Authentication and Role-Based Authorization in Microservice Architecture: A Generic Performance-Centric Design

Randa Ahmad Al-Wadi * and Adi A. Maaita

Department of Software Engineering, Isra University, Amman, Jordan; Email: adi.maaita@iu.edu.jo (A.A.M.)
*Correspondence: randa.it.alwadi@gmail.com (R.A.A.W.)

***Abstract*—In a microservice-based system, each microservice is a stand-alone application that may be targeted individually to obtain unauthorized access. Consequently, it is necessary to include authentication and authorization features. However, a set of related design decisions needs to be taken in a way that accommodates the scale of a developed system. To illustrate, a user may be authenticated depending on a password and authorized based on roles. In such a case, one integrated authentication and role-based authorization microservice can be added. Besides, the Application Programming Interfaces (APIs) that are associated with roles may be hard-coded as static API-level role authorization checks. Nevertheless, static relation between roles and APIs hinders the ease of modification of their associations when a massive number of APIs exist in a microservice system. To transform the relation into dynamic relation, this paper presents a generic microservice-based architectural design with a separate role-based authorization microservice that contains role/API database records. Moreover, it shows experimentation for performance optimization that was carried out on authentication and role-based authorization databases to utilize the suggested architectural design. The obtained results of password-based authentication encouraged employing not only Structured Query Language (NoSQL) databases with small microservice-based systems, which deal with 1500 users or less while employing Structured Query Language (SQL) databases with medium to large systems. Furthermore, the results indicated that there is no difference between the two database types in the role-based authorization process for all API-based system scale levels.**

***Keywords*—microservices, architectural design, security, password-based authentication, role-based authorization, Application Programming Interface (API), Structured Query Language (SQL), not only SQL (NoSQL), response time**

I. Introduction

In the area of software architecture design, there is a continuous need for devising new architecting paradigms to accommodate some emerging requirements. Generally, the direction is towards the "Separation of Concerns" concept which facilitates the software development process and raises maintainability. This concept led architects to shift from the monolithic architecture to Service-Oriented Architecture (SOA), then to microservice architecture to adapt to particular circumstances [1].

The monolithic architecture implies building applications as a single deployable unit. In contrast, SOA enforces some level of independence by splitting up functionalities into "services" but with central governance and data storage. Recently, microservice architecture emerged as a natural evolution to maximize the loose-coupling concept by eliminating centralization. This enables microservices to be highly maintainable and highly scalable [2, 3].

Microservice-based systems, as distributed systems, require careful attention to each related security consideration [1]. This is due to the broadness of their attack surface which arises from the existence of many small independent and distributed services that are accessible by various client applications [4, 5]. To secure such systems, it is not sufficient to protect the perimeter only. Instead, it is substantial to protect every microservice inside the perimeter by enforcing security mechanisms by using security modules. That involves safeguarding microservices from the blind trust of clients or other microservices that request access to their functionalities [6, 7].

Enforcing the authentication and authorization security requirements is considered an essential way of achieving protection and building the required conscious trust [6, 7]. Authentication is the process of checking the identity of a user whereas authorization is the process of granting the user permission to perform specific actions or reach certain resources [8, 9].

In the literature, different microservice-based architectural designs aimed to satisfy authentication and authorization characteristics. Focusing on the architectures that implement features of authentication and role-based authorization, it was found that the most adopted design idea is to add Identity microservice [10–13]. Identity microservice is responsible for authenticating a user based on a username and a password combination and issuing an access token

Manuscript received January 13, 2023; revised March 15, 2023; accepted April 26, 2023; published August 3, 2023.

for role-based authorization at the business microservices level. In addition, it manages all user-related data such as roles. The issue with this design is that authorization data cannot be stored since Identity microservice is only concerned with user data. As a specific example, user/role database records can be included in the Identity database while role/Application Programming Interface (API) database records cannot. That provides the ability to change users' roles dynamically, whereas modifying role-related APIs can only be done statically via hard coding at API-level. Hard coding of permissions is a common practice among developers [14], however, it is not recommended whenever the number of APIs is substantial. To overcome this limitation, an extended microservice architectural design was proposed that includes a separate role-based authorization microservice with an attached database of role/API associations.

Merging authentication and authorization characteristics with a microservice-based system without degrading its performance, in terms of response time, is a crucial point [15]. Therefore, the suggested architecture can be customized based on design decisions related to database type and design. As part of this research, suggestions for achieving the best performance are presented based on a comparison between Structured Query Language (SQL), and not only SQL (NoSQL) databases with an experimentation phase to determine the performance level of each during the password-based authentication and role-based authorization processes.

Security and performance are two important research aspects in the field of microservice architecture design. According to Vural's study [16], there is still a lack of related empirical studies which are greatly needed for extracting valuable information for making design decisions. This paper contributes to that branch of microservice research by conducting an experimental study to help resolve security and performance issues.

The rest of the paper is organized as follows. Section II analyzes the state of the art in microservice architectural designs that have inherent authentication and authorization features. Sections III–VI explain the implementation details of the suggested microservice architecture. These involve its architectural design, the SQL and NoSQL database structures of the password-based authentication and role-based authorization microservices, and a description of the authentication and authorization processes. Section VII explores the performance testing-related aspects of the architecture including the test scenarios and test cases, the hardware specifications, and an explanation of the data collection and analysis procedure. In addition, it presents the steps taken to achieve the reliability and validity of measurements as well as the experimental limitations. Section VIII shows and analyzes the obtained results from carrying out the test cases. Then, Section IX interprets the analyzed results. And finally, Section X clarifies the conclusions that were drawn.

II. Literature Review

A range of architectural design decisions aimed at achieving authentication and authorization security requirements within a microservice architecture can be found in the literature. For example, several architectural solutions employ a component called "API gateway" to play a specific role in the authentication and authorization processes either as the main entity or simply as a facilitator [4–6, 10].

