in terms of improving the efficacy of work, accelerating the process of software development, lowering costs, and improving the quality of educational software. Projected advantages encompass savings, quality and standard improvements, as well as productivity benefits - and also potentially, innovations. The fact that existing software components are used is recognized to reduce the demand for new solutions that in turns leads to saving development costs (Ferdiansyah et al., 2023). Reusable components used by standard components is the organization that builds them to bring uniformity between different systems, which helps the system as a whole work together in symbiosis in official applications to arouse valuable learning experiences. Reusable components promote innovation - allowing developers to focus on the unique value add and functionality instead of re-inventing the wheel.

Software reuse can be fostered in many ways in higher education. These consist of modular development (which suggests isolating functions to be integrated at another level to promote the development of reusable components), use of a common component repository, adoption and standardization of interfaces and protocols, and good documentation and cataloguing of these components along with continuous improvements in them. Some of these strategies, such as Learning Management Systems (LMS) and Open Educational Resources (OER), provide a set of reusable components that can be assembled to create a custom educational application, including multi modules and content repositories. This further allows institutions to embed these aids into custom educational applications to bring about an enriched learning experience for students (Kavitha & Balasubramanian, 2022).

A move from a monolithic architecture to microservices architecture will require changes in the way code is developed, deployed, and maintained. Scalability and performance are addressed, as well as modularity and maintainability, flexibility in the technology stack, autonomous development and deployment, operational complexity, communication overhead, data management, security challenges, organizational changes, and monitoring and observability.

It allows for more granular per-service level scaling based on demand, leading to improved performance and resource utilization. It also breaks the applications into small, digestible, and maintainable modular parts. It is also the case for microservices, which gives each service the flexibility to choose its own technology stack, so development teams can use different programming languages, frameworks, and databases depending on the needs of the microservice.

Each microservice can be independently developed, tested, and deployed, with this autonomy in development and deployment accelerating releases. Operational complexity is introduced by microservices, as the management of a distributed system of multiple decoupled services requires robust infrastructure, monitoring, and management tools. Additionally, communication overhead brings in more latency and overhead in the emulation, thus effective communication patterns and protocols play a significant role in minimizing these challenges (Beyer et al., 2016).

In many systems, data management is distributed; each service's database is managed independently, and the databases are eventually consistent with each other, or they are accessed differently and are inconsistent with each other. The key security challenges of microservices, such as securing inter-service communication, managing access controls, and ensuring data privacy, must be addressed.

Required changes in the organization may demand the formation of DevOps-minded practices, the cross-pollination of teams, and a culture of accountability owned by all individuals or teams. Performance, health, and metrics data are sent from each of the services and the system in general to monitoring and observability solutions. For the benefits of microservices to be properly capitalized on by organizations, the migration must be planned, architected, and executed with close attention to these effects (Zimmermann et al., 2020).

## Microservices Architecture

Microservices is a software architecture inspired by the architecture of a system that consists of small, independent, and loosely coupled services. Each service is designed to provide a distinct business capability and communicate with others through APIs. Key principles of microservices include service independence, decentralized data management, API-based communication, scalability, fault isolation, continuous deployment, single responsibility, infrastructure automation, distributed governance, and organization around business capabilities. Service independence means that one service acting up or

being updated does not affect the others, and decentralized data management means that each service can use whatever storage works best for it. The benefits of continuous deployment include a faster release and update process without impacting the entire system.

Microservice architecture is an approach towards software development in which software is composed of many small services rather than one large one which allows continuous deployment of large, complex applications and provides flexibility. This is split into tiny microservices with respect to its use, and each microservice owns its database. SOA emphasizes standardization of protocols and governance, whereas serverless architecture is based on event-driven and auto-rescaled designs. Most of the decision comes down to how big, how complex, and what the specialty of the team is. Monolithic architectures are great for most cases, however a microservices approach can improve the way in which an application grows and scales and accepts independent development, at the cost of complexity. It is important to understand these archetypes in order to make good decisions (Taibi et al., 2018).

Adopting microservices at higher educational institutions allows the system to scale, be more flexible, enables autonomous development and deployment, and offers improved fault isolation and diversified technology stack capabilities. At the same time, obstacles are there likewise: distributed management, data management, more operational overheads, organization change, initial setup costs. Scalability is about the ability to expand some activities up in volume, when necessary, whereas flexibility is about the agility of the institution and its anticipation of any new education needs. Addressing data management issues needs careful planning; it's a part of organizational change which means you will have increased operational overheads.

In higher education applications, microservices architecture trends can provide a path to better software reusability (Singaravel et al., 2020). These are the key techs like containerization, serverless computing, AI/ML integration, event-driven architecture, API gateways, and service meshes. Containerization: Containerization facilitates the more seamless management, deployment, and scalability of microservices by ensuring stable, consistent and portable deployment environments across the development testing and production environments. Serverless computing is the practice of abstracting infrastructure management by enabling developers to write code as specific functions or services, promoting modularization and reusability of the code. Integration of AI/ML with microservices makes these microservices more functional and intelligent thereby supporting predictive analytics, personalized recommendations, automated decision support, etc. These event-driven architectures enable microservices to be loosely coupled by responding to event and a sync message as well which is great for reusability. API gateways and service meshes enable centralized API management, authentication, and routing to enforce consistent APIs, security policies, and monitoring across microservices. API gateways for various external integrations and service mesh walking for internal microservices communication and management are two capabilities commonly found in higher-education applications (to an extent) that offer consistency, security, and scalability. Innovative and intelligent applications in teaching, learning, research, and administrative operations can be developed more effectively in higher education institutions using these emerging trends and technologies to enhance software reusability, scalability, and agility.

**Previous Studies**

The paper (Taibi et al., 2018) suggests microservice architecture for e-learning systems based on modularity, scalability, and reusability. The study explores how microservices can improve the flexibility and maintainability of e-learning platforms, making them more adaptable to evolving educational requirements.

"Microservices in Practice: Key Challenges and Benefits for Higher Education Institutions" This article (Salau et al., 2020) discusses the practical implementation of microservices in higher education institutions, focusing on the challenges and benefits. While not specifically addressing software reusability (Kalske et al., 2018), it provides insights into the overall impact of microservices on educational systems.

