Microservice architecture for food order processing and supply chain: A redesign of food chain processes at Omega Food Services

Dennis Mashoko.[1], Cross Gombiro.[2], Plaxedes R. Mhlanga.[3], Viola Jowa.[4], Taurai Rupere.[5], Edmore Mlambo.[6]

Abstract

Omega Food Services is one of the leading enterprises in the fast-food industry across Africa. The organisation is well known for its rapid expansion and strong brand presence. In Zimbabwe, the company has achieved a 25% annual growth in customer acquisition. However, the growth has strained its online food processing system, which operates on a monolithic software architecture. The main challenges associated with the architecture include limited scalability, slow response times during peak hours, and difficulties in system maintenance and upgrades. The system affects the overall customer experience and operational efficiency in a negative manner. This is because, as the number of customers continues to grow, the monolithic system struggles to handle the increased transactions. This results in delayed food order processing, contributing directly to poor customer satisfaction. The paper proposes a transition to a microservices architecture. The architecture will enable independent scaling of services, faster deployment of updates, faster processing of transactions, and improved fault tolerance. The architecture will make Omega Food Services more competitive and resilient in the fast-paced food industry. The Design Science Research methodology was used to guide the design of the proposed system, including requirement modeling, domain modeling and architectural patterns, resulting in a Food Ordering Microservice Framework. The expected benefits include enhanced system responsiveness, better resource management, and a more adaptable platform that can meet evolving client expectations.

Keywords: Scalability, Customer Experience, Fast Food Industry, Supply Chain Management.

1. INTRODUCTION

The food service industry in Africa has grown rapidly in recent years. This has created demand for convenient, accessible, and efficient food service delivery. Omega Food Services, a major player in the industry, operates multiple outlets that are connected through a centralised computer network, which is supported by an online ordering platform. The organisation's existing food order processing and supply chain system relies on a monolithic architecture. The architecture has become a restrictive factor as the business expands. The monolithic architecture fails to handle services associated with high user demand, usually resulting in slower response times, reduced system flexibility, and increased maintenance complexity.

These challenges are not unique to Omega Food Services. Many organisations in the food service sector are experiencing constraints that are similar as they attempt to modernise their digital platforms as a measure to meet rising customer expectations. The demand for fast, reliable, and personalised services is growing. Organisations have to incorporate scalability and flexibility in their systems. Omega's case reflects the broader trend in both the African and global food industry, where businesses need to adapt their technology infrastructure to remain competitive in a highly evolving market.

As the number of customers is growing, competition is also intensifying. Omega Food Services faces the need to improve system scalability, enhance service flexibility, and increase operational efficiency. Microservices architecture offers a favourable solution as it decouples the system into independent deployable services. These services will each focus on specific functionalities. The approach can provide significant benefits, including targeted scaling, faster deployment, and easier system maintenance. The architecture will make Omega Food Services more responsive to customer needs, at the same time boasting its resilience, which is much needed in the competitive market.

The aim of the research is to solve Omega Food Services' operational challenges by exploring the application of the microservices architecture. The architecture will be used as a technique to redesign

[1] Systems Engineer, Information and Communication Technology Department, University of Zimbabwe, *dsmashoko@gmail.com*
[2] Lecturer, Computer Science Department, University of Zimbabwe, *cgombiro@ceic.uz.ac.zw*
[3] Lecturer, Computer Science Department, University of Zimbabwe, *pmhlanga@ceic.uz.ac.zw*
[4] Lecturer, Computer Science Department, University of Zimbabwe, *vjowa@ceic.uz.ac.zw*
[5] Lecturer, Computer Science Department, University of Zimbabwe, *trupere@ceic.uz.ac.zw*
[6] Lecturer, Computer Science Department, University of Zimbabwe, *emlambo@ceic.uz.ac.zw*

both the company's food order processing and supply chain management. The study will contribute to the growing field of microservices architecture in the food service industry, demonstrating how architectural innovations can drive the competitive advantage and, at the same time, enhance customer satisfaction.

2. PROBLEM STATEMENT

Omega Food Services, a leading fast-food restaurant chain in Africa, is currently using the monolithic software architecture. The architecture has given the organisation a lot of challenges, and this has been mainly caused by the growing customer base and changing operational needs. The architecture presents several challenges that impact scalability, flexibility, and security. These challenges limit the company's ability to quickly innovate and scale effectively when responding to market changes

The growing customer base is affecting the monolithic system in a major way as it is failing to scale, causing frequent malfunctions, accessibility issues, and disrupting service delivery. The monolithic system is highly coupled; therefore, system updates and maintenance usually take a lot of time to complete, complicating the processes, which usually result in compatibility issues, system downtime, and performance bottlenecks. The challenges prevent the timely implementation of new technological advancements, slowing down innovation and quick responsiveness to market changes and demands.

Another major challenge of the monolithic architecture is security vulnerabilities. They are caused by the tightly integrated nature of the architecture. A failure or breach in one area can compromise the entire system. The monolithic architecture lacks both flexibility and scalability. This has resulted in system downtime and inefficiencies, which are affecting both customer satisfaction and operational resilience.

To address these challenges, the study proposes to introduce a microservices architecture. The implementation of the architecture would allow Omega Food Services to decompose the system into independently manageable services, which would enhance scalability, maintainability, and security, at the same time reducing operational costs and downtime.

3. OBJECTIVES

i. Assess the current software architecture being used by Omega Foods and critically investigate its limitations.

ii. Design a microservices architecture that solves the current challenges and meets the specific needs of Omega Foods.

iii. Develop a plan for transitioning from the monolithic architecture to the microservices architecture.

4. LITERATURE REVIEW

This literature review aims to explore the benefits and challenges associated with the transition from monolithic to microservices architecture, focusing on scalability, flexibility, and security improvements relevant to Omega Food Services. Recent advancements in computer technology have created a huge demand for fast and reliable services. This has led many organizations to change and modify their existing software architectures. Omega Food Services, which is a well-established fast food restaurant chain, is currently facing challenges with its monolithic architecture.

The architecture has exhibited challenges in scalability, flexibility, and security, negatively impacting the organization's ability to scale operations and respond to customer needs efficiently. To address the issues, Omega Food Services is exploring a transition to a microservices architecture. The systematic literature review aims to examine the existing research on challenges of monolithic architectures.

The review will also focus on the benefits of microservices, closely looking at organizations that have similar operational demands, namely e-commerce and food delivery. The review will also highlight best practices, challenges, and performance metrics associated with the implementation of the microservice architecture.

The systematic approach was adopted for the literature review to explore the transition from monolithic to microservices architecture in the context of Omega Food Services. The inclusion and exclusion criteria were used. Relevant studies were identified from reputable academic databases, namely Journal of Information Security and Cybercrimes Research, Journal of Systems and Software, Sensors, International Journal of Advanced Computer Science and Applications (IJACSA), American Institute of Physics, Journal of Information Systems and Software, Computer Science - Research and Development, Indian Scientific Journal of Research in Engineering and Management, IEEE Latin America Transactions Journal, Applied Computing and Informatics, Engineering, Technology and Computer Science, Multidisciplinary Digital Publishing Institute and the Journal of Physics. The following keywords, such as 'implementation of microservice architecture for the food service industry', 'implementation of microservice architecture for e-commerce', 'challenges of monolithic architecture', 'microservices benefits in food delivery', 'microservices scalability and transition strategies' were employed to provide a comprehensive coverage of the topic.

The selection criteria focused on studies published within the last seven years that examine the advantages, limitations, and practical implementation of microservices in high-demand service-oriented industries, specifically those related to food delivery and e-commerce. The studies that did not talk about challenges of monolithic systems, microservice architectures in the food processing industry, and in e-commerce were excluded. The selected literature was precisely analysed to extract key data on scalability, maintainability, deployment agility, security considerations, and performance metrics, therefore providing a robust foundation for understanding how microservices can enhance the operational efficiency of Omega Food Services.

4.1 Comparative Analysis of Monolithic and Microservice Architectures

Monolithic architecture is defined as a traditional way of building applications. A monolithic application is built as a single and indivisible unit (Mosafi et al., 2025). Typically, such a solution comprises a client-side user interface, a server-side application, and a database. It is unified, and all the functions are managed and served in one place (Seedat et al., 2024). Generally, monolithic applications have one large code base and lack modularity. If software developers want to update or change something, they access the same code base. They have to code the whole system (Qian et al., 2020). Microservice architectures offer lower processing times under normal loads, and they also demonstrate superior scalability and performance under high concurrency, highlighting their advantages for dynamic, stress-prone operations (Mosafi et al., 2025).

Monoliths are large, unified applications where each component is largely interconnected and dependent on neighbouring components. Monoliths are the traditional approach to development, with all functions managed and served in one place, and everything built on a single code base (Qian et al., 2020). Changes that programmers may wish to implement in monolith structures will naturally change the entire stack. Activities such as testing apply to the entire stack as well, and so changes are grouped together in large releases that can take a relatively long time from start to finish. Microservices are distinct from monoliths in that they are composed of several smaller, self-contained, loosely joined components. Changes may be implemented on the individual components without affecting other services within the application (Chen, 2023).

4.2 Overview of Microservices Architecture

Microservice architectures have become increasingly popular in both academia and industry, providing enhanced agility, elasticity, and maintainability in software development and deployment (Ahmad et al., 2025). Microservices permit a modular approach to software development, enabling organizations to break down monolithic applications into smaller, independent services that can be developed and deployed independently (Alshuqayran et al., 2025). Microservice architecture has acquired considerable traction in recent years as a means of developing complex applications (Alshuqayran et al., 2018). This architectural style enables software to be organized into small, modular, and independently deployed services, where each service runs in its own process and communicates through lightweight, well-defined mechanisms to serve business objectives

A microservices-based architecture can expedite the implementation of a traceability system for food safety management, allowing for the tracking and monitoring of food products throughout the supply

chain. Microservices offer the benefits of scalability, fault tolerance, and flexibility, but they also come up with new challenges such as service discovery, inter-service communication, and data consistency (Vangala et al., 2023). Microservices offer a framework for building scalable and resilient software systems, but they require careful design and implementation to ensure quality and maintainability (Thota, 2025). Microservices architecture offers flexibility, scalability, and fault tolerance, but this also increases the security risks added by its distributed nature. Unlike monolithic applications, which limit security issues, microservices use different programming languages and frameworks, leading to critical security vulnerabilities. A dedicated security service that centralizes security management is the solution (Ahmad et al., 2025).

4.3 Challenges of Monolithic Architecture in Modern Applications

Many existing applications are built in a monolithic style, and all the services are found within a single unit process. A disadvantage of using the approach is that any development, testing, or scaling is carried out on the entire unit; therefore, changes can be slow and expensive Anggreainy et al., 2021).Monolithic architectures can be difficult to maintain and update, leading to increased technical debt and reduced agility. Monolithic architectures can lead to tight coupling between components, making it difficult to scale and modify the system (Qian et al., 2020). The monolithic architecture can initially reduce infrastructure costs during implementation. However, as the system expands, the architecture becomes expensive with time due to system faults and bottlenecks, which are challenging to manage and scale efficiently (Nayim et al., 2023).