The API gateway acts as a single-entry point that mediates the communication between clients and microservices. The aims are to forbid malicious or unauthorized access [4, 10] and to minimize the number of calls from clients to microservices, or in other words, to solve the problem of "chatty communication" [6, 10]. Due to the benefits that the API gateway provides, it is seen as a substantial component in any microservice architecture [17]. Besides this, "having no API gateway" is considered, between researchers and practitioners, an anti-pattern or bad design practice [18, 19].

Alternatively, several architectures involve implementing authentication and authorization as a dedicated microservice. Microservices usually serve the business functionalities of a system. Nevertheless, using a dedicated microservice for security purposes showed the ability and benefit of employing microservices in achieving cross-cutting concerns as well.

To illustrate, He and Yang [12], and Sharma [13] dedicated a microservice to represent an authentication and authorization server. Authentication was achieved by taking the responsibility of checking the credentials of users while authorization was involved by issuing access tokens. Access tokens are stand-alone pieces of information that allow every business microservice to make its authorization checks for granting access to its functionalities. Such an approach served as an alternative to going back and forth with requests to the central dedicated authentication and authorization microservice. Another goal was to allow users to sign in only once and then use the generated tokens as indicators of authentication in subsequent requests. Fig. 1 describes that suggested architecture.

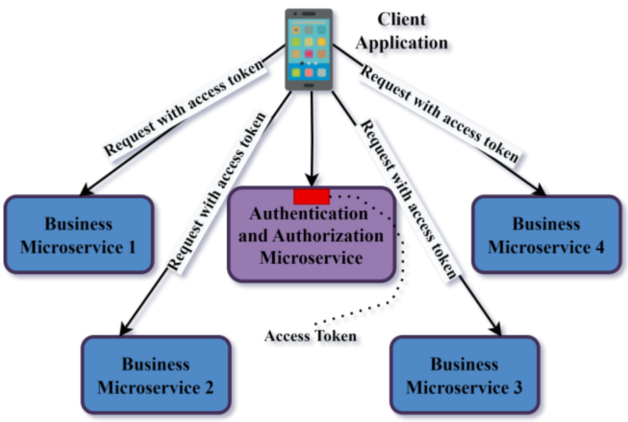
Client Application
Request with access token
Request with access token
Request with access token
Request with access token
Business Microservice 1
Authentication and Authorization Microservice
Business Microservice 4
Access Token
Business Microservice 2
Business Microservice 3

Figure 1. Authentication and authorization as a microservice.

Looking at industry practices in architecting, the mentioned architecture was adopted by leading companies such as Microsoft [10] and Oracle [11]. However, Microsoft renamed the dedicated security microservice to be "Identity Microservice". The reason is to reflect its task of managing the identities of users. Identity microservice has three responsibilities: storing the identities, verifying their correctness through the authentication process, and generating access tokens filled with identity information.

Furthermore, Microsoft [10] introduced another improved architecture as well. It is a design that combines Identity Microservice with an API gateway. One purpose for this integration is to optionally add two layers of authentication and authorization instead of one to a microservice-based system. The first layer is the mandatory authentication through Identity Microservice which includes user login and obtaining an authorization access token, while the second layer is the optional API gateway token-based authentication and claim-based authorization. Claim-based authorization is an authorization type that depends on what claims are included in the token attached to a sent request such as the user role.

By analyzing Refs. [10–13], a limitation related to role-based authorization was found. As discussed, Identity Microservice database includes information related to users including their roles. When a user logs in, his/her role(s) are filled into the issued access token to be evaluated against static API-level authorization attributes such as [Authorize (Roles = "Admin")]. Static authorization attributes imply that each API has a hard-coded role authorization check, which allows or prevents user access [14]. Hard coding of associations between APIs and roles is acceptable while the number of APIs is small, however, as the number increases, such an approach becomes tedious and difficult to manage. In order to better manage the latter case, this paper suggests adding database records of role/API associations for enabling dynamic re-association.

Since the software architecture community promotes the Separation of Concerns concept [1], this paper suggests separating authentication and role-based authorization into two microservices with two distinct databases. Authentication microservice database manages user-related data, while role-based authorization microservice manages authorization-related role data that is represented as role/API associations.

In addition, and as an attempt to preserve a satisfactory performance level of password-based and role-based authorization processes, related database design decisions have been included as part of the architecture. These design decisions determine the most efficient database type to be used along with its design. This is based on comparing SQL and NoSQL database performance, in terms of response time, within the context of microservice architecture.

III. Architectural Design

Selecting the right components that comprise a system and appropriately connecting them serve to fulfill specific quality attributes. This paper suggests an architectural design that achieves password-based authentication and role-based authorization security requirements with a focus on performance. Upon surveying the literature and observing the contributions of practitioners in the field, the proposed microservice architecture was structured as presented in Fig. 2. This architecture is comprised of the following components:

Business Microservices are provider applications that deliver the main services intended by a system.

Authentication Microservice is a security component that manages user data, authenticates user identities by validating the usernames and passwords, and issues access tokens.

Role-Based Authorization Microservice is a security component that stores roles in a system, information about the system's APIs, and the relationships between the roles and the APIs that represent the allowed list of APIs for each role.

RabbitMQ Message Broker is a queue-based communication component that enables asynchronous interaction between Business Microservices as well as between Authentication and Authorization Microservices.

Client Application is a consumer application that sends requests to Business Microservices asking for specific services.