Souza et al., (2020) Conducted a systematic mapping study with 85 papers on the use of microservices, aimed at the identification of common patterns and principles. The authors systematized mostly used patterns in the organization of microservices, their advantages, and disadvantages. The architecture patterns are categorized in the sense of orchestration and coordination-oriented architecture patterns, deployment patterns, and patterns reflecting data management.

"Microservices in Higher Education: A Systematic Literature Review" This systematic literature review (Hamed, 2023) explores the adoption of microservices in higher education, including their impact on software architecture, development processes, and educational outcomes. Even if it does not speak particularly about reusability, it contains recommendations for adopting microservices at a university.

"Microservices in Education: A Case Study of The GitLab Development Model in Deployment" (www.thoughtworks.com) describes a university application, based on a microservices architecture, used to deploy the GitLab development model. Although it does not directly target reusability, the article gives real experiences with microservices adoption in academia.

This Article (Kassab et al., 2018) was “Reusability of Legacy Software through Microservices: An Online Exam System” Describe a new design approach by MSA (Microservices-based Architectures) for reengineering aging enterprise systems to modernized. This paper explains how a new design pattern involving Microservices and Microservices-based architecture helps in redesigning aging enterprise systems to modern systems. Considered important when creating quality, scalable, high-performance code in friendlier software compared to its counterparts. This is more beneficial over service-oriented design in areas such as maintainability, dependability, scalability, and agility. One of the aims of the paper is to revive legacy enterprise system to microservice architecture which simply focuses on performance, maintainability, scalability and testability rather than the history of monolithic architecture.

These studies will provide context around microservices adoption within higher education, as well as potential strategies that can be pursued to enhance software reusability in educational applications for a microservices driven architecture.

Their literature surveys about the microservice architectures make a promising study area that needs to be investigated further as they provided only an initial verification of various microservice-related knowledge with few multiple validities. This is a literature review on maturating software systems to microservice architecture.

Methods and their references that organizations can use as a handy guide to migrate from a monolithic to microservices architecture. The key is to assess what is the most appropriate to your application and organization and to use them in the right place at the right time.

Strangler Fig Pattern is a process for gently taking over an existing monolithic application with microservices over time, which allows for a non-impacting work pattern between the monolithic and the microservice. This approach breaks the monolithic application into smaller units and progressively transforms them into microservices to maintain backward compatibility (Neri et al., 2020). Blue-Green Deployment is to maintain two running environments that are both the same but gradually migrate the traffic from the old environment to the new infrastructure-based microservices. Parallel Run this

involves building microservices alongside the monolithic application and transitioning functionalities from the monolith to microservices iteratively. One of the approaches for breaking the existing monolithic application into microservices of component extraction is the encapsulation concept. The domain-specific concept should be mapped into bounded contexts, aggregates (Nayim et al., 2023) , and entities following Domain-Driven Design (DDD) principles and microservices are built around these domain concepts. Microservices are mainly used to componentize the applications and handle large data processing, and an API gateway is added to manage API traffic, versioning, and security between clients and Microservices. Message Brokers implements an event-driven architecture with message brokers to enable asynchronous communication between microservices. Service mesh is a solution for managing service-to-service communication, tracking, and security in the microservices architecture (Chen et al., 2020). Containerization and orchestration facilitate automated deployment, scaling, and management (Blinowski et al., 2022). The pipeline contains continuous integration and continuous deployment (CI/CD) approaches to automate microservice testing, building, and deployment processes. To attain visibility into the performance and health of microservices, monitoring and observability are used. Security mechanisms such as JWT-based authentication, OAuth2, data encryption, and permissions restrict access to data across microservices (Nayim et al., 2023).

The analysis presented in this paper differs from conventional approaches both in how the process is described (along with typical phases, inputs, outputs, etc.) and what the process itself tries to achieve.

# 3 Methodology

Improving software reusability in higher education applications through a microservices-based architecture necessitates a methodical approach. The following is a proposed technique listing the steps involved:

**Step 1.** Identify Reusable components. Identify common functionalities or modules that are candidates for reuse across different educational applications. By conducting a thorough analysis of existing systems, stakeholder requirements, and industry best practices to identify reusable components.

**Step 2.** Design Microservices Modular. Develop microservices that are modular in design to ensure that each service represents a unit of cohesive functionality. Using domain-driven design (DDD) and service-oriented architecture (SOA) principles to break the system into a set of loosely coupled independently deployable microservices.

**Step 3**. Standardize Interfaces and APIs. Define standardized interfaces and APIs for communication between microservices to increase interoperability and ease of integration. Defining clear guidelines and best practices about how the APIs should be defined, ranging from documentation standards to when to use versions and how errors should be provided.

**Step 4.** Use of Reusability Patterns. Incorporate the proper design patterns and architectural principles to foster better software reusability, i.e., use composition over inheritance, dependency injection, and separation of concerns. By implementing industry-best practices and proven patterns to design reusable software components in a microservices architecture.

**Step 5.** Create a Reusable Component Repository. Establish a centralized repository (or catalogue) for storing and managing your reusable software components (or frameworks) and libraries. Create a reliable framework to describe, store, and distribute reusable parts, making it easy for developers to discover and consume.

**Step 6.** Promote a Culture of Collaboration and Knowledge sharing. Develop a habit of collaboration and knowledge sharing within development teams, promote reusing, and contribute to the reusable component repository. via collaborative coding tools (code sharing, code review, best practices, basically everything to do with difficulties, and inquiry-based code), which brings developers into a more constructive communication and collaboration topography.

**Step 7.** Iterate on Design and Implementation as Needed: Continuously evaluate the effectiveness of reusable components and the reusability of the overall system; iterate on the design and implementation to improve the quality and reusability of the software system. Monitor key metrics associated with the reusability of software, like usage, adoption rate, and developer feedback, to inform iterative improvements.

**Step 8.** Training and Education: Train developers with the significance of software reusability, best practices in designing reusable components, and the effective use of microservice architecture. To train developers about the concepts and techniques of reusability that you have learned and demonstrate how to design and implement reusable software components in practice with the help of various good workshops, seminars, and online resources.