The e-commerce sector faces a lot of challenges from monolithic systems; there is a need to upgrade or change from this traditional platform to microservice architectures that are integrated with DevOps practices. The monolithic architecture has scalability issues that are commonly found in conventional systems (Seedat et al., 2024). The microservice architecture is the solution as it improves overall system efficiency. Rapid development cycles, continuous integration, and automated deployment can be employed as techniques to facilitate quicker product launches (Paulo Francisco, 2020). The integration of security measures throughout the development lifecycle is important as it combats cyber and security threats, making the architecture resilient and adaptable to evolving market demands (Divyansh, 2024). Microservices architectures have become the solution to the limitations of traditional monolithic systems, especially in terms of scalability and rapid development cycles. Migrating from a monolithic to a microservices architecture is complex because of dependencies between system modules. Most approaches focus on design elements that are usually used in object-oriented software. Database migration remains unresolved. There is a need for mixed-method approaches to enhance the database migration process (Paulo Francisco, 2020)

4.4 Benefits of Microservices in Food Services and E-commerce

Microservice architecture is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource and API (Application Programming Interface). These services are built around business capabilities and are independently deployable by fully automated deployment machinery (Karan Dhiman et al., 2022). Microservices are a service-oriented architecture pattern wherein applications are built as a collection of various smallest independent service units. It is a software engineering approach that focuses on decomposing an application into single-function modules with well-defined interfaces. These modules can be independently deployed and operated by small teams that own the entire lifecycle of the service (Niño-Martínez et al., 2023).

The term “micro” refers to the sizing of a microservice, which must be manageable by a single development team (5 to 10 developers). In this methodology, big applications are divided into the smallest independent units (Hossen & Islam, 2023). Microservices architecture is a technique that allows a single application to be made up of an assortment of small, separate services. Each service can be independently developed, tested, and scaled. It is possible that each service can be written in a different language and use diverse storage technologies (Rasheedh & Saradha, 2020).

A lot of researchers have observed the benefits of microservice architectures for modern software systems. They discovered that microservices architectures can improve scalability, maintainability, and agility, while minimising the impact of failures and enabling faster innovation. Microservices

architectures improve the flexibility and adaptability of software systems, enabling organisations to respond quickly to changing business needs (Hossen & Islam, 2023). Modular development and efficient deployment of e-commerce applications in microservices can be made more efficient by using the Spring Boot framework. The framework addresses challenges related to service coordination, data consistency, and transaction management in distributed environments (Samoylenko & Selivanova, 2023).

The implementation of microservices-based architectures in the e-commerce sector needs to be centred on enhancing scalability and performance to meet growing customer expectations. Rapid development cycles can be used for enhancing deployment efficiency and fault isolation, allowing the e-commerce platforms to quickly adapt to market changes (Ileana et al., 2024).

4.5 Challenges in Transitioning from Monolithic to Microservices Architecture

Successful transitions require careful planning, including a thorough analysis of the existing system, clear communication with stakeholders, and a phased approach to implementation. Similarly, in a study by (Dirgahayu, 2023), the author emphasized the importance of designing for failure and building resilience into the system during the transition process.

Microservices have become an important architectural style for building robust and scalable software systems. A system's functionality is split into independent units. The microservices that communicate over a network can be deployed independently. The shift of complexity into the integration layer necessitates enhanced collaboration among stakeholders, stressing the importance of effective communication (Schwarz et al., 2025). Microservice architecture is an architectural style in modern software systems, characterized by small, independent services called microservices. This architecture is ideal to facilitate rapid feature deployment. However, it presents a challenge for software engineers, who often lack a comprehensive architectural view due to the distributed nature and complex inter-dependencies of microservices (Alshuqayran et al., 2025).

Microservice architecture occupies a core position in contemporary distributed system design due to its excellent scalability, mobility, and reliability. However, the introduction of microservices has also brought new challenges in distributed data processing and real-time query optimization. How to achieve high efficiency in data storage, querying, and processing under a microservice architecture has become a core issue in ensuring efficient system operation (Rodrigues et al., 2025). Monitoring query performance typically involves evaluating indicators from multiple dimensions, such as query response time, database workload, cache efficiency, and probability of query failure. These indicators enable developers to instantly grasp performance barriers during query execution and quickly identify potential performance risks. For example, if the query response time is too long, it may indicate that the database index needs to be optimized, or a cache failure leads to frequent direct queries. If the database pressure is too high, it may be due to overly dense database interactions between services, which may require optimizing query logic or improving load balancing efficiency (Li, 2025). In a monolithic application, all of the data objects and actions are handled by a single, tightly knit code base. Data is typically stored in a single database or file system. Functions and methods are developed to access the data directly from this storage mechanism, and all business logic is contained within the server code base and the client application (ZargarAzad & Ashtiani, 2023).

It is possible to migrate several monolithic applications and platforms, each with its own data storage mechanisms, user interfaces, and data schema, into a unified set of microservices that perform the same functions as the original applications under a single user interface. Migrating these applications to microservices offers the following advantages, namely removing duplication of effort for manual entry, reducing programmatic development risks, providing a single, unified view of the data, and improving the control and synchronization of these systems.

The Service-Oriented Modelling and Architecture method was employed, and microservices were evaluated in terms of scalability, performance, and availability. The test results were implemented using Apache JMeter, and they reflected the superiority of microservices over the monolithic architecture in terms of flexibility, improved scalability, and better availability. These insights reflect that microservices can be implemented in large-scale enterprise environments that have high user demands and perform well (Dirgahayu, 2023). The design of microservices has high cohesion and low coupling that optimizes the system for future scalability and easy maintenance, all of which makes the platform

a practical model for fast food service systems needed in adaptable and modular architectures (Divyansh, 2024).

The researchers introduced a microservice platform designed to enhance flexibility and maintainability in fast food services through distributed service bases, micro-applications, and micro-services. The platform supported efficient order processing by enabling micro-applications to call specific micro-services, with key components such as the registration centres and service gateways to ensure unified access.

Implementing a microservice approach means a quicker build and deploy process compared to a larger, centralized application. There is no wasted space from unnecessary and unwanted features, as developers can choose how each microservice is implemented (Seedat et al., 2024). Microservices also deal well with failure. It is certainly possible for a service to fail; however, in a microservices architecture, the other services around the failed service can continue running whilst fixes are carried out. (Alshuqayran et al., 2025).

The increased adoption of microservice architectures brings a new set of challenges, such as managing fluctuating workloads. For example, Amazon experienced service outages in 2018 due to its failure to manage workload fluctuations. Similarly, companies such as Microsoft and Zoom faced service disruptions during the COVID-19 pandemic due to sudden surges in workload (Ahmad et al., 2025). Microservice architectures have become a widespread option for designing distributed applications, but designing secure microservice systems remains challenging. To implement secure systems and ensure that they remain secure throughout the development and production cycles, developers must identify and understand the various security tactics used throughout their systems (Genfer et al., 2025).

To address these challenges, employing a higher level of abstraction to visualize the entire system can effectively eliminate extraneous implementation details and enable developers to focus on the security concepts employed and their associated implications without becoming entangled in the intricate web of various technology stacks (Genfer et al., 2025).The researchers developed a microservices architecture using the Saga pattern, which had a backup mechanism. The architecture addressed the challenges of both data consistency and transaction integrity, which usually occur in complex applications developed with microservices. This was relevant for applications requiring precise data synchronization across multiple services, such as those found in supply chains and food order processing systems. The automatic fault recovery through backup points and restarts was also provided, therefore improving resilience and operational uptime, which are key factors in high-demand sectors like food services (Lee et al., 2023).

The authors provided an in-depth comparison between microservices and service-oriented architecture (SOA). The researchers identified microservices as a modern and highly improved variant of SOA that incorporates technologies like RESTful, HTTP, and DevOps practices. The paper discussed key tenets of microservices, showing that they address SOA limitations and also introduce advanced cognitive and design complexity. The authors stressed that microservices require strong architectural expertise to develop and manage (Abdiaziz, 2024). The researchers explored the implementation of microservices architecture in an e-commerce web service. They highlighted how microservices improve flexibility and maintainability through enabling independent development and minimizing service inter-dependencies. This adaptability allows e-commerce platforms and similarly structured industries like food delivery to easily evolve with changing business requirements more effectively (Niño-Martínez et al., 2023).Microservices, when containerized, significantly outperform monolithic architectures in terms of scalability, resource optimization, and processing speed. Microservices offer better support of high-traffic environments by allowing tasks to run independently (Anjum Era et al., 2025).

4.6 Evaluation Metrics for Microservices Architecture

Several Researchers have examined the key metrics that are important for evaluating the success of microservices architectures. The authors found that system performance, scalability, and availability were the most important metrics for evaluating microservices architectures (Rodrigues et al., 2025) The authors found that user satisfaction was a critical metric for measuring the success of microservices

architectures (Mosafi et al., 2025). The researchers investigated the use of microservices in a Point-of-Sale application that utilized RESTful APIs and Webhooks to boost performance, scalability, and data accuracy. They employed unit and functional testing. The results reflected an improved data input accuracy from 80% to 100%. This showcased efficient and enhanced data handling, which is a critical need in fast-paced environments like food services. The approach underscored the benefits of microservices for efficiency and accuracy, making it highly applicable to food order systems where speed and precise data communication are critical (Rizki et al., 2024).

The authors analyzed microservice architecture within e-commerce platforms. They focus on achieving modularity, scalability, and deployment flexibility. Docker was highlighted as essential for containerising microservices, ensuring seamless operations across diverse environments. Kubernetes was also highlighted as good at automating container deployment, scaling, and management. The integration of microservices, Docker, and Kubernetes created a stable foundation for continuous integration and deployment (CI/CD), at the same time boosting development efficiency and adaptability, which is essential in high-demand applications like e-commerce (Abdiaziz, 2024).

The paper examined inter-service communication in microservices within the e-commerce industry, addressing scalability and fault tolerance challenges relevant to high-demand industries like food delivery. It analysed different communication methods and their impacts on microservices performance, emphasizing the importance of selecting efficient communication protocols to support reliable and scalable operations in distributed systems (Asrowardi et al., 2020). The researchers discussed the application of microservices architecture in e-commerce using DevOps practices. The study demonstrated how microservices improve scalability and operational efficiency through the support of rapid development, continuous integration, and automated deployment. The architecture addresses security considerations, making it equally suitable for food ordering and delivery services that require high scalability and frequent updates (Divyansh, 2024).