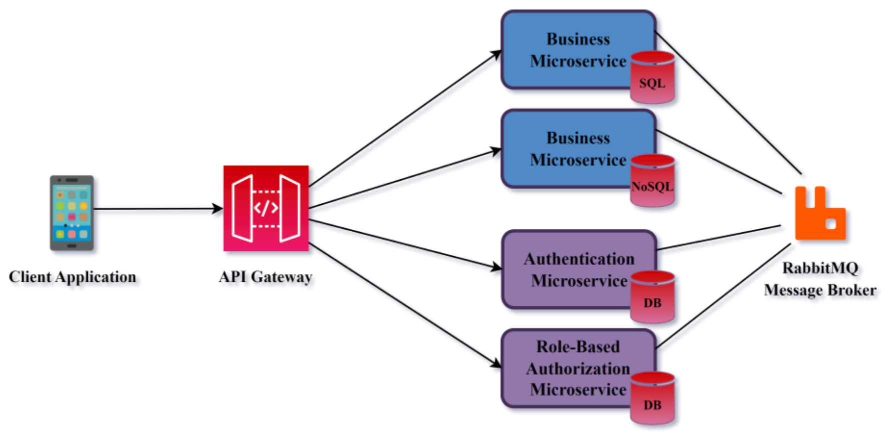

Figure 2. The proposed microservice architectural design.

API Gateway is a mediator communication component that facilitates the interaction between Client Application and microservices. Furthermore, it is a security component that can be activated to make initial token-based authentication and authorization.

From Fig. 2 once again, it is noticed that one of the Business Microservices is connected to a SQL database while the other is connected to a NoSQL database. This reflects the diversity and heterogeneity that can be found in any microservice-based system, which relates to the nature of each Business Microservice. In contrast, the database technologies utilized for each of the Authentication and Authorization Microservices are not determined. These microservices deal with some cross-cutting concerns that all microservice-based system components rely on. Selecting the right database technology to use with these cross-cutting concerns microservices cannot be arbitrary and such a decision requires proper justification.

From the performance perspective, SQL and NoSQL technologies may have a positive or negative effect on authentication and authorization processes. To make the suggested architectural design performance-centric, it was important to include database architectural design decisions that are related to the Authentication and Authorization Microservices. These decisions help architects in selecting the most efficient database type proved by experimentation along with its design. The next two sections present the SQL and NoSQL database structures which were employed in the experimentation phase.

IV. SQL Database Schemas

SQL database schemas are commonly utilized in implementing Authentication and Authorization Microservices. By reviewing the SQL database schema provided by ASP.NET Core Identity [20], the minimum requirements for an authentication microservice were extracted, which are shown in Fig. 3. The experiments in this paper depended on the full schema of ASP.NET Core Identity but what appears in the figure is a suggestion for architects who are looking for a simplified design solution.

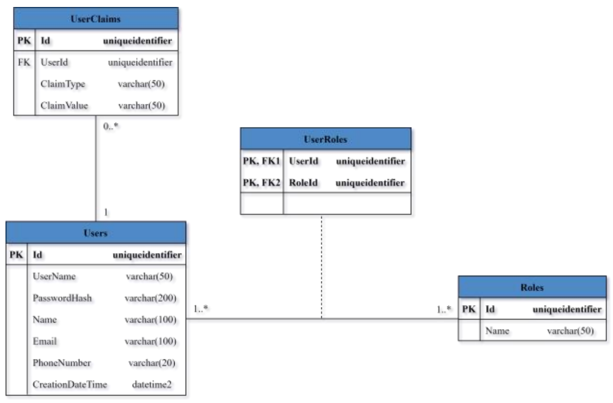
UserClaims
PK | Id | uniqueidentifier
FK | UserId | uniqueidentifier
ClaimType | varchar(50)
ClaimValue | varchar(50)
0..*
1
UserRoles
PK, FK1 | UserId | uniqueidentifier
PK, FK2 | RoleId | uniqueidentifier
Users
PK | Id | uniqueidentifier
UserName | varchar(50)
PasswordHash | varchar(200)
Name | varchar(100)
Email | varchar(100)
PhoneNumber | varchar(20)
CreationDateTime | datetime2
1..*
1..*
Roles
PK | Id | uniqueidentifier
Name | varchar(50)

Figure 3. A basic SQL schema for an authentication microservice.

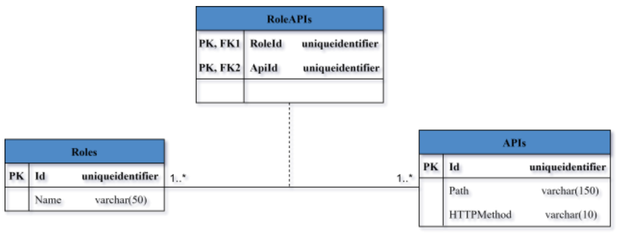
RoleAPIs
PK, FK1 | RoleId | uniqueidentifier
PK, FK2 | ApiId | uniqueidentifier
Roles
PK | Id | uniqueidentifier
Name | varchar(50)
1..*
1..*
APIs
PK | Id | uniqueidentifier
Path | varchar(150)
HTTPMethod | varchar(10)

Figure 4. The SQL schema for the role-based authorization.

The extracted schema involves three main tables which are Users, Roles, and UserClaims. The Users table stores user identity data. The Roles table defines the different available roles for users in a system. The association between these users and roles is represented in a subsidiary table named UserRoles. Besides, the UserClaims table keeps user claims to be used in the access tokens generation process. Typically, the essential claim in the suggested microservice architecture is the role. Therefore, it is mandatory to be included within access tokens as a preparation for the role-based authorization process.

Each role in a system is supposed to be a key for enabling users to access a specific range of APIs. Hence, the role-based authorization SQL schema was structured to be as clarified in Fig. 4.

By comparing Figs. 3–4, it is found that both schemas have a Roles table. The two tables, in fact, are identical in content, i.e., they store the same roles existing in a system. However, the main table is the one owned by Authorization Microservice while the other is a subordinate. What is meant by “subordinate” is that it receives a data copy from Authorization Microservice through the RabbitMQ message broker component appearing in Fig. 2.

Authentication Microservice’s responsibility is to deal with the identities of users whereas the responsibility of Authorization Microservice is to preserve the authorization-related data including roles and APIs, thus, employing the principle of Separation of Concerns.