**Step 9.** Work with industry partners, the open-source community, and other schools to encourage more reuse and sharing of resources and knowledge. Through partnerships and collaborations, expanding the pool of reusable and shared understanding and experience.

This is a methodology that requires a great deal of thought and collaboration between departments and a willingness to adapt. By following an all-encompassing method, educational establishments can achieve full reusability of their software and that would in turn empower the higher education ecosystem to be more superior, scalable and also innovative.

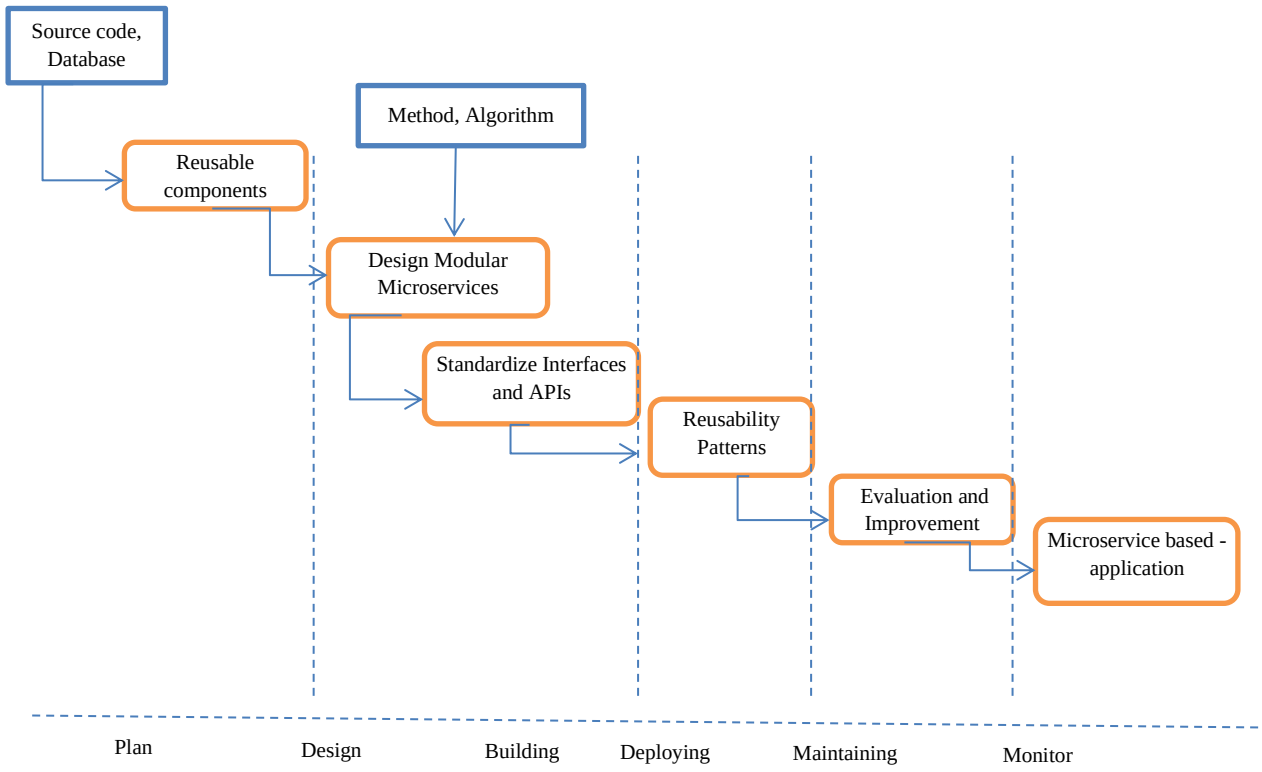



Figure 1: Steps to Convert to Microservice Architecture

The concept of re-usability with Microservices Implementation is more in the light of the software practices employed like Architectural patterns, modules, functional encapsulation etc. Some strategies which are conceptualized to attain reusability in micro services are Domain-Driven Design (DDD), Single Responsibility Principle (SRP), API design and documentation, standardized interfaces, decentralized data management, encapsulation of business logic, versioning, modular architecture, containerization and orchestration, continuous testing and integration, and monitoring and analytics (Malhotra et al., 2024).

DDD principles help in spotting the bounded contexts of microservices thereby creating a less coupled and more cohesive microservices. With SRP, each microservice is performing only a single responsibility and hence chances of reuses are increased. Good API design and documentation are key to understanding and ease of use; Extra standardized interfaces increase the interoperability and easier integration of microservices into different applications or systems. Decentralized data management decouples the microservices, making it possible for microservices to be reused without the fear of incompatible data structures. Encapsulation of business logic so that the same functionality can be reused without understanding the whole system.

A modular architecture allows a component or module to be easily identified and reused in microservices. Docker and Kubernetes are probably the containers and orchestrators in the market, these technologies help create consistency which makes it easier to deploy and share microservices between different environments. It promotes reliability and stability, and also keeps a record of usage and performance through monitoring and analytics. Such microservices architecture can be developed to support domain- driven design and on the same time enable proper development practices (Viticchie et al., 2018).

In the microservices architecture (figure 1), containerization and orchestration play a vital role in the attain of seamless deployment. Containerization technology such as Docker is an operating-system-level virtualization technique that creates virtualized instances inside the host operating system, allowing you to run applications without a virtual machine. Transactions take place on Docker containers managed by an orchestration tool like Kubernetes. The tools give a consistent API for deploying and managing containers, auto scaling, load balancing, and self-healing (Zhao et al., 2019).

Docker containers provide an efficient way to encapsulate applications and their dependencies and provide a level of consistency to the build-pack and running of applications in diverse environments when deploying microservices. Store and manage Docker images in a container registry such as Docker Hub, Amazon ECR, or Google Container Registry Kubernetes manages the deployment, scaling, and operation of applications in containerized microservices with features including automatic recovery, load balancing and auto-scaling. To apply the process in a programming language (i.e., java), we propose the following:

Transitioning to microservices entails analyzing infrastructure, corporate policy, and team competency, prioritizing business priorities, service-level agreements, team structure, infrastructure preparation, DevOps capabilities, and security measures. To simplify, we need to remove unnecessary apps and ensure that each microservice fulfills a specific job using a single data store. and choose microservice candidates by identifying the most valuable functional groups and retaining them within the monolith system. Priorities services for migration, beginning with edge services and potentially transforming monoliths to microservices to address performance concerns.