4.7 Case Studies in Microservices Adoption

The researchers introduced a framework for integrating enterprise architecture (EA) patterns as microservices. The researchers aimed at enhancing the alignment between EA design and its practical application during transitions from monolithic systems. The framework was structured around four steps, namely pattern selection, pattern adaptation, microservices elaboration, and application development. A multi-seller online store case study reflects how the framework successfully applied the Vending Machine EA pattern. This ensured consistency across the business and technology architecture layers. This approach is relevant to complex systems such as food order processing, as it is able to maintain coherence between high-level enterprise goals and operational technology, creating a reliable service delivery environment (Dirgahayu, 2023).

4.8 Identified Research Gaps

The existing literature highlights a significant gap in empirical studies assessing the software architectures of food service organizations, especially regarding their limitations. The literature does not clearly identify specific points and inefficiencies that hinder the operational performance of microservices architectures in the food industry. The understanding of these limitations is important as it creates a basis for the development of effective microservice architecture solutions that are custom-made to satisfy the unique demands of the food service sector. There is a need to address the complexities of how to migrate from a monolithic to a microservices architecture in the food service sector.

The current research overlooks the adaptations necessary for microservices implementation in food service processing environments. There is a need for a microservices architecture that caters to the specific challenges, such as real-time data handling, inventory management, and customer interaction, found in the food service industry. A comprehensive understanding of how microservices can enhance scalability and operational efficiency in the context of the food service industry is needed to contribute to the theoretical framework of microservices.

The current literature lacks practical guidance on transitioning from monolithic to microservices architecture, especially in operational environments that have established monolithic systems. There is a need for a detailed transition plan for addressing the complexities involved in this migration, which

will look at data migration, service orchestration, and ensuring operational continuity. There is a need for a plan that will provide valuable insights into the methodologies, best practices, and frameworks necessary for successful implementation in a food service context.

The researchers acknowledge the potential benefits of microservices architecture, but there is a notable lack of standardized performance metrics specifically tailored to the food service industry. This gap leaves organizations like Omega Foods without a clear framework for assessing the effectiveness of their microservices implementation.

The establishment of metrics that reflect the unique demands of food order processing, such as order accuracy, speed of service, inventory turnover, and customer satisfaction, will ensure that the evaluation of the microservices architecture is not only based on theoretical advantages but also grounded in practical, quantifiable outcomes that resonate with the operational realities of the food service businesses.

4.9 Conceptual Framework

The conceptual framework for this project will be based on the following mechanisms:

i. Assessment of the current monolithic architecture: This component will involve a comprehensive analysis of the current monolithic architecture used by Omega Foods to identify the limitations and challenges related to the current architecture.

ii. Design of a microservices architecture: This component will involve the design of a microservices architecture that addresses and solves the limitations of the current architecture and meets the detailed needs of Omega Foods.

iii. Design and implementation of a transition plan: This component will involve the development of a detailed plan of a transition from the monolithic architecture to the microservices architecture.

A successful transition from the monolithic architecture to the microservices architecture can be achieved by Omega Food Services, as it will follow this conceptual framework. The main goal and aim of the conceptual Framework is to improve performance, agility, scalability, maintainability, flexibility, and security, while addressing user satisfaction and enabling faster and smoother operation of the system.

The diagram below shows that stages of the conceptual framework of transitioning from a monolithic architecture to a microservices-based architecture.

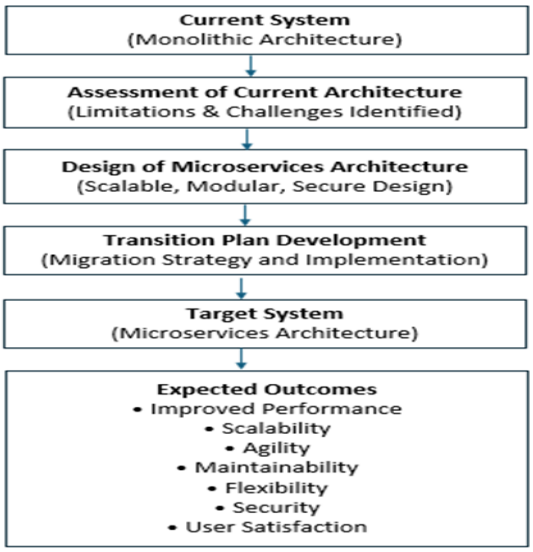

Figure 1. Conceptual Framework diagram for Transitioning from Monolithic to Microservices Architecture

5. METHODOLOGY

The study employed Design Science Research methodology, a problem-solving approach that focuses on creating innovative solutions to real-world challenges well (Dirgahayu, 2023).In Design Science Research, the research output can take the form of an artefact, method, model or framework that is designed to address identified problems systematically, a microservices-based framework was developed to address the limitations of Omega Food Service's monolithic system, including scalability, slow response times and maintainability issues.

The Design Science Research methodology guided the research through the following procedures:

1. Functional requirements development – defining how each epic operates and the expected system behaviours.
2. Domain modelling – constructing models of Omega Food Service's food chain processes and use cases to set the context for the framework.
3. Architectural design selection – applying microservice design principles and the model-view pattern for the user interface, informed by established design patterns well (Dirgahayu, 2023).
4. Framework modelling and documentation – creating UML diagrams and related documentation to illustrate system interactions and pattern applications.

The use of Design Science Research in the study ensured that the proposed framework is systematically aligned with organisational needs and grounded in rigorous design principles. This methodology involves the creation of a new artifact that will involve a new system process and solutions to solve the challenges at hand. The following procedures will be followed:

1. Develop the functional requirements for each epic, explaining how the system will operate.
2. Create a domain model to set the context for Omega Foods food chain processes with use cases.
3. Select the architectural design method presented in the module and follow the outlined steps.
4. Implement a model-view pattern for the user interface and use the micro-service pattern for the underlying service(s).
5. Transform the UML diagrams to code by showing appropriate design patterns that apply.
6. Develop appropriate UML documents for the system and each selected pattern.

5.1 Requirements Modelling

The first step in the Design Science Research methodology encompasses system development, which is the elicitation and modelling of functional requirements for each identified epic. Design Science Research stresses that artefacts should be designed to solve real organisational problems. Looking at the case of the proposed Omega Food Service system, the functional requirements are directly derived from the identified challenges in the current food chain processes, ensuring that the system addresses actual operational needs rather than abstract ideas

The Design Science Research methodology aids the requirements process by making the research focus on problem driven design. All the system requirements are formulated to align with stakeholder needs thus ensuring that the system is both relevant and usable. The methodology offers support to the proposed microservice architecture therefore guaranteeing modularity, scalability and maintainability. The methodology also reflects real-world processes, captured in a domain model of the food chain, which provides context for system interactions and defines the boundaries for each microservice.

Additionally, use cases are developed for each functional requirement, illustrating how users and all the system components will interact to accomplish business goals. Thus ensuring both clarity and traceability from requirements and system functionalities challenges.

The following subsequent steps of the Design Science Research methodology that will be applied include:

1. Design and Development – constructing of the microservice-based framework based on the defined functional requirements.

2. Demonstration – illustrating how the framework can be applied to model and support Omega Food Service's food chain processes, showing its practical utility in addressing the identified challenges.
3. Evaluation – assessing the framework against criteria such as effectiveness, adaptability and alignment with stakeholder needs, therefore ensuring that it provides meaningful guidance for system implementation.
4. Communication – documenting and presenting the framework, design rationale and insights to stakeholders and the research community, thus highlighting its potential application in practice.

The explicit linking of requirements elicitation to Design Science Research methodology principles makes the proposed system to be grounded in solving real challenges, rather than being a theoretical construct.

Online Ordering- The new system must allow customers to place orders online through a secure and user-friendly platform, which is a major requirement. This would necessitate the development of an interactive website and mobile application that makes it simple for customers to browse menus, personalize orders, and securely make online payments. Order history, real-time tracking, and the capacity to receive customer feedback are the features of the system.

Inventory Management- For a well-designed food ordering, processing, and supply chain, a comprehensive inventory management system is essential. This system will make it possible to manage and monitor inventory levels from a centralized location and to track inventory levels in real time across all locations. This will make it possible for supplies to be replenished in time, and automated notifications will be activated when inventory levels drop.

Supply Chain Logistics- Supply chain logistics that are both effective and dependable and crucial for ensuring that all food products arrive at customers in good condition and on time. Processes for selecting and acquiring raw materials, managing storage and transportation, and ensuring quality control throughout the supply chain ought to be included in the model.

Food Safety- service operation. The model ought to include comprehensive food safety procedures and systems like temperature controls, regular testing and inspections, and staff training programs on how to handle food safely. Audits should be done on a regular basis, both internally and externally, to find any potential problems that need to be fixed.

Customer Service- Customer service can make any business flourish or break the business completely, and Omega Food Services should prioritize customer satisfaction by providing efficient delivery, responsive complaint management, and other customer service systems. Well-designed and efficient customer support systems should be put in place, including helplines, complaint tracking systems, and follow-up protocols.

By factoring into account these requirements into the redesign of food chain processes, the organization can offer its clients a quality and reliable service while improving its operations and standard of procedures.

Research-Based Modelling of System Interactions and Domain Structures

The design of the use case diagrams and domain models presented for the Online Ordering, Inventory Management, Food Safety and Delivery Logistics subsystems was a research-driven activity guided by the Design Science Research methodology. In line with Design Science Research principles, the models were derived from a combination of empirical problem analysis, established research on microservices architectures, and expert evaluation.

Prior studies highlight that effective microservice-based systems require clear separation of concerns, well-defined service boundaries, and explicit interaction models to support scalability, maintainability, and resilience well (Dirgahayu, 2023).These findings directly influenced the decomposition of the

system into distinct functional epics and informed the identification of actors, system interactions, and entity relationships represented in the use case and domain diagrams.

Empirical observations of Omega Food Services' existing monolithic system namely delayed order processing, tight coupling between inventory and ordering functions and limited scalability during peak hours, served as problem evidence that shaped the modeled interactions and responsibilities of each subsystem. The resulting diagrams were subsequently evaluated by a panel of five expert software architects, whose feedback validated the suitability of the proposed service boundaries, entity relationships and interaction flows. The expert evaluation constituted demonstration and evaluation of the Design Science Research phases. This ensured that the diagrams reflect both current research-informed best practices and real-world operational requirements.

Epic 1: - Online Ordering System

Functional Requirements:

- Allow customers to register online and manage their user accounts.
- Customer browses the menu and selects food products to add to their cart.
- Provide real-time notification on order status and delivery time.
- Allow customers to give comments and feedback on their shopping experience

Use Cases

- A customer navigates to the Omega Foods website, browses the menu, and adds items to their cart.
- A customer inputs their delivery details and payment details.
- A customer tracks the progress of their order and receives real-time updates on delivery time.

Use Case for Online Ordering

The use case diagram below illustrates the functional requirements of the Online Ordering process.