V. NoSQL Database Collections

Simply, the authentication SQL schema presented in Fig. 3 can be transformed into two NoSQL collections with the same data fields. The first collection is Users, which is the union of the three SQL tables: Users, UserRoles, and UserClaims. It holds the identity data of users as well as their roles and claims. Whereas the second collection replaces the Roles SQL table that maintains the roles supported by a system and can be assigned to users. This structure is shown in Fig. 5.

Regarding the role-based authorization process, its database can be implemented in NoSQL as appears in Fig. 6. To clarify, the Roles collection holds a system’s roles with their associated APIs. On the other hand, the full information of the APIs that are exposed by a system is stored in the APIs collection.

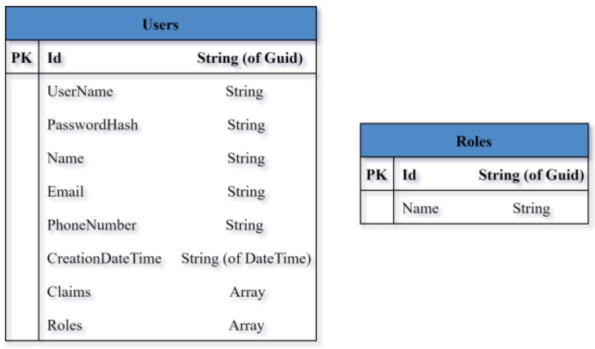

Figure 5. Authentication’s NoSQL database collections.

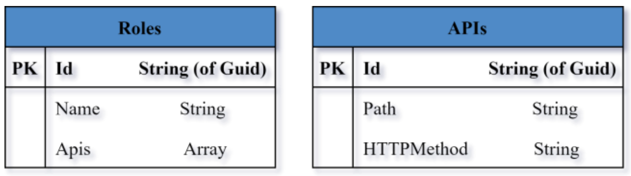

Figure 6. The role-based authorization process’s NoSQL collections.

VI. Authentication and Authorization Processes

To allow a user to access a protected resource, the user needs to pass two layers of authentication and one layer of authorization. The first authentication level is password-based authentication conducted by Authentication Microservice. It requires the user to send his/her credentials to the login API to be verified. The verification includes accessing the Authentication Microservice’s database to look for a matching username and an identical password hash. And then, in case of success, generating a token that is embedded with the user’s claims which are retrieved from the same database.

Using the SQL or NoSQL technology to complete this process may make a difference in terms of performance. Thus, it was decided to include password-based authentication in the experiments to see which database technology performs better in this context.

The second level of authentication is token-based. This is performed by any microservice that exposes protected APIs. For more explanation, when a protected API is called, the responsible microservice checks whether an access token is attached to the request or not. If a token is found, the microservice verifies that token against some criteria such as the issuer of the token as well as the expiration date. Finally, if all tests are passed, the procedure of token-based authorization commences. The experimentation stage was not concerned about the second layer of authentication because database access is not required. However, it was planned to test the authorization process.

The token-based authorization in the suggested microservice architecture is the role-based authorization process, which the SQL and NoSQL databases’ structures were prepared for in the previous two sections. It is a procedure performed whenever an HTTP request arrives by the Authorization middleware that is found in each microservice.

The first step in this process is extracting the user’s roles from the token. Then, slicing the HTTP request to find the request path and the HTTP method, i.e., determining the information of the requested API. The third step is to find an association between the roles of the user and that API in the Authorization Microservice’s database. And finally, allowing the user to access the functionality provided by the API or return a refusal in case of failure.

Similar to password-based authentication, employing a SQL or NoSQL database to be accessed during the role-based authorization may present different performance levels. To make a proven comparison, an experimentation phase is required. In the next section, all the aspects of the conducted database performance tests will be discussed in detail.

VII. Research Method

In this section, the test scenarios and test cases that were followed to execute the performance testing on the suggested microservice architecture will be clarified. Also, the specifications of the machine that was employed in carrying out those test cases will be listed. Furthermore, the data collection and analysis procedure as well as how the reliability and validity of measurements were assured will be explained. Finally, the limitations of experimentation will be mentioned.

A. Test Scenarios and Test Cases

The experiments on the proposed microservice architecture include two parts. One part is to measure the time needed to authenticate a user based on the password-based authentication process. The second part is to determine how long it takes to authorize a user depending on the role-based authorization process. Table I describes the test scenarios that were conducted—it is worth mentioning that the tables’ structure in this section is inspired by the work provided by Fahrurazi, Ibrahim, and Suffian [21].

TABLE I. The Test Scenarios of the Proposed Microservice Architecture

Test Scenario ID	Scenario	Test Objects or Subsystems	Test Items
TS1	Response time for authentication	Authentication Microservice	Login API & Authentication database
TS2	Response time for role-based authorization	Authorization Microservice & any microservice that has APIs	Authorization database & Authorization middleware

Determining test scenarios is not sufficient for conducting research experiments. Therefore, they were supplemented by two test suites, one for each scenario. Table II shows the first test suite that is dedicated for testing the scenario of authentication. It contains five test cases, each of which is intended to test, in terms of performance, the authentication of one virtual user against a specific number of stored user records. It is noticeable that the number of these records increases exponentially.

The test cases in Table II begin with one stored user only to represent a baseline performance measurement. This baseline forms a starting point for the measurements of response time. For more explanation, the value of the baseline does not include the time consumed in the search action. Instead, it is related to the procedure of authenticating a user by directly accessing his/her record in the authentication database.

TABLE II. THE AUTHENTICATION PROCESS' TEST CASES

Test Scenario ID	Test Case ID	Authentication Scheme	Number of Virtual Users	Number of Stored Users
TS1	TS1_TC1	Password-Based	1	1
	TS1_TC2			10
	TS1_TC3			100
	TS1_TC4			1000
	TS1_TC5			10,000

In the second test case and beyond, the number of stored records rises exponentially by a factor of ten. Usually, in performance testing, the baseline is determined, then it is doubled, tripled, or multiplied by a larger factor [22]. The factor of ten was chosen here to reach huge record numbers, which reflect a realistic situation in the microservice architecture, with minimum numbers of test cases. In other words, to reduce the performance testing cost.