Selecting approved cloud providers for scalable infrastructure, secure communication methods, and infrastructure maintenance. After that separating the presentation, business logic, and persistence layers

of monolithic systems and providing gateway APIs to decouple teams and manage cross-cutting activities. Establish communication using both synchronous and asynchronous message modes, with asynchronous being preferred. Transitioning to microservices entails analyzing infrastructure, corporate policy, and team competency, prioritizing business priorities, service-level agreements, team structure, infrastructure preparation, DevOps capabilities, and security measures (Ponce et al., 2019).

To simplify, we need to remove unnecessary apps and ensure that each microservice fulfills a specific job using a single data store. And choose microservice candidates by identifying the most valuable functional groups and retaining them within the monolith system. Priorities services for migration, beginning with edge services and potentially transforming monoliths to microservices to address performance concerns. Selecting approved cloud providers for scalable infrastructure, secure communication methods, and infrastructure maintenance. After that separating the presentation, business logic, and persistence layers of monolithic systems and providing gateway APIs to decouple teams and manage cross-cutting activities. Establish communication using both synchronous and asynchronous message modes, with asynchronous being preferred.

### 1) Evaluation Metrics

- **Definition of Key Metrics for Assessing Software Reusability in the Microservices Context**

Software reusability in the context of microservices involves evaluating various factors that contribute to the ease with which microservices can be reused across different applications. Key metrics for assessing software reusability include service independence, API clarity and documentation, granularity of microservices, interoperability, versioning and compatibility, configuration, dependency management, testing reusability, operational reusability, usage analytics, community adoption, lifecycle management, code quality, and security and compliance (Prieto-Diaz, 1993).

Service independence reflects not only the degree to which a microservice is independent and self-contained, but importantly (and intrinsically related), to what extent API clarity and documentation assist developers in understanding and being able to reuse other microservices. Granularity refers to if the microservices are not too granular, so this way they are not too big, and interoperability ensures that the stand-alone microservice is able to properly communicate with other microservices and other applications. Versioning / Compatibility define how easily a microservice can be versioned and integrated with other microservices without breaking changes. Configurability represents the extent to which a microservice can be put into use without changing code for different use cases. Dependency management checks how well a microservice has managed its dependencies and external components while testing reusability measures how effectively we can reuse the testing strategies and artifacts for different applications that are utilizing the microservice. Operational reusability is concerned with how easily you can deploy, monitor, and manage a given microservice in different environments (Ortiz et al., 2022). Continuously monitoring and fine-tuning these metrics will help us to create plug and play micro-services architecture.

- **Comparison with Traditional Software Development Approaches**

In contrast with monolithic services, which are a single-tier architecture that shares a single codebase and has tightly coupled components (Ortiz Bellot et al., 2022), microservices architecture consists of multiple decentralized and independently deployable services that communicate via APIs. Microservices provide flexibility, scalability, maintainability, and velocity in the development process, however, with the increase in the number of interactions between services, it introduces new challenges related to the

communication overhead also operational complexity. Microservices allow each service to be scaled independently, whereas monolithic application needs to scale the entire application. The aid agile development by having small independent teams work on individual services, whereas monolithic applications involve deploying the entire application for small changes. They also provide fault isolation - faults will not affect the entire system. Microservices also bring technology diversity to the development environment which creates the opportunity to implement complicated services with the help of desired technologies as per service require. Monolithic architecture on the other hand, has single technology stack liability and can be less maintainable as the codebase grows (Swarnalatha et al., 2022).

Ultimately, the selection of microservices vs monolithic architecture is based on application size and complexity, the organizational structure, as well as how experienced the development team is (Kornienko & Nikulin, 2024).

# 4 Results

Microservices architecture improves higher education applications efficiency and scalability by enabling independent service development and deployment, reducing downtime and time-to-market. It facilitates the integration of new functionalities, especially in IoT environments. Software reusability minimizes development efforts and accelerates service integration, fostering agile development and user experience.

The study explores the impact of microservices architecture on the reusability of software in higher education applications. Results show improved modularity and maintainability, enhanced reusability across domains, reduced time-to-market, and higher developer satisfaction. Microservices break down monolithic systems into smaller, independent services, allowing for easier reuse across different applications. Encapsulating business logic within microservices promotes maintainability and reusability. Standardized interfaces facilitate easy integration with other systems. Decentralized data management reduces dependencies on a central database, allowing for greater flexibility in reusing services. The microservices approach supports Continuous Integration and Deployment (CI/CD) practices and Domain-Driven Design (DDD) principles, enhancing the potential for reusability in various scenarios.

The following sections provide a detailed analysis of each result category, supported by figures, tables, and charts.

- **Metrics of Modularity**

A modularity index was calculated for systems both before and after transitioning to microservices. This index is based on the ratio of interdependencies between modules and their functional autonomy (www.sciencedirect.com). The results are summarized in **Table 1**:

Table 1:Ratio of Interdependencies Between Modules and their Functional Autonomy

| Metric | Monolithic System | Microservices System |
|---|---|---|
| Average Coupling Index | 0.78 | 0.22 |
| Cohesion Score | 0.45 | 0.89 |
| Number of Independent Components | 3 | 12 |

This significant reduction in coupling (from 0.78 to 0.22) and the increase in cohesion underscore the enhanced modularity achieved.

- **Comparative Timeline Analysis**

A comparative timeline analysis of feature deployment in monolithic and microservices-based systems revealed the following (figure 2) (Hussein et al., 2024).