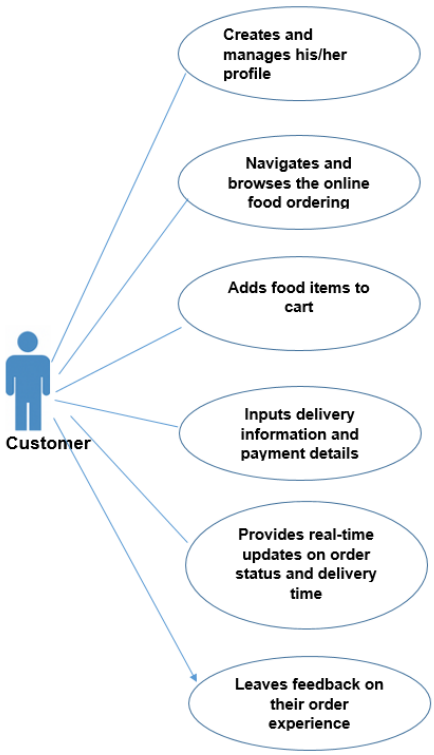

Figure 2. Use Case diagram for Online Ordering process

Domain Model for Online Ordering

The Domain Model diagram below shows the relationships between the key entities for the Online ordering system.

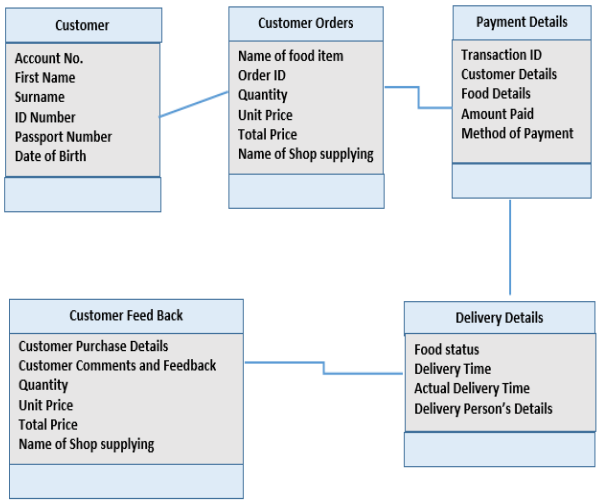

Figure 3. Domain Model for Online Ordering

Epic 2: - Inventory Management System

Functional Requirements:

- Track inventory levels in real-time.
- Provide alerts for low inventory levels.
- Allow managers to update inventory counts.
- Generate reports on inventory usage and waste.

Use Cases

- A manager logs in to the inventory management system, updates the inventory details, and generates a report on inventory usage.
- An employee receives an alert for low inventory levels and orders additional supplies.
- The system automatically adjusts inventory levels based on order fulfillment.

Use Case for Inventory Management

The following diagram shows the Use Case diagram for the Inventory Management, showing how the entities interact and communicate, as explained above.

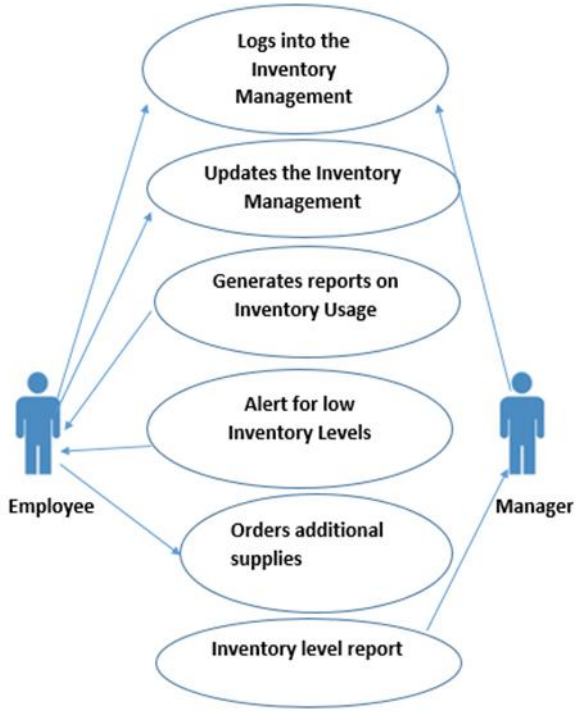

Figure 4. Use Case for Inventory Management

Domain Model for Inventory Management

The Domain Model diagram below shows the relationships between the key entities for the Inventory Management system.

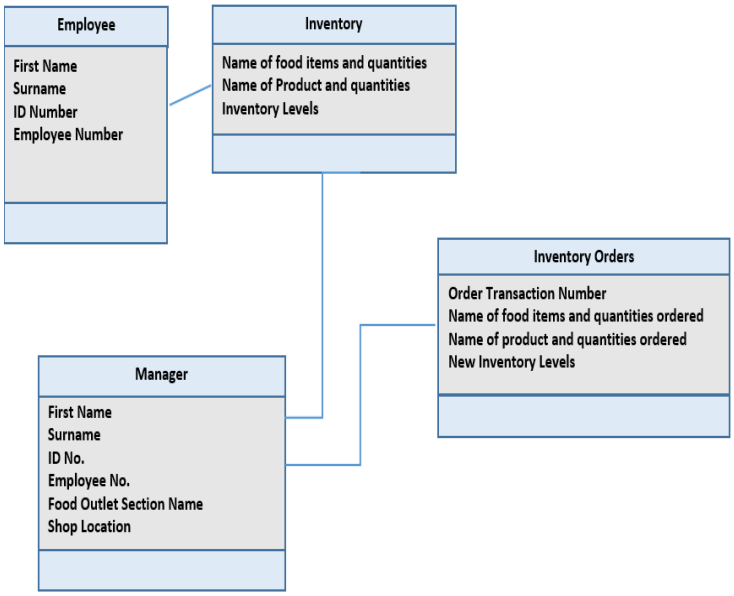

Figure 5. Domain Model for Inventory Management

Epic 3: - Food Safety Protocols and Training Programs

Functional Requirements:

- Establish and enforce standardized food safety protocols.
- Develop and implement a training program for all employees.
- Conduct regular audits and inspections of food safety procedures.

Domain Model: A comprehensive system for ensuring food safety across the restaurant chain, including standard operating procedures, training materials, and audit and inspection tools.

Use Cases

- An employee completes a food safety training program and quizzes to demonstrate compliance.
- An auditor reviews restaurant operation and verifies compliance with food safety regulations.
- An employee raises a concern about food safety procedures, which is then investigated and addressed by management.

Use Case for Food Safety and Training Programs

The following diagram below shows the Use Case diagram for the Food Safety and Training Programs showing how the entities interact and communicate, as explained above.

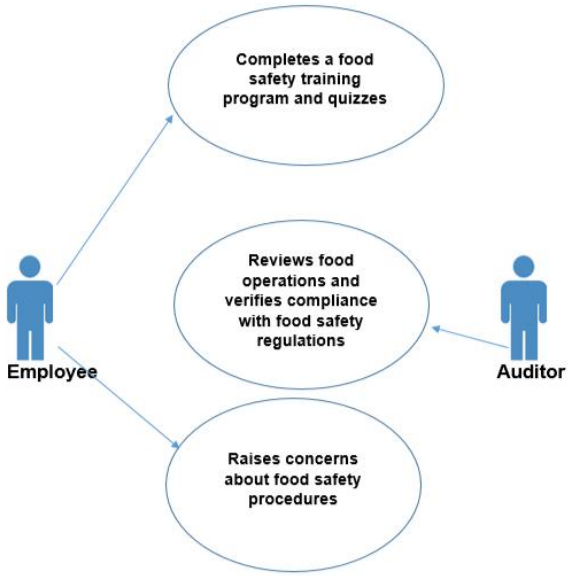

Figure 6. Use Case for Food Safety and Training Programs

Domain Model for Food Safety and Training Programs

The Domain Model diagram below shows the relationships between the key entities for the Food Safety and Training Programs.

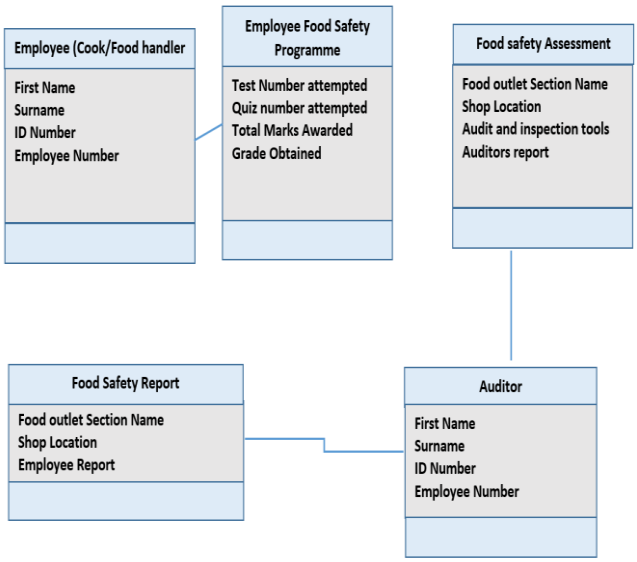

Figure 7. Domain model for Food safety and training programs

Functional Requirements

- Optimize delivery routes for drivers to reduce wait times and improve order accuracy.
- Track delivery vehicles in real-time to update order statuses and ETA.
- Integrate with third-party delivery services for wider reach and expanded options.
- Provide customer support for delivery-related issues.

Domain Model: A platform that optimizes delivery logistics by integrating with various delivery services and tracking systems.

Use Cases

A driver receives optimized delivery routes for the day and tracks their progress via GPS.

- A customer receives real-time ETAs and order status updates for their delivery.
- A customer contacts customer support with a delivery-related issue, which is then addressed by support staff.

A clear understanding of how the various entities in the food chain interact and the various scenarios that can occur is provided by the domain model and use cases. The functional requirements and software architecture for the proposed microservice architecture pattern for food order processing and supply chain management will benefit from this understanding.

Use Case for Delivery Logistics and Operations

The following diagram shows the Use Case diagram for the Delivery Logistics and Operations, showing how the entities interact and communicate, as explained above.

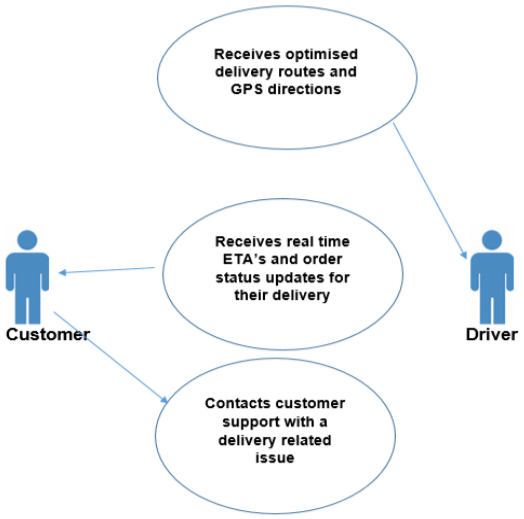

Figure 8. Use case for Delivery Logistics and Operations

Domain Model for Delivery Logistics and operations

The Domain Model diagram below shows the relationships between the key entities for the Delivery Logistics and operations system.