Table III, on the other hand, clarifies the second test suite that was created for testing the scenario of role-based authorization. This test suite involves ten test cases that are divided into two groups. The first consists of four test cases that test the performance of authorizing a user when the authorization database is filled with exponentially increasing APIs records that are linked to one role only in the system. Whereas, the second is comprised of six test cases that are similar to what exists in the first group but with two roles, each of which is taking half of the stored APIs.

As general remarks regarding the test cases of the second test scenario, firstly, the exponential rise in the number of stored APIs stops at one thousand in the first part of these test cases. This is because it is not possible to find ten thousand APIs, for instance, in a microservice-based system or even more than one thousand in such a system that contains one role only. In the second part of the test cases, two further points after one thousand were defined since the system now supports more than one role. Nevertheless, the increase stopped at three thousand records due to the unrealism of the existence of more than this number in a microservice-based system.

It is important to mention that both test suites, explained in this section, were executed against the SQL technology one time and the NoSQL technology another time. The reason is the analysis of the results will be comparative in the first place. The purpose of this is to be able to select the appropriate database technology for completing the password-based authentication and role-based authorization processes in a microservice-based system.

B. Hardware Specifications

To execute the experimentation's test cases, a laptop personal computer with the following specifications was used:

- Intel Core i5-4210U CPU @ 1.70GHz.
- 8GB RAM.
- 64-bit Operating System.

C. Data Collection and Analysis

Research experimentations are concentrated on collecting data about time intervals required to complete the processes of authentication and authorization. Consequently, timers were adjusted to take these measurements, in milliseconds as integers. Also, pieces of code were written to embed the taken values into result records. The result records are of a previously defined structure, and they are part of a dedicated database for performance testing results.

To obtain the performance testing results, each prepared test case was carried out more than one time against each database technology. In addition, multiple readings were taken and stored in the performance testing database during each test. This is to ensure the trustworthiness of the collected values. For analyzing the gathered data, the average of the readings was calculated, for each test case and each database technology separately, and anomalies were excluded depending on the standard deviation method.

TABLE III. THE ROLE-BASED AUTHORIZATION PROCESS' TEST CASES

Test Scenario ID	Test Case ID	Authorization Scheme	Number of Virtual Users	Number of Stored Roles	Number of Stored APIs	Num of Linked APIs to each Role
TS2	TS2_TC1	Token-Based	1	1	1	All of the APIs
	TS2_TC2				10	
	TS2_TC3				100	
	TS2_TC4				1000	
	TS2_TC5			2 (The virtual user is assigned to one of these roles solely)	2	Half of the APIs
	TS2_TC6				10	
	TS2_TC7				100	
	TS2_TC8				1000	
	TS2_TC9				2000	
	TS2_TC10				3000	

D. Reliability and Validity of Measurements

In any quantitative research, reliability and validity are considered two fundamental criteria that make the measurements trustworthy. Reliability is related to the consistency of results, which indicates that the obtained results must be identical whenever an experiment is repeated under the same circumstances. Whereas, validity is related to the accuracy or the closeness of the collected data to the truth [23].

To assure these two criteria, as much as possible, it was planned to achieve the following:

- Conducting the experiments multiple times and storing several readings during each of them.
- Eliminating anomalous values based on the standard deviation method.
- Testing the worst cases, i.e., the cases of reaching the last record or document in the databases. This is to get accurate comparative results regarding the performance of SQL and NoSQL.
- Executing a test case on the SQL and NoSQL databases in turn instead of carrying out all the test cases against one technology and then all of them against each other. The goal is to keep the ratio of the taken values between the SQL and NoSQL consistent. This is in case the performance of the computer used in testing degrades at a point in time for some reason.

E. Experimental Limitations

The following represent two major limitations of this research:

- **Sample size.** The computer that was used in the development and performing testing is a personal computer with limited resources. As a result, it was hard to generate more than 10,000 records using this computer to fill a database table in preparation for conducting experiments. Consequently, tests on huge numbers, 100,000 or 1,000,000, for instance, that reflect some real situations could not be executed.
- **Time limitation.** The time that was dedicated for completing all the aspects of this research was limited. Thus, the number of cases to be studied and tested was minimized. Besides, that led to a partial implementation of the microservice architecture but with enough details to make conducting experiments possible.

VIII. Results

In this section, the experiments' results of the process of password-based authentication and role-based authorization will be presented and analyzed.

A. General Analytical View

The experimentation in this research is comparative in nature, i.e., it compares the performance level, in terms of response time, of the SQL and NoSQL database technologies in certain contexts within the microservice architecture. Therefore, the relative relationship between the SQL and NoSQL resulting time values is more significant than the absolute time values themselves.

Besides, it is important to mention that the testing environment does not reflect a realistic deployment environment. Consequently, some unexpected behaviors in the results' curves may be noticed. To illustrate, an enhancement in performance instead of decay, a fluctuation, or semi-fluctuation may appear with the increase in the stored record numbers in the databases. Those matters were considered to give correct interpretations of the results at the end of the research.

B. Password-Based Authentication

This branch of experimentation focused on measuring the required time to authenticate one virtual user when the SQL and NoSQL authentication databases are filled with a particular number of users' records. By looking at the SQL part of the Authentication Time column in Table IV, it is seen that the numbers of the five experiments are, nearly, close to each other and there is no considerable increase or decrease. In other words, there was general stability in performance. However, when the NoSQL part is scanned, a disparity between numbers is noticed besides an obvious performance drop in the last test.