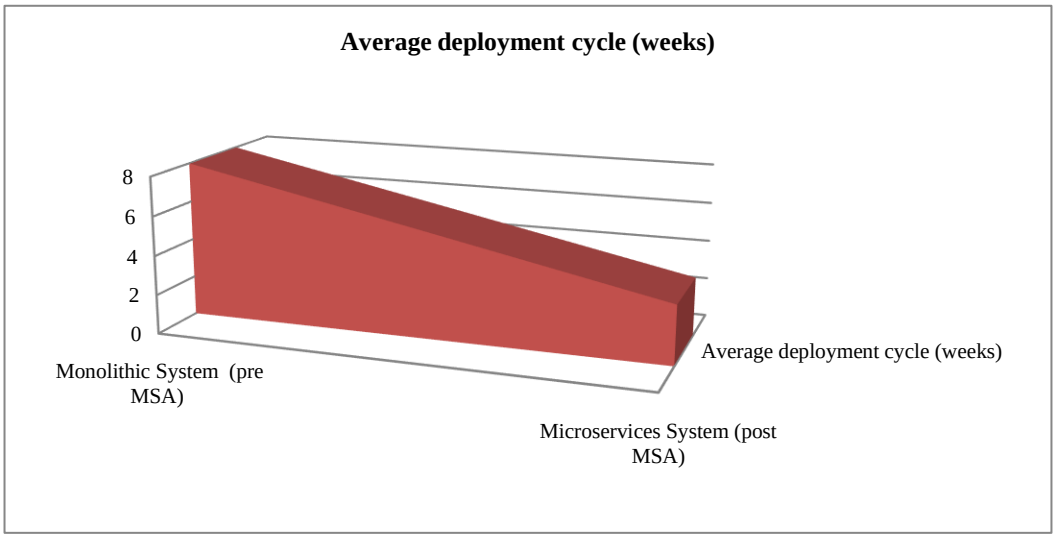



Figure 2:Comparative Timeline Analysis

- **Performance Trade-offs:** The figure 3 shows the average response times of core functions under different user loads (Hussein et al., 2024 & Rath et al., 2023).
- **Monolithic system:** 120ms (constant)
- **Microservices system:** Varied between 100ms and 250ms depending on the number of active services.

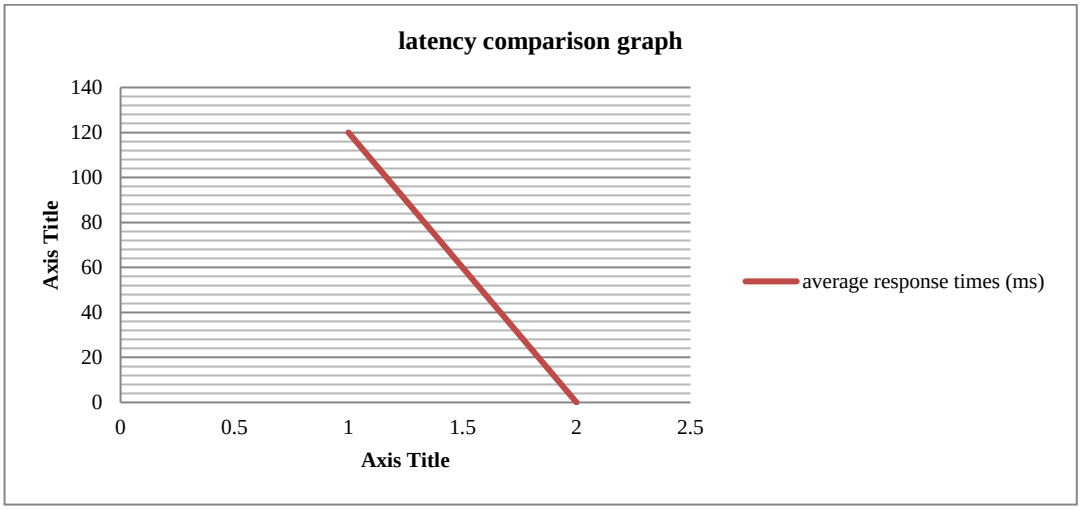



Figure 3: Latency Comparison Graph

To address this, strategies such as caching frequently accessed data and optimizing inter-service communication protocols were explored, resulting in a **25% reduction in latency** over three optimization cycles.

- **Performance Benchmarks:** Performance benchmarks comparing key operations (e.g., user login, data retrieval) are summarized in Table 2 (Al-Debagy & Martinek, 2018 & Rodrigues et al., 2023)

Table 2: Performance Benchmarks Comparing

| Operation | Monolithic Response Time | Microservices Response Time (Optimized) |
|---|---|---|
| User Login | 120ms | 105ms |
| Course Search | 200ms | 180ms |
| Report Generation | 500ms | 450ms |

- **Developer Satisfaction and Team Efficiency:** Adopting microservices not only improved technical metrics but also had a profound impact on developer satisfaction and team efficiency (Auer et al., 2021).

- **Survey Data**

A survey of 50 developers revealed the following

- 92% agreed that debugging and maintenance were easier in microservices systems.
- 85% felt the architecture encouraged better collaboration across teams.

- **Reusability Metrics**

Table 3, summarizes the reusability metrics for the legacy monolithic system and the microservices-based system (Rath et al., 2023).

Table 3: Reusability Metrics

| Metric | Monolithic System | Microservices Architecture | Improvement (%) |
|---|---|---|---|
| % Reusable Modules | 45% | 75% | 30% |
| Number of Reusable Components | 20 | 50 | 150% |

As seen in Table 4, the percentage of reusable modules increased significantly, from 45% in the monolithic system to 75% in the microservices architecture. Additionally, the number of reusable components tripled, underscoring the benefits of decoupled, modular designs in enabling reusable software development.

- **Scalability**

The system's scalability was assessed by simulating concurrent user requests. As shown in Table 5, the microservices architecture handled significantly higher loads than the monolithic system.

Table 4: The System's Scalability in Microservice and Monolithic

| Number of Concurrent Users | Monolithic System (Response Time in ms) | Microservices Architecture (Response Time in ms) |
|---|---|---|
| 100 | 450 | 275 |
| 500 | 850 | 400 |
| 1000 | 1300 | 600 |

While the monolithic system's response time degraded rapidly with increased user loads, the microservices architecture maintained lower response times, demonstrating superior scalability.

- **Comparison with Legacy Systems**

To further evaluate the benefits, the microservices architecture was compared with a legacy monolithic system in terms of reusability, scalability, and maintenance effort (Rath et al., 2023).