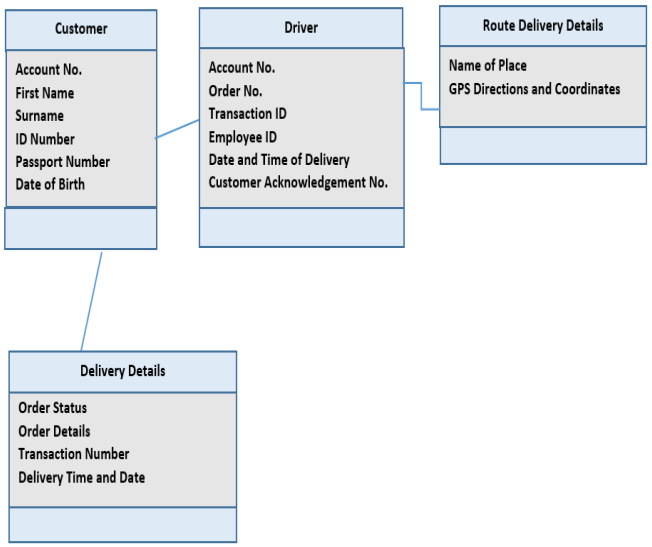

Figure 9. Domain Model for Delivery Logistics and operations

In conclusion, Omega Foods's supply chain management and processing of food orders are clearly defined by the domain model's use cases, relationships, and entities. The software architecture and functional requirements for the proposed microservice architecture pattern will benefit from this comprehension.

5.2 Proposed microservice architecture pattern consists of six services

1. Order Management Service- the service is responsible for managing the complete order process, including order creation, modification, and cancellation. It communicates with the inventory management service to make sure that all the needed goods and food items are available. It also links up with the payment management service to process all the required payments

2. Inventory Management Service- the service is responsible for managing the inventory of all the goods and food items needed by the kitchen management service, and to make sure that the stock levels of the kitchen management service are adequate

3. Kitchen Management Service- the service is responsible for making sure that food is cooked well and also packaged in an efficient and orderly manner. It communicates with the inventory management service to make sure that the much-needed goods, services, and food items are available, and also with the delivery management service to prepare the food for delivery and also prepare adequate transportation services of the food.

4. Delivery Management Service- the service is responsible for the physical delivery of food items and updating the customer on the status of his/her food item(s). It communicates with the kitchen management service to prepare the required food and pack it accordingly. It also is in constant communication with the service management service to deal with any service issues that may be raised by the clients.

5. Service for managing payments- The processing of client payments, refunds, and cancellations is the primary responsibility of the service. In order to manage and resolve all customer payment issues, it is in constant communication with the order management service and the customer management service.

6. Customer Management Service- All inquiries, complaints, comments, and feedback from customers are managed and handled by the service. To address all customer-related issues, the service also communicates with the order management, delivery management, and payment management services.

The microservice pattern is used for the underlying services in the microservice architecture, while the model-view pattern is used for the user interface. Using REST APIs or message queues, the various services can communicate with one another, offering greater adaptability, scalability, and dependability.

Real-time inventory management, order processing delays, and inadequate channels for departmental communication are just a few of the issues highlighted in the literature review that are addressed by the proposed microservice architecture pattern. Food order processing and supply chain management benefit greatly from the greater flexibility, scalability, and dependability provided by the microservice architecture pattern.

In conclusion, Omega Food Service's proposed microservice architecture pattern for processing food orders and managing the supply chain is based on the Design Science Research methodology. This included developing functional requirements, selecting an architectural design method, developing the software architecture, implementing design patterns, and making sure that the various functional areas could communicate with one another effectively. The proposed microservice design addresses the difficulties recognized and gives a versatile and dependable answer for food order processing and supply chain management in the food service industry

5.3 Architectural Design Method and Architectural Review Methods

The Model-View-Controller (MVC) design pattern would be an ideal architectural design method.

An application is broken up into three interconnected parts using the MVC pattern: The Model, which embodies the data and logic of the business; the View, which is an image of the interface; and the Controller, which updates the Model and View in response to user input.

The MVC pattern is well-suited for complex systems with multiple entities and interactions, as it promotes separation of concerns and modularity. In the case of Omega Foods, the Model would represent the food items, orders, inventory, and other entities involved in the system.

The user interface for order placement, inventory management, and other tasks would be represented by the View. The Controller would take care of user input, update the Model and View in line with it, and interact with other system parts like the Kitchen and Delivery parts.

The Model and Controller components could be implemented as separate services; the MVC pattern would also make it easier to create a microservice architecture pattern. This would allow for greater modularity, scalability, and flexibility.

In conclusion, the Model-View-Controller (MVC) pattern would be a good architectural design method for the proposed microservice architecture pattern for Omega Food Services food order processing and supply chain management because it encourages modularity and promotes concern separation.

5.4 ATAM review of the proposed microservice architecture pattern for food order processing and supply chain management at Omega Foods food service

5.4.1 Determining the stakeholders' issues

Customers, employees, suppliers, and management are all stakeholders in this system. The ease with which orders can be placed, the accuracy of the information provided, and the promptness with which orders are delivered are important to customers. The employees are concerned about the ease of inventory management, food preparation, and order delivery. The accuracy of inventory levels and the promptness with which orders are delivered concern the suppliers. The system's overall effectiveness, scalability, and safety are of the utmost importance to management.

5.4.2 Transition Plan for Omega Food Services Microservices Migration

The migration from the existing monolithic system to the proposed microservices architecture will follow a phased. A careful monitoring approach will be used to minimize operational disruption. The process will begin with an assessment of current data structures, workflows, and inter-dependencies across modules such as orders, inventory, kitchen, delivery, payments, and customer service. Non-critical services, such as Customer Management and Payment, will be migrated first, followed by Order, Inventory, Kitchen, and Delivery Services. This is important as it allows controlled integration and verification at each stage.

The timeline will cover 12 weeks, the plan includes system assessment and stakeholder consultation which will span (Weeks 1–2), followed by development of migration scripts and containerized environments with Docker and Kubernetes which will cover (Weeks 3–6), phased service migration and data transfer to MySQL and MongoDB with REST API and message queue integration will span (Weeks 7–10), then lastly deployment, performance testing, user training and monitoring (Weeks 11–12).

Key risks include operational disruption, data loss, processing delays, and staff adaptation challenges. The challenges will be mitigated through backup and rollback procedures, automated data validation, temporary parallel runs, targeted and comprehensive staff training, and dedicated support.

While the transition entails costs for container orchestration, additional infrastructure, and personnel time, however, these are outweighed by the long-term benefits, which include improved scalability, modularity, faster feature deployment, and enhanced reliability. The Transition plan ensures a structured, low-risk migration aligned with Omega Food Services operational goals and future growth.

5.4.3 Designing the Microservices Architecture for Omega Food Services

The proposed Omega Food Services microservice architecture pattern consists of several independent microservices, each responsible for a specific business function. The key services include Order

Management**,** Inventory Management**,** Kitchen**,** Delivery, and Payment microservices. Each service exposes its functionality through well-defined RESTful APIs, which will be implemented using the Spring Boot framework. They will communicate with other services through lightweight message queues, namely Apache Kafka and RabbitMQ, and will also employ synchronous API calls where necessary.

The Order Management microservice will be responsible for placement, validation, and tracking orders. It will store the transactional data in the MySQL relational database; this will be done to maintain both consistency and reliability.

The Inventory Management microservice will be responsible for maintaining real-time stock levels and integrating with supplier systems through secure APIs. It will communicate with the Kitchen service to update ingredient availability, and it will rely on a NoSQL database, which will be MongoDB, to efficiently manage inventory records that will be changing rapidly. The Kitchen microservice is responsible for processing incoming orders, updating preparation statuses, and ensuring communication with the Inventory and Order Management services.

The Delivery microservice will integrate with third-party mapping APIs, in this case Google Maps API will be used for routing and will employ event-driven messaging to update customers on delivery statuses in real time. Finally, the Payment microservice will securely process online payments through third-party gateways; in this case, the PayPal, Zimswitch, and Ecocash payment services will be used. Other payment services will be added at a later date after the system has been functional. All communication and updates of the system's microservice services will be encrypted through API calls.

Performance and scalability have to be functional always, to ensure that all microservices will be containerised using Docker and orchestrated through Kubernetes. This is important as it will allow for horizontal scaling of high-demand services, at the same time managing service discovery, load balancing and traffic routing, ensuring fault tolerance and efficient resource utilization, for example the Order and Payment services during peak hours will benefit from this. Service discovery and load balancing will be controlled and managed by Kubernetes Ingress controllers and service mesh frameworks. This is vital as it will enforce efficient routing and fault tolerance.

Auto-scaling will be based on CPU and memory thresholds. The system health will be continuously monitored using Prometheus and ELK stack so as to trigger scaling actions dynamically. The design will enable Omega Food Services to dynamically allocate resources, minimize downtime, and provide a resilient, high-performance platform that is capable of supporting the growing customer base

5.4.4 Analyzing the architectural model

We evaluated the proposed microservice architecture pattern based on its impact on the following quality attributes and stakeholder concerns:

- Performance- The microservice architecture pattern could have an impact on the performance of the system, as the communication between microservices could introduce latency and overhead. To mitigate this risk, we propose using a message queue system, such as RabbitMQ or Kafka, to handle asynchronous communication between microservices and improve scalability.
- Reliability- The microservice architecture pattern could have an impact on the reliability of the system, as failures in one microservice could affect the entire system. To mitigate this risk, we propose implementing redundancy and fault tolerance mechanisms, such as load balancing, automatic failover, and circuit breakers.
- Security- The microservice architecture pattern could have an impact on the security of the system, as the communication between microservices could increase the attack surface. To mitigate this risk, we propose implementing secure communication protocols, such as SSL/TLS, and using authentication and authorization mechanisms, such as OAuth2 or JWT.
- Maintainability- The microservice architecture pattern could have an impact on the maintainability of the system, as the complexity of the system could increase with the number of microservices. To mitigate this risk, we propose using a container orchestration platform,

such as Kubernetes or Docker Swarm, to manage the deployment and scaling of microservices, and using a centralized logging and monitoring system, such as ELK or Prometheus, to track the performance and health of the microservices.

5.4.5 Identifying and resolving architectural risks

Based on the analysis, we identified the following architectural risks and propose solutions to mitigate or eliminate them-

- Risk 1- Performance degradation due to communication overhead between microservices.
  Solution- Use a message queue system, such as RabbitMQ or Kafka, to handle asynchronous communication between microservices and improve scalability.
- Risk 2- System failures due to failures in one or more microservices.
  Solution: Implement redundancy and fault tolerance mechanisms, such as load balancing, automatic failover, and circuit breakers.
- Risk 3- Security breaches due to unsecured communication between microservices.
  Solution: Implement secure communication protocols, use authentication and authorization mechanisms like OAuth2 or JWT, as well as SSL/TLS.
- Risk 4- Costs associated with maintenance because of the system's complexity

  Solution: Manage the deployment and scaling of microservices with a container orchestration platform like Kubernetes or Docker Swarm, and monitor their performance and health with a centralized logging and monitoring system like ELK or Prometheus.