TABLE IV. The Results of SQL and NOSQL Performance Testing on Authenticating One Virtual User Based on a Password

Test Scenario ID	Test Case ID	Number of Stored Users	Database Technology	Authentication Time (in milliseconds)
TS1	TS1_TC1	1	SQL	195.78
	TS1_TC2	10		189.34
	TS1_TC3	100		189.22
	TS1_TC4	1000		182.86
	TS1_TC5	10,000		187.40
	TS1_TC1	1	NoSQL	176.63
	TS1_TC2	10		169.11
	TS1_TC3	100		155.88
	TS1_TC4	1000		171.03
	TS1_TC5	10,000		248.97

To study the behaviors over experiments and compare the performance of SQL and NoSQL database technologies, the results were illustrated as a chart as appears in Fig. 7. By tracking the SQL line, which is in blue color, the performance stability, which was observed before, becomes assured. In contrast, the NoSQL line, which is in green color, shows different behavior. In the beginning, the values were going down, then, a small rise occurred followed by a significant one. That reflects huge degradation in performance affected by the increase in the number of stored users in the system.

When the performance of SQL is compared to NoSQL, by observing Fig. 7 again, it is realized that NoSQL shows better performance in password-based authentication with small numbers of users. Specifically, when the number of stored records is around two thousand or less. However, SQL is superior in the case that user records numbers exceed the two thousand limit.

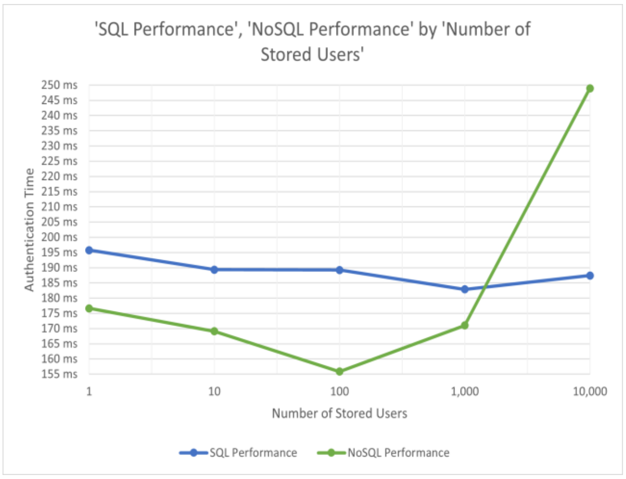

Figure 7. The SQL performance against the NoSQL performance in completing the password-based authentication process for one virtual user.

C. *Role-Based Authorization*

The experimentation on the role-based authorization procedure included two parts. The first was to measure the time taken to authorize a virtual user when the system under test contains one role. The second part was the same as the first but with two roles stored in the system such that the virtual user is assigned to one of them.

The latter testing aimed to see the effect of dividing the system's APIs between two roles on the performance of SQL and NoSQL databases. More precisely, to determine whether the division makes the NoSQL performance surpasses that of SQL. This is because the search, then, in NoSQL will be limited to an array that is embedded in the NoSQL document of a role, which saves, solely, the APIs that are linked to that role. This is in comparison to looking inside a SQL relationship table that has the associations between all the roles and APIs in the system.

Looking at the first part of the experimentation, it is noticed that the results of SQL and NoSQL, as appears in Table V, are approximately constant and equal. They are all around 1.25ms. That indicates stability in the performance of SQL as well as NoSQL. Furthermore, that shows that both technologies complete the role-based authorization process within the same time period.

TABLE V. THE RESULTS OF SQL AND NOSQL PERFORMANCE TESTING ON AUTHORIZING ONE VIRTUAL USER WHEN THE SYSTEM INVOLVED ONE ROLE

Test Scenario ID	**Test Case ID**	**Number of Stored APIs**	**Database Technology**	**Authorization Time (in milliseconds)**
TS2	TS2_TC1	1	SQL	1.57
	TS2_TC2	10		1.11
	TS2_TC3	100		1.19
	TS2_TC4	1000		1.31
	TS2_TC1	1	NoSQL	1.20
	TS2_TC2	10		1.28
	TS2_TC3	100		1.27
	TS2_TC4	1000		1.23

Fig. 8 confirms the remarks that were mentioned. The two lines that represent SQL and NoSQL performance are almost straight, which reflects constancy. In addition, they show overlapping which clarifies the equality between the two database types in the execution time required to authorize a user.

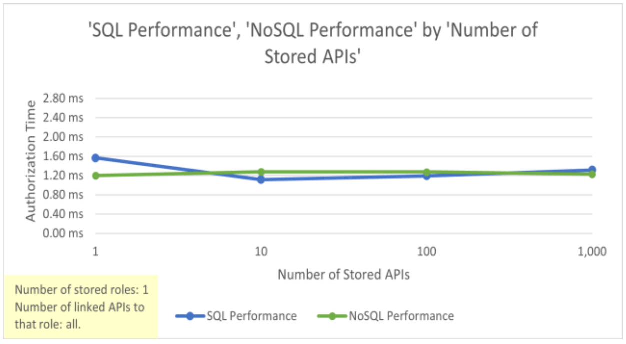

Figure 8. The SQL performance against the NoSQL performance in completing the role-based authorization process for one virtual user when the system held one role.

Regarding the second part of the role-based authorization testing, the numbers that Table VI presents are very similar to what Table V showed before. This similarity conveys two points. The first is that the existence of two roles in the system, which share the stored APIs, neither enhanced nor degraded the efficiency of the authorization process. And the second is sharing or dividing the APIs set between two roles did not give either database technologies superiority over the other.