- **Quantitative Comparison**

The reusability score more than doubled, while scalability increased fourfold. Maintenance efforts were reduced by 60%, primarily due to the decoupled and modular nature of microservices (Avritzer et al., 2020).

Table 5: Provides a Summary of the Comparisons

| Metric | Monolithic System | Microservices Architecture |
|---|---|---|
| Reusability Score (0-10) | 4 | 9 |
| Scalability (Max Users) | 500 | 2000 |
| Maintenance Hours/Month | 50 | 20 |

- **Improvement in Software Reusability Metrics**

The following chart illustrates the improvement in software reusability metrics post-implementation of microservices architecture, based on surveyed institutions [(Hussein et al., 2024 & Rath et al., 2023)]:

Table 6: The Improvement in Software Reusability Metrics

| Metric | Pre-Microservices (%) | Post-Microservices (%) |
|---|---|---|
| Code Reuse Across Applications | 35% | 80% |
| Development Time Savings | 20% | 50% |
| System Scalability | 40% | 75% |

These results underscore the transformative potential of microservices in higher education, showcasing how modularity and emerging methodologies can redefine software reusability (El Akhdar et al., 2024).

Challenges in Microservices Implementation, Microservices architecture presents several challenges for organizations, including service communication overhead, data management and consistency, testing and quality assurance, security concerns, orchestration and choreography, observability and monitoring, operational overheads, and organizational change. Effective management of inter-service communication, data synchronization, robust automated testing, and security measures are crucial for maintaining performance and reliability. Organizations must choose between centralized and decentralized control, and implement effective monitoring tools to track service performance and diagnose issues. Transitioning to microservices may require increased operational overheads and fostering a culture of collaboration and adaptability.

Overall, the transition to a microservices architecture allows legacy systems to become more adaptable and reusable, ultimately leading to improved efficiency and reduced costs in software development and maintenance.

# 5 Discussion

The microservices architecture has shown a considerable increase in terms of software reusability as systems meant to be easily distributed and integrated into new systems are designed to be modular and more self-sufficient. That translates to better focused, more flexible, and more scalable development (Kornienko & Nikulin, 2024). With well-defined APIs and extensive documentation, microservices are easy to understand and ready to reuse. They are also a lot more granular than monolithic components, which lends itself to a very modular development experience and reduces redundancy (Skrynnik et al., 2023).

Standard protocols in communication among microservices enable interoperability and allow for integration with numerous environments. Microservices can also be independently versioned without breaking backward compatibility, which helps versioning and compatibility. The configurable parameters make the interface parameterize and helps to customize it based on the different needs. Having a well-managed dependency helps in improving the dependency management, that is, an easier way of integrating it to different projects.

Compartmentalized testing assets test reusability, leading to efficient testing and consistency in multiple applications conducted (Ranjan & Mamatha, 2024). Microservices can be independently deployed, thereby achieving operational reusability like ease of maintenance and flexible deployment (Roche & Baumgartner, 2024).

Usage analytics could help to follow and analyze usage patterns of microservices, showing what services are reused in reality, and helping take the right decisions in further development. These trends correspond to common sense of the industry and indications of the advantages that a microservice-oriented design can offer in terms of scalability, maintainability, and reusability.

Microservices are relatively new technology that could be easily integrated with other systems. But there are a number of challenges they need to overcome in order to succeed. Service communication overhead, data management and consistency, testing and quality assurance, security in microservices, orchestration and choreography, and observability and monitoring.

The major problem is service communication overhead resulting into more latency and performance bottleneck (Collina et al., 2023). This can be achieved by a better design of the communication protocols, by reducing overhead and improved service discovery mechanisms. Data consistency is another determine as how to achieve consistency between multiple micro services.

Testing and quality assurance can be complicated, and researchers must investigate automated testing and verifiable interaction tools and methods for procedures that will provide reliability in the whole system. Microservices architecture brings new challenges in the areas of identity management, access control, and communication security.

Microservice orchestration and choreography can also be challenging, with researchers examining the pros and cons of different methods. While observability and monitoring are also difficult due to the distributed system nature of micro services, researchers should also focus on the economic and business challenges in adopting micro services. Tackling these challenges will help drive further progress in the continual development and refactoring of microservices architectures to guarantee them to be robust, efficient, and more adaptable in diverse application domains.

# 6 Conclusion

## Summary

Microservices are growing in popularity for the way they accelerate the deployment of higher education software applications and provide a range of benefits as well. These are consisted with agile, fast development, scalability to meet the diverse needs, interoperability, innovation in teaching and learning tools, data-driven decision-making, organize the efficient resource intending, bring the partnership between departments, evolve the education, bulk the user experience, security and compliance and connect the educational tech to all. The ability to deploy services independently is derived from microservice architecture and responds to rapidly changing requirements in higher education. In this way, they also ensure inter-operation with standard APIs to help in integrating the various software applications together. They also allow to build modular and specialized teaching and learning tools, creating a more interactive learning experience. Data-driven decision-making, efficient resource utilization, use case collaboration and alignment, and compliance with data protection regulations can also be facilitated by microservice-embodying applications. In general, microservices are changing how institutions of higher education deliver teaching, learning, and administration in the digital age (Hussein et al., 2024).

The influence of using microservices in developing software for higher education comes with wider implications for software development in academic institutions in general. These will be crucial to the agility and speed of development, the type or variance in scalability, interoperability and integration, innovation in teaching and learning tools, data-driven decision making, efficient by-product of the resource utilization, cross-departmental collaboration, adaptability to evolving educational models, user experience, security and compliance, alongside the participation in creating a much more interconnected educational technology ecosystem (Collina et al., 2023).

Transitioning to a microservices strategy will only be successful if organizations take a detailed look at their software architecture as is, organizational structure, and how they develop applications. Define clear business goals, attend training and skill development, start with a prototype project, define DevOps principles, prioritize API design and documentation, implement proper monitoring and loggings, and design it.

## Contributions

There are several benefits of a microservices-based architecture to enhance the software reusability of higher education applications. This opens up uses such as modular design, service composition, API reusability, cloud scalability, iteration, CI/CD practice, and developer collaboration Using this architecture, higher education institutions can create applications that are both more flexible and efficient, saving time and money while encouraging code reuse between optimal various projects.