5.4.6 Displaying the findings

Stakeholders are given the ATAM evaluation's findings and proposed solutions, and we solicit their feedback and support. In addition, we devise a strategy for putting the microservice architecture pattern into practice and talk about the risks and trade-offs that are associated with the suggested solutions.

The ATAM review of the proposed microservice architecture pattern for Omega Foods Services food order processing and supply chain management reflects that the quality attributes and stakeholder concerns have been identified, the architectural model has been analyzed, architectural risks have been identified and resolved, and stakeholder feedback and buy-in have been obtained. A message queue system, redundancy and fault tolerance mechanisms, secure communication protocols, a container orchestration platform, and a centralized logging and monitoring system are some of the suggested solutions.

5.4.7 Factory Method Design Pattern

The UML diagrams were converted into code using the Factory Method pattern. Customers will be able to order food for delivery, takeout, and dine-in, among other options, through the food ordering system. A particular handling rationale and set of prerequisites for every one of these orders was set. The Factory Method pattern was used to create a variety of objects to handle various orders by utilizing abstract classes. The Factory Method was employed to convert UML diagrams into Java code, and four phases were used, as shown below.

i. Define the Order interface or abstract class

```
public interface Order {
   void process();
}
```

ii. Implement the specific order types by extending the Order interface or abstract class

```
public class DineInOrder implements Order {
   public void process() {
```

```
        // Processing logic for dine-in order
    }
}
public class TakeoutOrder implements Order {
    public void process() {
        // Processing logic for takeout order
    }
}
public class DeliveryOrder implements Order {
    public void process() {
        // Processing logic for delivery order
    }
}
```

iii. A factory Method class that generates the appropriate order objects based on user input:

```
public class OrderFactory {
    public Order createOrder(String orderType) {
        Order order = null;
        if (orderType.equalsIgnoreCase("dine-in")) {
            order = new DineInOrder();
        } else if (orderType.equalsIgnoreCase("takeout")) {
            order = new TakeoutOrder();
        } else if (orderType.equalsIgnoreCase("delivery")) {
            order = new DeliveryOrder();
        }
        return order;
    }
}
```

iv. Use of the factory class to create order objects based on user input

```
OrderFactory orderFactory = new OrderFactory();
Order order = orderFactory.createOrder("delivery");
order.process();
```

Based on a common Order interface, the Factory Method pattern was used to create various Order objects, including Dine-In, Takeout, and Delivery Orders. The process () method are called on the returned Order object to carry out the particular processing logic for that order type. The OrderFactory class is used to generate the appropriate Order objects based on user input.

The food ordering and supply chain management system was made modular and flexible using the Factory Method pattern while maintaining abstract classes for all orders. This is vital as it increasing the system's maintainability and scalability.

5.5 Proposed Food Ordering Microservice Framework

The diagram below, in Figure 10, shows the proposed Food Ordering Microservice Framework, showing the relationships and interactions that the various microservices have with each other

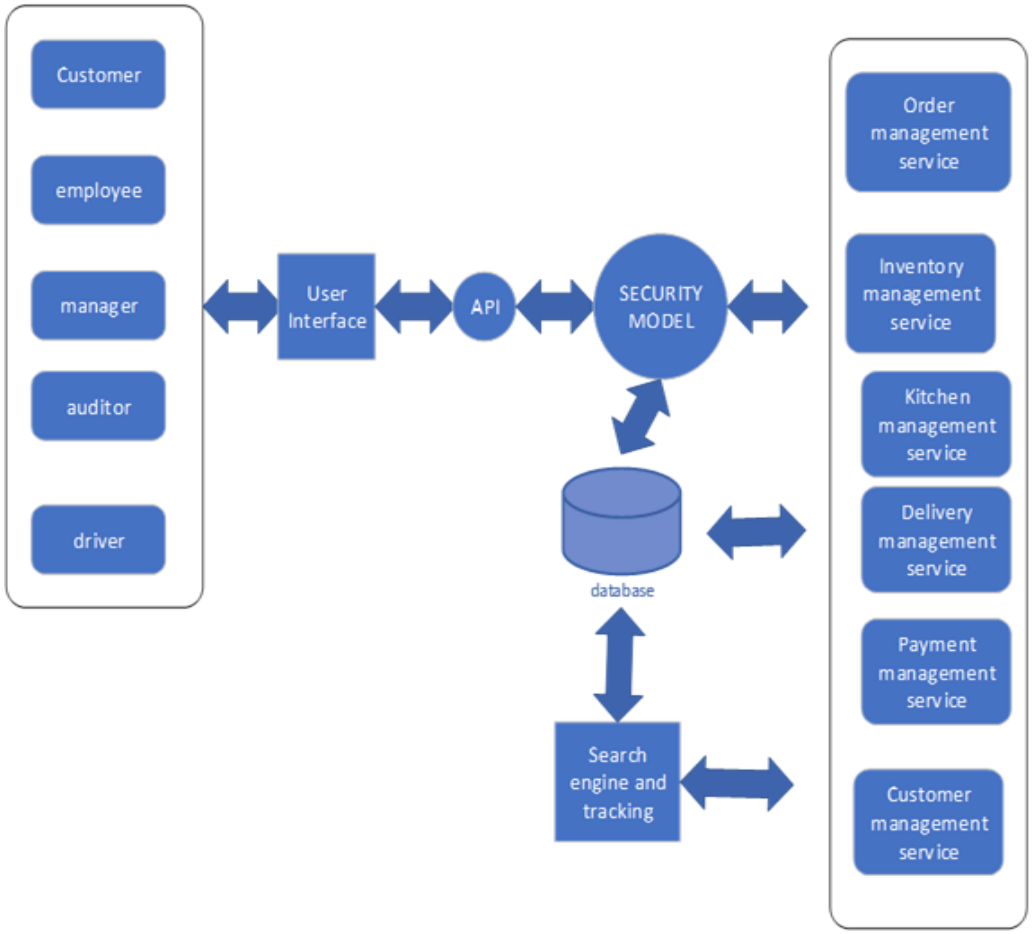

Figure 10. Proposed Food Ordering Microservice Framework Diagram.

Figure 10, above depicts the food ordering process, the interface to the system allows customers as well as Omega employees access to the system where customer places order on the portal or interface and this order is passed to the order management service. Several services are invoked to ensure the customer gets required service within the stipulated time.

5.5.1 Proposed Food Ordering Microservice Architecture Deployment

The Figure below, Figure 11 shows the Architecture Deployment diagram of the proposed Food Ordering Microservice Architecture, showing how the components interact and communicate. The figure depicts the various components for the order processing microservice architecture with user being able to use system on a tablet, or desktop using a web browser. This enables application to fit on either platform and improve usability. The usage of the application is authenticated through API gateway through various microservices.

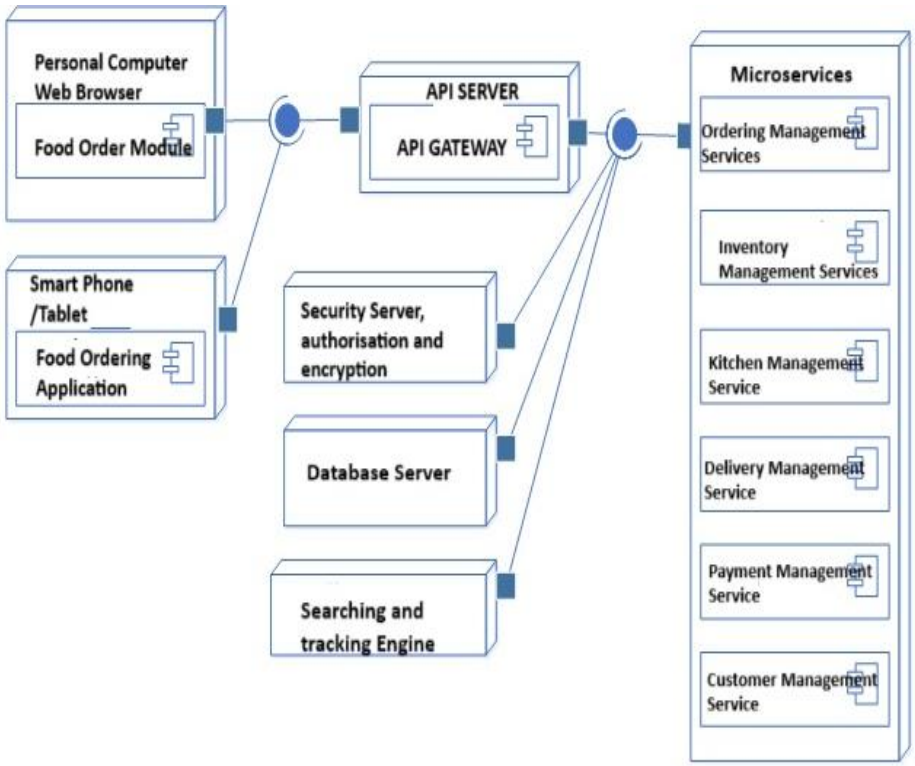

Figure 11. Proposed Food Ordering Microservice Architecture Deployment Diagram

5.5.2 SYSTEM IMPLEMENTATION PLAN

The food processing and ordering microservices architecture-based application will follow the aforementioned procedures.

1) For production infrastructure, use cloud services: Production infrastructure will be built using cloud services.

2) Plan for failure: Servers, or container nodes, must be treated as stateless entities that can independently stop, restart, and patch when microservices are designed and implemented. Applications must be able to handle component-level failures and tolerate failures of microservices components.

3) Make data management decentralized: When someone needs to update the database structure that other microservices rely on, they should not share the same database with the application teams for microservices (Dirgahayu, 2023).

4) Governance distributed: In order to successfully design and deploy microservices, it is necessary to establish a culture with minimally restrictive procedures and the obligation to recognize and address issues as they arise. Prioritize speed over effectiveness (Silva, et al., 2023). Due to the fact that the cloud service provider handles the majority of operational tasks, the infrastructure team ought to be small. Product-focused teams encourage quick thinking, clear communication, and coordinated efforts. The process of building the necessary API interfaces for process automation and microservices integration is also made simpler by this team structure.

5) Use CI/CD processes and automate the deployment of infrastructure: Microservices thrive in an agile DevOps environment that demands processes that are quick and free of friction. Automate processes so that a specific team member's actions trigger the necessary responses, such as updating a code module, generating software builds and tests, to reduce overhead and friction. Both rapid code development and evolutionary design are essential characteristics of microservices, and process automation encourages both.