TABLE VI. THE RESULTS OF SQL AND NOSQL PERFORMANCE TESTING ON AUTHORIZING ONE VIRTUAL USER WHEN THE SYSTEM INVOLVED TWO ROLES

Test Scenario ID	**Test Case ID**	**Number of Stored APIs**	**Database Technology**	**Authorization Time (in milliseconds)**
TS2	TS2_TC5	2	SQL	1.23
	TS2_TC6	10		1.26
	TS2_TC7	100		0.97
	TS2_TC8	1000		1.26
	TS2_TC9	2000		1.09
	TS2_TC10	3000		1.26
	TS2_TC5	2	NoSQL	1.34
	TS2_TC6	10		1.26
	TS2_TC7	100		1.18
	TS2_TC8	1000		1.22
	TS2_TC9	2000		1.24
	TS2_TC10	3000		1.26

As an additional note on the new results, raising the number of stored APIs in the system up to three thousand kept the same level of performance of SQL and NoSQL in the process of authorization. By performing experiments on a huge number of APIs, it was intended to provide evident outcomes that help architects in taking the right architectural design decisions while creating large-scale microservice systems.

Fig. 9 clarifies that the performance line of SQL is stable along all the experiments as well as the line of

NoSQL. Furthermore, it is observed that some of the two lines' data points are laid on top of each other and the rest are very close to each other. These are signs of equivalence between the SQL and NoSQL database technologies in the time that they spend completing the role-based authorization process.

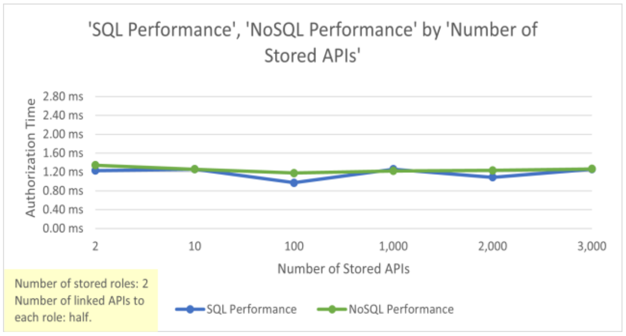

Figure 9. The SQL performance against the NoSQL performance in completing the role-based authorization process for one virtual user when the system held two roles.

D. *Final Analytical Thoughts*

Results can be analyzed from another perspective such as the scale of a microservice application. In this research, three scales were defined based on the value of the independent variable in each test scenario. These include small-, medium-, and large-scale applications as presented in Table VII.

TABLE VII. MICROSERVICE APPLICATION SCALES BASED ON USER AND API NUMBERS

Test Scenario ID	Independent Variable	Microservice Application Scale	Value of Independent Variable per Scale
TS1	Number of users	Small	Less than 1500 users
		Medium	Between 1500 and 5000 users
		Large	More than 5000 users
TS2	Number of APIs	Small	Less than 500 APIs
		Medium	Between 500 and 1000 APIs
		Large	More than 1000 APIs

Looking at the results of the authentication process, it is optimal to use NoSQL databases with small-scale software while it is preferable to use SQL databases for medium- to large-scale software. On the other hand, the role-based authorization process results indicate that using either SQL or NoSQL databases is acceptable for all scale levels.

IX. DISCUSSION OF RESULTS

This section provides a discussion and interpretation of the obtained results. Particularly, it explains the reasoning behind the superiority of each database type over the other in each branch of experimentation. The explanation will be based on hypotheses, previous studies, and practitioners' opinions.

A. *Password-Based Authentication*

The password-based authentication process comprises two programmatic main steps. The first is the search to find a match for a username and password, submitted by a user, inside the authentication database. The second is the generation of a token that contains the user's claims which are retrieved from the authentication database as well.

To complete such a process efficiently, the right database selection decision must be made. Regarding this context, the expectation is that using a NoSQL database gives better performance. As proven in research works, the NoSQL technology performs well in the execution of simple CRUD operations [24, 25]. In the password-based authentication process, the token generation step includes a straightforward read operation and what happened in the search step is reading as well but without a retrieval action.

Despite this assumption, it is speculated that SQL technology will outperform NoSQL at some point with the increase in the number of user records. The reason for that is that both the search and the claims fetching tasks depend on indexed columns. Indexes in SQL work on enhancing the search and querying speed to achieve optimization [26].

The results of authentication affirmed these two hypotheses. In the beginning, they revealed the superiority of the NoSQL database in performance, then, at a certain limit, they showed reversing in favor of the SQL database.

B. *Role-Based Authorization*

To authorize a user based on roles that are associated with APIs, simple database read operations are required. That leads to the anticipation of the suitability of the NoSQL technology for this context in terms of performance because, as was mentioned before, NoSQL was proven to carry out simple CRUD operations efficiently.

After performing experiments, surprising results were obtained. The SQL and NoSQL databases spent equal time completing the authorization process. The process was too short and lasted solely one millisecond to finish. The shortness of the process is the reason behind the equivalence since there was no opportunity for NoSQL to exceed.

Even when the authorization performance testing was repeated to include two roles each of which is holding the half set of the system's APIs, it was expected that NoSQL will exhibit more superiority. The reason behind that is each role in NoSQL is represented by a document that stores an array of the linked APIs' IDs. This contrasts with SQL which preserves the relationships between all roles and APIs inside one table. Nevertheless, as discussed, the procedure's duration was not long enough to show any difference in performance.

That shows the significance of the conducted performance tests since developers may think in the same way as the authors. And consequently, insist on employing a NoSQL database in authorization losing the feature of preserving data consistency that SQL provides. Security contexts are very sensitive so if there is a possibility of using reliable data sources like SQL databases, then it should be done.

X. Conclusion

Authentication and authorization are principal security requirements in any microservice-based system. In the context of password-based authentication and role-based authorization, one integrated microservice may be sufficient to achieve their requirements. Nevertheless, depending on two separate microservices, each of which is connected to a distinct database, becomes a necessity in case of significant growth in API numbers. This is to facilitate storing role/API associations and enable modifying them dynamically. This is instead of hard-coding these associations via API-level authorization attributes.