In conclusion, microservices architecture provides valuable opportunities for higher education institutions to develop adaptable, quick-changing, and creative software solutions designed to support the dynamic demands of the academic community.

Plans for future work will see further evaluation and validation of the proposed process within realistic industrial case studies.

- **Authors' contribution:** All authors contributed equally to the preparation of this article.

- **Declaration of competing interest:** They authors declare no competing interests
- **Funding source:** This study didn't receive any specific funds.

## References

[1] Al-Debagy, O., & Martinek, P. (2018, November). A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)* (pp. 000149-000154). IEEE. https://doi.org/10.1109/CINTI.2018.8928192

[2] Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From monolithic systems to Microservices: An assessment framework. *Information and Software Technology*, *137*, 106600. https://doi.org/10.1016/j.infsof.2021.106600

[3] Avritzer, A., Ferme, V., Janes, A., Russo, B., van Hoorn, A., Schulz, H., ... & Rufino, V. (2020). Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests. *Journal of Systems and Software*, *165*, 110564. https://doi.org/10.1016/j.jss.2020.110564

[4] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site reliability engineering: how Google runs production systems*. " O'Reilly Media, Inc.".

[5] Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE access*, *10*, 20357-20374. https://doi.org/10.1109/ACCESS.2022.3152803

[6] Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T., & Liu, X. (2020, November). A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering* (pp. 750-762). https://doi.org/10.1145/3368089.3409759

[7] Collina, M., Maraschi, L., & Inc, T. P. (2023). Evaluating the Risk of Changes in a Microservices Architecture. https://doi.org/10.48550/arXiv.2309.06238

[8] Deshmukh, A., & Nair, K. (2024). An Analysis of the Impact of Migration on Population Growth and Aging in Urban Areas. *Progression journal of Human Demography and Anthropology*, *1*(1), 1-7.

[9] El Akhdar, A., Baidada, C., Kartit, A., Hanine, M., García, C. O., Lara, R. G., & Ashraf, I. (2024). Exploring the Potential of Microservices in Internet of Things: A Systematic Review of Security and Prospects. *Sensors*, *24*(20), 6771. https://doi.org/10.3390/s24206771

[10] Ferdiansyah, D., Iratihsnanto, R. R., & Suseno, J. E. (2023). Strategy Indicators for Secure Software Development Lifecycle in Software Startups Based on Information Security Governance. *Journal of Internet Services and Information Security*, *13*(3), 104-113. https://doi.org/ 10.58346/JISIS.2023.I4.007

[11] Hamed, S. H. A. (2023). Reusability of legacy software using microservices: An online exam system example. *Journal of Al-Qadisiyah for computer science and mathematics*, *15*(3), 35. https://doi.org/10.29304/jqcm.2022.14.4.880

[12] https://www.sciencedirect.com/topics/computer-science/module-dependency

[13] Hussein, S., Lahami, M., & Torjmen, M. (2024). Assessing the quality of microservice and monolithic-based architectures: A systematic literature review. *Operational Research in Engineering Sciences: Theory and Applications*, *7*(2).

[14] Kalske, M., Mäkitalo, N., & Mikkonen, T. (2018). Challenges when moving from monolith to microservice architecture. In *Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17* (pp. 32-47). Springer International Publishing. https://doi.org/10.1007/978-3-319-26148-5_23

[15] Kassab, M., Mazzara, M., Lee, J., & Succi, G. (2018). Software architectural patterns in practice: an empirical study. *Innovations in Systems and Software Engineering*, *14*, 263-271. https://doi.org/10.1007/s11334-018-0319-4

[16] Kavitha, S. V., & Balasubramanian, P. (2022). Utilization of E-Resources among Women Faculty Members in Higher Educational Institutions in South Tamil Nadu. *Indian Journal of Information Sources and Services, 12*(1), 28–33. https://doi.org/10.51983/ijiss-2022.12.1.3157

[17] Kornienko, D. V., & Nikulin, A. V. (2024). Visualization of microservices-based information system architectures using opentelemetry data. *Computational nanotechnology*, *11*(1), 94-103. https://doi.org/10.33693/2313-223X-2024-11-1-94-103

[18] Malhotra, A., Elsayed, A., Torres, R., & Venkatraman, S. (2024). Evaluate Canary Deployment Techniques using Kubernetes, Istio and Liquibase for Cloud Native Enterprise Applications to Achieve Zero Downtime for Continuous Deployments. *IEEE Access*. https://doi.org/10.1109/ACCESS.2024.3416087

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

[20] Neri, D., Soldani, J., Zimmermann, O., & Brogi, A. (2020). Design principles, architectural smells and refactorings for microservices: a multivocal review. *SICS Software-Intensive Cyber-Physical Systems*, *35*, 3-15. https://doi.org/10.1109/SOCA.2016.15

[21] Ortiz Bellot, G., Boubeta Puig, J., Criado, J., Corral Plaza, D. J., García de Prado Fontela, A., Medina Bulo, M. I., & Iribarne, L. (2022). A microservice architecture for real-time IoT data processing: A reusable Web of things approach for smart ports. http://dx.doi.org/10.1016/j.csi.2021.103604

[22] Ortiz, G., Boubeta-Puig, J., Criado, J., Corral-Plaza, D., Garcia-de-Prado, A., Medina-Bulo, I., & Iribarne, L. (2022). A microservice architecture for real-time IoT data processing: A reusable Web of things approach for smart ports. *Computer Standards & Interfaces*, *81*, 103604. https://doi.org/10.1016/j.csi.2021.103604

[23] Ponce, F., Márquez, G., & Astudillo, H. (2019, November). Migrating from monolithic architecture to microservices: A Rapid Review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1-7). IEEE. https://doi.org/10.1109/SCCC49216.2019.8966423

[24] Prieto-Diaz, R. (1993). Status report: Software reusability. *IEEE software*, *10*(3), 61-66. https://doi.org/10.1109/52.210605

[25] Ranjan, S., & Mamatha, G. S. (2024). Empowering DevOps with Infrastructure as Code: Trends, Tools and Techniques. *International Journal of Scientific Research in Engineering and Management*, *8*, 1-5. https://doi.org/10.55041/IJSREM35407