6. FINDINGS AND RESULTS

In this section, the developed Food Ordering Microservice Framework, together with its components, will be explained in detail, looking at how they are contributing to the achievement of the operational objectives of Omega Food Services.

Firstly, there is the User Interface, which has the purpose of serving as the primary point of interaction for different user roles within Omega Food Services. The roles include customers, employees, managers, auditors, and delivery drivers. Role-based access will be implemented through role-based customization, whereby each user will only see the functionalities relevant to their tasks. A customer, for example, will interact with the order placement and tracking, while the kitchen staff can view order preparation details, and the managers can monitor inventory and performance.

The User interface will communicate with the backend through an API Gateway, enabling a seamless user experience without directly exposing individual microservices. This encapsulation protects the backend at the same time, allowing smooth, secure interactions for each user role. The Applicable Programming Interface (API) Gateway serves as a centralized access point through which all requests from the User Interface are directed to the appropriate microservices. The API is responsible for request routing and load Management. The API Gateway routes requests to specific services such as Order Management and Inventory Management. It also works by helping the microservice manage load by distributing traffic efficiently. The API Gateway makes sure that the system can handle high volumes of requests during peak times without compromising performance. The API Gateway plays a major role in securing the system by handling both user authentication and authorization. It makes sure that only authenticated users can access the services they are permitted to, based on their roles,

The Security Model manages all aspects of system security, including authentication, authorization, and data protection. It controls access at a granular level, ensuring that each user role has permission to access only the data and services that are relevant to them. The Food Ordering Microservice Framework offers centralised security, which will ensure that Omega Food Services's customer data and other sensitive information are protected. This will enable the organisation to meet the industry standards for privacy and data protection. The security model supports role-based access, which is essential for auditing and regulatory compliance.

The Database acts as a central repository that stores essential information for all microservices, such as order details, inventory levels, customer information, and transaction histories. The database ensures data consistency across all services. This setup is crucial for inventory management, real-time order tracking, and reporting. The database is designed to scale up in conjunction with the microservices, supporting high volumes of transactions during peak business hours and ensuring rapid response times.

Each core business function within Omega Food Services is encapsulated within its own independent microservice. The modular structure allows each service to be developed, deployed, and scaled separately, ensuring agility and operational resilience. The primary microservices are order, inventory, kitchen, delivery, payment, and customer Management. Order Management Service is responsible for handling all aspects of order processing, from order placement and validation to updating order status and managing cancellations. Usually, order volume fluctuates significantly; this service will be scaled independently to handle high demand without affecting other services. The isolation of the order management service functionality minimizes the impact of potential issues on other services.

The Inventory Management Service is responsible for managing stock levels, tracking inventory changes, and notifying relevant users about low stock. As the various types of orders are placed and processed, the inventory levels are updated in real-time, which will ensure that kitchen and management staff have a correct level of stock at hand. The service will trigger restocking alerts and manage inventory across multiple locations, helping Omega Food Services to avoid shortages and overstocking.

Kitchen Management Service is responsible for the order preparation process, providing kitchen staff with up-to-date order information and ensuring timely preparation. The isolation of kitchen operations makes the service to streamline workflow at the same time ensure that kitchen staff are not overwhelmed by other system functionalities. All of this directly improves order accuracy and speed. The Delivery

Management Service oversees the whole delivery process, from assigning orders to drivers to tracking delivery status in real time. The service provides real-time tracking to customers and staff, enabling them to view delivery status and estimated arrival times. The service also helps with route optimization, therefore improving delivery efficiency and customer satisfaction.

The Payment Management Service functions by processing payments securely, handling different payment methods, transaction records, and refunds. The isolation of payment processing is ideal as the service minimizes exposure of sensitive payment data to other parts of the system. This enhances security and compliance with financial regulations.

Customer Management Service is responsible for managing customer data, order history, and preferences, and providing a personalized experience. The service enables Omega Food Services to offer personalized promotions, loyalty programs, and faster service by maintaining a complete view of customer interactions.Search Engine and Tracking component supports fast, accurate searches for orders, customer records, and delivery tracking, improving the user experience for both the customers and staff.

Order Tracking is responsible for tracking customer orders in real time, its integrated with the Delivery Management Service to provide real-time updates to customers about their order status and delivery. Both drivers and customers stay informed, and it minimizes missed or delayed deliveries. The search engine and tracking component improves the customer service by enabling efficient searching and tracking, enabling Omega Food Service to provide better services and respond more quickly to customer inquiries.

This proposed Food Order Microservice Framework is designed to address the specific needs of Omega Food Service. The transition from a monolithic system to a highly scalable, efficient, and resilient microservices framework, and by modularizing key functions such as order management, inventory, kitchen, delivery, payments, and customer management, the system ensures that Omega Food Service will rapidly scale, adapt to changes, and improve operational efficiency. Enhanced security, user-focused interfaces, and optimized data handling will further enable the business to provide high-quality service at the same time, preparing for future growth. The Food Ordering Microservice architecture aligns well with Omega Food Services' long-term vision of improving service quality, increasing operational agility, and achieving robust scalability, which will directly lead to better customer satisfaction and business success.

7. DISCUSSION OF RESULTS

The Food Ordering Microservice Framework was reviewed by a panel of five expert Software Architects. Each expert gave feedback on the framework's design, structure, and functionality in relation to Omega Food Services' requirements. All five experts agreed that the framework is well-suited for its purpose. They mentioned that it is suitable to handle a growing customer base as it can offer continuous operations to all the different functional areas of the system.

The experts pinpointed the framework's modular architecture, which is made up of independent microservices, namely Order Management, Inventory Management, Kitchen Management, Delivery Management, Payment Management, and Customer Management. They stated that the software architecture and design of the microservice framework go hand in hand with the current software engineering practices that allow scalability, maintainability, and resilience. The security model that is integrated within the API layer was also mentioned. The experts stated that this was a good design as secure access to the database and all the core services will be provided, thus safeguarding customer and transaction data effectively.

The software architects pinpointed that the microservice provides enhanced functionality for both order tracking and customer engagement. The experts appreciated the scalability feature, which allows resource allocation to critical services during peak demand, making the system have optimised performance.

The Software Architects assessment reflects the framework's suitability and effectiveness for Omega Food Services. The feedback confirms that the Food Ordering Microservice Framework is suitable to meet current and future demands as it offers a scalable and secure solution. The developed Food Ordering Microservice Framework improves scalability, and it manages the growing customer base for Omega food service. This is because the framework allows the independent scaling of services. The Food Ordering Microservice Framework decomposes the monolithic system into smaller, independent, manageable services, namely Order, Inventory, kitchen, Delivery, Payment, and Customer Management. Each service can be scaled independently based on demand. During peak hours, for example, the Order Management and Delivery Management services can be scaled up without affecting other services such as Inventory or Payment Processing.

The Food Ordering Microservice Framework carries out efficient resource allocation. In the monolithic system, all components are interdependent, in this case the Framework will allow Omega Food Services to allocate resources more effectively. The Framework makes sure that only the services that are in demand in terms of requests and usage are only the ones consuming additional resources. This leads to cost savings and improved efficiency.

The Food Ordering Microservice Framework reduces downtime and, at the same time, improves performance. The isolation of services, which is done by the system, reduces performance bottlenecks that frequently occur in a monolithic structure when it’s under heavy load. The Food Ordering Microservice Framework can handle increased traffic without disrupting other areas of the system, which ensures better performance, faster response times, and minimized downtime.

The Food Ordering Microservice Framework has also enhanced flexibility for faster innovation and adaptation. The monolithic architecture has a nature of being tightly coupled, making it difficult to update or modify parts of the system without affecting the entire application. However, the Food Ordering Microservice Framework makes each service independent, allowing Omega Food service to update or modify specific services without causing system-wide downtime. Faster deployment of new features is enhanced. This is because the Framework permits continuous delivery and deployment practices, which allow Omega Food service to roll out updates and new features more quickly. This is possible because each service can be updated and deployed independently. Omega Food Service can respond more swiftly to market changes, customer demands, and technological advancements.

The Food Ordering Microservice Framework supports agile development practices, allowing Omega Food Services to experiment, test, and deploy new services or enhancements quickly. creates flexibility for the company as it can quickly adapt and fine-tune its systems to changing trends and customer expectations in real-time giving it a competitive advantage.

Due to the highly interconnected structure of monolithic systems a security breach in one area can compromise the entire system. The Food Ordering Microservice Framework enables each service to operate independently, this controls and limits the scope of a potential security breach. A failure or breach in one microservice, for example a breach in the Payment Management Service, will not directly impact other services in the Delivery Management. The Food Ordering Microservice Framework supports role-based access control, where each service has defined access rules. For example, only authorized users can access sensitive data in the Payment Management Service, and these rules are enforced independently across services. Food Ordering Microservice Framework enhances security by minimizing unauthorized access to critical parts of the system. The Security Model, which is integrated into the Framework, centralizes security management, enabling Omega Food Services to enforce consistent security policies across all services. The Food ordering Microservice architecture will have various security protocols embedded in it, namely authentication, authorization, and data encryption. These security protocols will ensure that each service complies with the up-to-date and current industry standards for data protection and privacy.

When updating or maintaining any part of the Monolithic system, it usually requires the entire application to be taken offline. This leads to lengthy downtimes. The Food Ordering Microservice Framework will enable Omega Food Services to deploy and maintain each service independently, greatly reducing the need for a system-wide downtime during updates or maintenance. The Framework allows for fault isolation, meaning that if one service fails, for example, let’s say the Inventory

Management Service fails, this will not bring down the entire system. The architecture exhibits system resilience as it allows other services to continue functioning while an issue is resolved, improving operational continuity.

The structure of a monolithic architecture, which has functions that are tightly coupled, makes it challenging to identify the root causes of issues. Each service in the Food Ordering Microservice Architecture is responsible for specific functions and tasks. This makes it easy to isolate and troubleshoot issues. This leads to fast troubleshooting and reduced time to resolve problems. Directly leads to lower downtime and increased system availability. The Food Ordering Microservice Framework enables Omega Food Service to use resources more efficiently. The services are deployed independently, and they can also be hosted on different servers or cloud environments based on their specific resource needs. This flexibility allows Omega Food Services to avoid over-provisioning of resources for services that do not require high performance. The scaling of services on demand rather than as a whole system will enable Omega Food Services to reduce operational costs, as resources will only be allocated where they are needed. Order Management and Payment Processing services, for example, may require additional resources during peak hours, while Inventory Management may not. The Food Ordering Microservice Framework promotes modular development. This will allow Omega Food Services to make incremental improvements to individual services without carrying out any extensive development and testing of the entire application. This modular approach not only reduces development time and costs but also enables efficient use of developer resources.