For database performance optimization of the separate password-based authentication and role-based authorization microservices, several performance tests were executed as a comparison between SQL and NoSQL databases. Depending on the analyzed and interpreted performance testing result, some conclusions were drawn. These conclusions are split into two parts in the same way that the results and their discussion were divided earlier.

A. Password-Based Authentication

To authenticate users based on passwords efficiently, it was discovered that the NoSQL database technology is better to use with small microservice-based systems which involve about 1500 users or less. On the other hand, SQL was found to be more suitable with medium- to large-scale microservice-based systems that deal with more than 1500 users.

B. Role-Based Authorization

In the process of role-based authorization, there is no difference between employing a SQL or NoSQL database. They both provide the same level of performance in the different API-based scale levels including small (less than 500 APIs), medium (from 500 to 1000 APIs), and large (more than 1000 APIs). That is true whether a microservice-based system contains only one role that is linked to all the system's APIs or to many roles that share these APIs.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Authors had integral roles in producing this research work as best as possible. Randa Ahmad Al-Wadi conducted the research, performed the testing and analysis phases, and drafted the manuscript. Adi A. Maaita suggested the research idea, supervised the research work, and made regular revisions to the manuscript. All authors had approved the final version of the manuscript.

References

[1] N. Dragoni, *et al.*, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216.

[2] Z. Xiao, I. Wijegunaratne, and X. Qiang, "Reflections on SOA and Microservices," in *Proc. 2016 4th International Conference on Enterprise Systems (ES)*, 2016, pp. 60–67.

[3] M. Fowler and J. Lewis. Microservices. [Online]. Available: https://martinfowler.com/articles/microservices.html

[4] C. K. Rudrabhatla, "Security design patterns in distributed microservice architecture," arXiv preprint, arXiv:2008.03395, 2020.

[5] A. Nehme, V. Jesus, K. Mahbub, and A. Abdallah, "Securing microservices," *IT Professional*, vol. 21, no. 1, pp. 42–49, 2019.

[6] R. M. Munaf, J. Ahmed, F. Khakwani, and T. Rana, "Microservices architecture: Challenges and proposed conceptual design," in *Proc. 2019 International Conference on Communication Technologies (ComTech)*, 2019, pp. 82–87.

[7] T. Yarygina and A. H. Bagge, "Overcoming security challenges in microservice architectures," in *Proc. 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2018, pp. 11–20.

[8] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira, "Security in microservices architectures," *Procedia Computer Science*, vol. 181, pp. 1225–1236, 2021.

[9] S. Aruna, "Security in web services-issues and challenges," *International Journal of Engineering Research & Technology*, vol. 5, no. 9, 2016.

[10] C. D. L. Torre, B. Wagner, and M. Rousos. NET Microservices: Architecture for Containerized .NET Applications. [Online]. Available: https://dotnet.microsoft.com/learn/aspnet/microservices-architecture

[11] Oracle Developers Channel. (2018). Authentication as a Microservice. YouTube. [Online]. Available: https://www.youtube.com/watch?v=SLc3cTlypwM

[12] X. He and X. Yang, "Authentication and authorization of end user in microservice architecture," *Journal of Physics: Conference Series*, vol. 910, 012060, 2017.

[13] U. Sharma, *Practical Microservices*, Packt Publishing, 2017.

[14] Role-based authorization in ASP.NET Core. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-7.0

[15] M. McLarty, R. Wilson, and S. Morrison, *Securing Microservice APIs*, 1st ed., O'Reilly, 2018.

[16] H. Vural, M. Koyuncu, and S. Guney, "A systematic literature review on microservices," in *Proc. 2017 International Conference on Computational Science and Its Applications*, pp. 203–217, 2017.

[17] P. Nkomo and M. Coetzee, "Software development activities for secure microservices," in *Proc. 2019 International Conference on Computational Science and Its Applications*, 2019, pp. 573–585.

[18] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE Software*, vol. 35, no. 3, pp. 56–62, 2018.

[19] R. Tighilt, M. Abdellatif, N. Moha, and Y.-G. Guéhéneuc, "Towards a tool-based approach for microservice antipatterns identification," in *Proc. the 12th International Conference on Advanced Service Computing*, 2020, pp. 15–20.

[20] R. Anderson. Introduction to Identity on ASP.NET Core. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio

[21] F. Fahrurazi, S. Ibrahim, and D. Suffian, "The design and execution of performance testing strategy for cloud-based system," *International Journal of Software Engineering and Technology*, vol. 1, 2014.

[22] O. Prusak. How many virtual users do I need for load testing? TechBeacon. [Online]. Available: https://techbeacon.com/app-dev-testing/how-many-virtual-users-do-i-need-load-testing#aoh=16424249205485&referrer=https%3A%2F%2Fwww.google.com&amp_tf=%D9%85%D9%86%20%251%24s&ampshare=https%3A%2F%2Ftechbeacon.com%2Fapp-dev-testing%2Fhow-many-virtual-users-do-i-need-load-testing

[23] F. Middleton. Reliability vs validity: What’s the difference? [Online]. Available: Scribbr. https://www.scribbr.com/methodology/reliability-vs-validity/

[24] M.-L. E. Chang and H. N. Chua, “SQL and NoSQL database comparison,” in *Proc. 2019 International Conference on Advances in Information and Communication Networ*ks, 2019, pp. 294–310.

[25] Z. Parker, S. Poe, and S. V. Vrbsky, “Comparing NoSQL MongoDB to an SQL DB,” in *Proc. the 51st ACM Southeast Conference*, 2013, https://doi.org/10.1145/2498328.2500047

[26] SolarWinds. How to Increase Database Performance—6 Easy Tips. [Online]. Available: https://www.dnsstuff.com/how-to-increase-database-performance

Copyright © 2023 by the authors. This is an open access article distributed under the Creative Commons Attribution License (CC BY-NC-ND 4.0), which permits use, distribution and reproduction in any medium, provided that the article is properly cited, the use is non-commercial and no modifications or adaptations are made.