[26] Rath, C. K., Mandal, A. K., & Sarkar, A. (2023). Microservice based scalable IoT architecture for device interoperability. *Computer Standards & Interfaces*, *84*, 103697. https://doi.org/10.1016/j.csi.2022.103697

[27] Roche, K. E., & Baumgartner, R. J. (2024). Development of a strategy deployment framework combining corporate sustainability and operational excellence. *Corporate Social Responsibility and Environmental Management*, *31*(3), 2159-2174. https://doi.org/10.1002/csr.2683

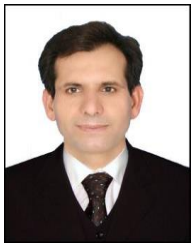
[28] Rodrigues, H., Rito Silva, A., & Avritzer, A. (2023, September). Performance Comparison of Monolith and Microservice Architectures: An Analysis of the State of the Art. In *European Conference on Software Architecture* (pp. 185-199). Cham: Springer Nature Switzerland. https://doi.org/10.54499/UIDB/50021/2020

[29] Saied, M. A. (2024). Migration to microservices: A comparative study of decomposition strategies and analysis metrics. https://doi.org/10.48550/arXiv.2402.08481

[30] Salau, S. O., Misra, S., & Demirors, O. (2020). Microservices in Higher Education: A Systematic Literature Review. *IEEE Access, 8*, 168553-168569.
[31] Sellami, K., Ouni, A., Saied, M. A., Bouktif, S., & Mkaouer, M. W. (2022). Improving microservices extraction using evolutionary search. *Information and Software Technology*, *151*, 106996. https://doi.org/10.1016/j.infsof.2022.106996
[32] Singaravel, G., Keerthilal, S., Gokulkannan, A., Prasanth, S., & Tharunkumar, R. (2020). A Meta Data System for Accumlation and Distribution of Bigdata. *International Journal of Advances in Engineering and Emerging Technology*, *11*(2), 121-124.
[33] Skrynnik, I. O., Fedotova, M. O., Darienko, V. V., & Dzhyrma, S. O. (2023). Experience and trends in the development of monolithic construction in the construction of buildings and structures. https://doi.org/10.32515/2664-262X.2023.7(38).2.190-195
[34] Souza, W. R., Araújo, D. F., & Muccini, H. (2020). Microservices in Education: A Case Study in Deploying the GitLab Development Model. *In Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*, 1-9.
[35] Swarnalatha, K. S., Mallya, A., Mukund, G., & Ujwal Bharadwaj, R. (2022). Solving Problems of Large Codebases: Uber's Approach Using Microservice Architecture. In *Emerging Research in Computing, Information, Communication and Applications: Proceedings of ERCICA 2022* (pp. 653-662). Singapore: Springer Nature Singapore. https://doi.org/10.1109/ICSAW.2017.65
[36] Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: a systematic mapping study. In *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. SciTePress.
[37] Thought Works article on Blue-Green Deployment (https://www.thoughtworks.com/continuous-integration).
[38] Viticchie, A., Basile, C., Valenza, F., & Lioy, A. (2018). On the impossibility of effectively using likely-invariants for software attestation purposes. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *9*(2), 1-25. https://doi.org/10.22667/JOWUA.2018.06.30.001
[39] Younis, Y. S., Hamed, S. H. A., & Meften, S. (2022). Models of Trust and Trusted Computations to an Ad-hoc Network Security. *Journal of Al-Qadisiyah for computer science and mathematics*, *14*(4), 92. https://doi.org/10.29304/jqcm.2022.14.4.1090
[40] Zhao, J., Chen, Z. H., & Gao, Z. H. (2019). Design and implementation of integrated scientific research management platform based on microservice architecture. *Radio Engineering, 49*(5), 436-441,
[41] Zimmermann, O., Lübke, D., Zdun, U., Pautasso, C., & Stocker, M. (2020, July). Interface responsibility patterns: Processing resources and operation responsibilities. In *Proceedings of the European Conference on Pattern Languages of Programs 2020* (pp. 1-24). https://doi.org/10.1007/978-3-031-84617-5_15

## Authors Biography

**Nagham Kamil Hadi** holds a Master's degree in Computer Science from Al-Qadisiyah University and works as a researcher in the fields of computer science, security, and machine learning. She teaches at Iraq's Al-Qadisiyah University. In the domains of computer science, software engineering, and information technology, Nagham has authored over ten peer-reviewed research pieces that have been published in prestigious conferences and journals. She is a network security pioneer, and her many practical works are backed by up-to-date, cutting-edge research.



**Saad Hussein Abed Hamad** is a researcher and software developer and a computer science Ph.D. candidate at Sfax University-Tunise. He is a lecturer at the University of Al-Qadisiya, Iraq. Saad has published over 30 refereed research articles in leading conferences and journals in the areas of computer science, software engineering, and information technology. His research interests include software engineering, microservice architecture, and web site design, Microservice, Big data.

**Safa Jaber Abbas** graduated with a master's degree in artificial intelligence. She is a lecturer at Al-Qadisiyah University's College of Computer Science and Information Technology. Her numerous studies in artificial intelligence have been published in scholarly journals and conferences. As part of her applied job, she supports scientific and administrative work and uses artificial intelligence to support research. She also completes high-quality work in academic institutions.

**Gamal Fathalla Ali**, a Master of Mechanical Engineering, has over 40 years of research and applied expertise in computing, aviation engineering sciences, power plants, and structures, as well as over 20 research projects in these areas. He is also an expert in the design and preventative maintenance of control and monitoring systems. Lecturer at Libyan British University and Faculty of Mechanical Engineering Technology, Benghazi. He has also filed numerous ideas and inventions, including the design of a system for recording, documenting, and analysing flight barometers, as well as the installation and operation of security, protection, and monitoring systems.

A software engineer and lecturer at the Faculty of Computer Technology in Benghazi, **Mahmoud Mohamed Mahmoud Maadi** has fifteen years of programming experience, is highly proficient in computer maintenance and network setup, has designed numerous systems and websites, and has numerous published research papers in programming specialties.