The Framework will reduce downtime at the same time improving system resilience, which ensures more reliable service delivery. Customers will experience fewer interruptions, thus enhancing their overall satisfaction. The Food Ordering Microservice Framework, combined with a real-time database and tracking system, will provide customers with timely and up-to-date order information, including delivery status and estimated arrival times. This transparency improves the customer experience by keeping them informed and engaged throughout the food ordering process. The independent services will allow for faster processing and response times. The Customer Management Service, for example, can handle customer-specific promotions and loyalty programs independently, enabling Omega Food Services to personalize its customer experience. Quick order processing and accurate delivery tracking will further contribute to improved customer satisfaction.

In conclusion, by transitioning to the Food Order Microservice Framework, Omega Food Services can overcome the limitations of its current monolithic system, positioning itself to grow, innovate, and at the same time adapt efficiently to evolving market demands and customer expectations. This new framework not only addresses immediate operational challenges but also provides a robust foundation for future growth and technological integration, ensuring that Omega Food Services remains competitive in the fast and ever-changing food service industry.

8. CONCLUSIONS AND FUTURE DIRECTIONS

The existing software architecture at Omega Food Service was assessed with the help of research literature, which clearly identified that the monolithic system has the following key limitations, namely scalability issues, limited flexibility for updates, slow development cycles, and high risk of failure. For example a bug in the inventory module can bring down the entire system, interrupting all its functionalities. The limitations of the monolithic approach justify the need to move over to a microservices architecture that can address scalability, flexibility, and robustness challenges.

The proposed Food Ordering Microservice Framework solves the identified limitations and aligns with Omega Food Services' specific operational needs. The framework offers modularisation of services by breaking down the entire system into independent microservices. The Framework directly addresses scalability and flexibility issues. Each service is autonomous; therefore, Omega Food Services can scale services independently based on demand. This selective scaling saves resources and reduces costs, ensuring that Omega Foods only allocates extra resources where necessary. Since each microservice operates independently, updates to one service can be deployed without affecting other services, thereby minimizing downtime and deployment risks. The microservice framework offers a user-centric interface, a centralized security model, optimized data handling with a database and search engine, operational efficiency, and resilience. The food ordering Microservice Framework design meets Omega

Food Services' needs for a more scalable, resilient, and agile system capable of supporting rapid business growth an improved user experience. Transitioning from a monolithic system to a microservices architecture is a complex process, predominantly in a live operational environment. An implementation plan was designed, which needs to be adhered to. The plan, which is a structured approach, allows Omega Food Services to mitigate risks, minimize disruptions, and ensure a seamless shift to the new architecture.

Microservices are widely studied; however, empirical evidence on their benefits specifically within food service applications, such as order processing and supply chain management, is limited. There is a need to carry out comprehensive empirical studies across multiple food service organizations to validate the benefits of microservices for scalability, deployment agility, and performance. Future research could include comparative studies with other architectures in similar industry contexts to provide a robust data set supporting microservices' advantages for food order processing. Insights could include order accuracy, service speed, inventory management efficiency, and overall customer satisfaction in real-world applications.

The existing metrics for evaluating microservices are often general and not tailored to the food service industry's unique operational needs, such as order accuracy, service speed, and customer satisfaction. There is a need to create and validate standardized performance metrics specifically designed to evaluate microservices in food service. These metrics could include order accuracy, service speed, customer satisfaction, operational resilience, and inventory turnover rates. Future research should focus on the development of a framework for collecting and analyzing these metrics, allowing organizations in the Food service industry to make data-driven decisions about microservices implementation and performance optimization.

9. REFERENCES

Abdiaziz, A. A. (2024). Microservice Architecture in E-commerce Platforms using Docker and Kubernetes. *Indian Scientific Journal of Research in Engineering and Management*.

Ahmad, H., Treude, C., Wagner, M., & Szabo, C. (2025). Towards resource-efficient reactive and proactive auto-scaling for microservice architectures. *Journal of Systems and Software*, *225*, 112390. https://doi.org/10.1016/j.jss.2025.112390

Alshuqayran, N., Ali, N., & Evans, R. (2018). Towards Micro Service Architecture Recovery: An Empirical Study. *2018 IEEE International Conference on Software Architecture (ICSA)*, 47–4709. https://doi.org/10.1109/ICSA.2018.00014

Alshuqayran, N., Ali, N., & Evans, R. (2025). A model-driven architecture approach for recovering microservice architectures: Defining and evaluating MiSAR. *Information and Software Technology*, *186*, 107808. https://doi.org/10.1016/j.infsof.2025.107808

Anggreainy, M. S., Setiawan, A. S., Subekti, M., Jingga, K., Noprianto, & Hartanto, J. (2021). Implementing Online Food Ordering System for Food Court Using Scrum Approach. *2021 25th International Computer Science and Engineering Conference (ICSEC)*, 351–356. https://doi.org/10.1109/ICSEC53205.2021.9684632

Anjum Era, C. A., Alvi, S. T., Rana, Md. S., & Mitu, S. J. (2025). A Comprehensive Study of Microservices Architecture in Comparison with SOA and Monolithic Models. *2025 2nd International Conference on Advanced Innovations in Smart Cities (ICAISC)*, 1–6. https://doi.org/10.1109/ICAISC64594.2025.10959671

Asrowardi, I., Putra, S. D., & Subyantoro, E. (2020). Designing microservice architectures for scalability and reliability in e-commerce. *Journal of Physics: Conference Series*, *1450*(1), 012077. https://doi.org/10.1088/1742-6596/1450/1/012077

Chen, A. C. H. (2023). Research on Efficiency Analysis of Microservices. *ArXiv. Https://Arxiv.Org/Abs/2303.15490*. https://doi.org/10.1109/ICMLANT59547.2023.10372984

Dirgahayu, T. (2023). *A framework for implementing enterprise architecture patterns as microservices*. 020040. https://doi.org/10.1063/5.0116376

Divyansh, K. (2024). Implementing Microservice Architecture in E-Commerce with DevOps Practice. *International Conference on Computer Science and Technology (JCSTS)*.

Genfer, P., Serbout, S., Simhandl, G., Zdun, U., & Pautasso, C. (2025). Understanding security tactics in microservice APIs using annotated software architecture decomposition models – a controlled experiment. *Empirical Software Engineering*, *30*(3), 66. https://doi.org/10.1007/s10664-024-10601-1

Hossen, M. R., & Islam, M. A. (2023). Practical Efficient Microservice Autoscaling. *ACM SIGMETRICS Performance Evaluation Review*, *50*(4), 50–52. https://doi.org/10.1145/3595244.3595262

Ileana, M., Oproiu, M. I., & Marian, C. V. (2024). E-commerce Solutions using Distributed Web Systems with Microservices-Based Architecture for High-Performance Online Stores. *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, 994–999. https://doi.org/10.1109/MIPRO60963.2024.10569273

Karan Dhiman, Mayuresh Phansikar, Mohamed Zain Kazi, & Rutuja Thakur. (2022). A Web-Based Service Marketplace. *International Research Journal of Engineering and Technology (IRJET)*.

Lee, W.-T., Song, P.-Y., Tsai, M.-K., Wu, M.-H., & Ma, S.-P. (2023). A High Availability Microservices Architecture Implementation using Saga and Backup Mechanism. *2023 10th International Conference on Dependable Systems and Their Applications (DSA)*, 470–471. https://doi.org/10.1109/DSA59317.2023.00063

Li, J. (2025). Research On Optimization Model of High Availability and Flexibility of Blockchain System Based on Microservice Architecture. *Procedia Computer Science*, *261*, 207–216. https://doi.org/10.1016/j.procs.2025.04.191

Mosafi, F. F. D. S., Cosenza, M. S., Vasconcelos, L. V. C., Pires, L. F., Duarte, J. C., & Claudia Reis Cavalcanti, M. (2025). Performance Evaluation of Monolithic and Microservice Architectures for Natural Language Processing in Command and Control Applications. *IEEE Access*, *13*, 155447–155461. https://doi.org/10.1109/ACCESS.2025.3604775

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

Niño-Martínez, V. M., Ocharán-Hernández, J. O., Limón, X., & Pérez-Arriaga, J. C. (2023). Microservice Deployment. *Proceedings of the Institute for System Programming of the RAS*, *35*(1), 57–72. https://doi.org/10.15514/ISPRAS-2023-35(1)-4

Paulo Francisco. (2020). Migrating from Monolithic Architecture to Microservices: A Rapid Review. *IEEE Latin America Transactions*, *18*(3).

Qian, L., Chen, H., Yu, J., Zhu, G., Zhu, J., Ren, C., Mei, Z., Pang, H., Xu, M., & Wang, L. (2020). Research on Micro Service Architecture of Power Information System Based on Docker Container. *IOP Conference Series: Earth and Environmental Science*, *440*(3), 032147. https://doi.org/10.1088/1755-1315/440/3/032147

Rasheedh, J. A., & Saradha, S. (2020). An optimal micro-services architecture for runtime dynamic binding in web services using hybrid heuristic algorithms. *International Journal of Advanced Science and Technology*.

Rizki, A. S., Bambang, P., Widyastuti, A., & Sri, R. C. N. (2024). Microservices Architecture in Point of Sales Application Based on Restful API and Webhook. *Journal of Intelligent Software Systems*, *3*(1), 31–35.

Rodrigues, H., Rito Silva, A., & Avritzer, A. (2025). Assessment of performance and its scalability in microservice architectures: Systematic literature review. *Journal of Systems and Software*, *230*, 112500. https://doi.org/10.1016/j.jss.2025.112500

Samoylenko, H. T., & Selivanova, A. V. (2023). Features of microservices architecture in e-commerce systems. *Mathematical Machines and Systems*, *3*, 51–58. https://doi.org/10.34121/1028-9763-2023-3-51-58

Schwarz, G.-D., Bauer, A., Riehle, D., & Harutyunyan, N. (2025). A taxonomy of microservice integration techniques. *Information and Software Technology*, *183*, 107723. https://doi.org/10.1016/j.infsof.2025.107723

Seedat, M., Abbas, Q., Ahmad, N., & Amelio, A. (2024). Systematic Mapping of Monolithic Applications to Microservices Architecture. *VAWKUM Transactions on Computer Sciences*, *12*(1), 94–110. https://doi.org/10.22541/au.168110476.68608378/v1

Thota, K. (2025). Strategic Infrastructure Modernization in the Food Industry. In *Modernizing the Food Industry* (pp. 1–22). IGI Global Scientific Publishing. https://doi.org/10.4018/979-8-3373-5288-6.ch001

Vangala, S. R., Mallidi, R. K., & Appili, V. L. P. (2023). Micro-services Transactions Resilience Across Bounded Domains: An Architecture Perspective. *International Journal of Computer Applications*, *184*(45), 30–35. https://doi.org/10.5120/ijca2023922557

ZargarAzad, M., & Ashtiani, M. (2023). An Auto-Scaling Approach for Microservices in Cloud Computing Environments. *Journal of Grid Computing*, *21*(4), 73. https://doi.org/10.1007/s10723-023-09713-7