

**Taras Batiuk**

PhD, Assistant of Information Systems and Networks Department
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0000-0001-5797-594X
taras.m.batiuk@lpnu.ua

Yurii Laruk

PhD, Assistant of Information Systems and Networks Department
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0000-0001-5514-6312
yurii.v.laruk@lpnu.ua

**IMPLEMENTATION OF OPTIMIZED AUTHENTICATION AND AUTHORIZATION
ALGORITHMS IN THE INTELLIGENT SYSTEM**

Abstract. The implementation of bcrypt and HMACSHA512 algorithms, as well as the use of JWT Access and Refresh tokens to ensure reliable authentication and authorization in the intelligent system, are proposed. The role of these algorithms in increasing the security of passwords, protecting against data forgery and providing effective access control is considered. It describes in detail the implementation process of each algorithm and their use in various scenarios. The importance of using powerful algorithms to ensure security and privacy in intelligent systems is highlighted, especially considering the growing cyber security threats. Features of creation, processing and storage of JWT Access and Refresh tokens on the server with storage of coded and hashed information in the database and in the client part of the program with storage of data in local storage and using http-only cookies are described. All the given algorithms are described with the help of formulas and conceptual diagrams, after which the software implementation of authentication and authorization in an intelligent system is given.

Keywords: bcrypt; HMACSHA512; authentication; authorization; JWT Access and Refresh tokens.

INTRODUCTION

The creation and implementation of optimized authentication and authorization algorithms in an intelligent system is a relevant and promising task, in the modern digital world, the security and reliability of authentication and authorization are critically important aspects for the development of intelligent systems. The growth of cybersecurity threats, which is clearly visible in the increase in the number of various cyberattacks on Ukrainian state and private online services after Russia's attack on Ukraine, requires the implementation of powerful algorithms and effective methods for password protection, data falsification prevention, and access control.

This work is devoted to the implementation of a web application using ASP.NET Core Web Api and Angular for user authentication and authorization in the most secure way. To achieve this goal, we consider important concepts such as frontend and backend validation, implementation of bcrypt and HMACSHA512 algorithms, as well as the use of JWT Access and Refresh tokens to securely store the data of an authorized user during his work on the created web resource.

Describing the implementation process of each algorithm and their use in various scenarios, we reveal the importance of these powerful algorithms for ensuring security and confidentiality in intelligent systems. Special attention is paid to the creation, processing and storage of JWT Access and Refresh tokens on the server using a database and the client part of the application using localStorage and http-only cookies.

The object of the study is the authentication and authorization of users in an intelligent system using various algorithms such as bcrypt and HMACSHA512, as well as the use of JWT Access and Refresh tokens to ensure reliable authentication and authorization. The main focus of the study is on describing the features of creating, processing and storing JWT Access and Refresh tokens. In the course of the work, the process of generating, encoding and storing these tokens will be studied in detail, including the use of cryptographic algorithms to ensure the security and confidentiality of information. The object of the study involves the analysis and assessment of the role of the considered algorithms in ensuring security during the process of user authentication and authorization. When creating an intelligent system, the features of using these algorithms to increase password security, protect



against data forgery and effectively manage access will be studied. The study examines the features of creating, processing and storing JWT Access and Refresh tokens in order to implement safe and reliable authentication and authorization of users. Attention is paid to ensuring the integrity, confidentiality and authenticity of tokens during their processing and transfer between the client and the server. The object of the study involves the development of a methodology and software implementation of authentication and authorization in an intelligent system, including a description of the algorithms used and the process of their implementation. The result of the research will be a deep understanding of the process of ensuring security and confidentiality during authentication and authorization in intelligent systems and the ability to apply this knowledge to the development of secure applications. Another important task is to show the distribution of various authentication and authorization metrics. The subject of the research is the creation of a web application for user authentication and authorization in the most secure way in an intelligent system. The research aims to study various concepts, such as frontend and backend validation, implementation of bcrypt and HMACSHA512 algorithms, as well as the use of JWT Access and Refresh tokens to ensure reliable authentication and authorization in an intelligent system. The main aspects studied include the role of these algorithms in increasing password security, protecting against data forgery, and ensuring effective access control.

It is necessary to describe in detail the implementation process of each algorithm and their use in various scenarios. It is also worth noting the importance of using powerful algorithms to ensure security and privacy in intelligent systems, especially in the context of growing cybersecurity threats. In the course of the work, it is important to describe the features of creating, processing and storing JWT Access and Refresh tokens on the server with the preservation of encoded and hashed information in the database and in the client part of the program using local storage and http-only cookies to ensure security and effective authentication and authorization.

The purpose of the work is to implement optimized authentication and authorization algorithms in an intelligent system. An important task is to create a part of the intelligent system responsible for authentication, starting from a general analysis of the literature and existing algorithms and programs of analogues. Next, it is necessary to use diagrams and flowcharts to carry out conceptual design and system analysis of both the main algorithms that will be used for authorization and the client-server system itself as a whole, in order to have a clear understanding of both the operation of the algorithms and the transmission and storage of data within the intelligent system. Having a conceptual understanding of the designed system, it is necessary to implement the main authentication and authorization algorithms, verify their correct operation and implement the intelligent system. The last important step is the need to test the results of the created system, first as a separate application in the form of requests to the created API and in general to check the correct operation of the client-server system and its compliance with the specified requirements and safety for the work of end users of the intelligent system and the reliability of protection of authorization and other data provided by users.

The paper [1] is devoted to the study of the key storage mechanism for functions related to asymmetric keys in the Cryptography Next Generation cryptographic library. The authors of the paper consider important aspects related to the security of storing and managing asymmetric keys in this library. The paper analyzes the key storage mechanism and identifies possible security problems that may arise. The authors carefully consider the applied encryption and key storage methods and discuss possible risks, such as attacks on cracking the key or leaking important data. The paper also considers recommendations for improving the key storage mechanism and increasing security. The authors suggest the use of additional security measures, such as encrypting keys using passwords or using hardware security modules.

The paper [2] is also devoted to the development of energy-efficient distributed calculation of password hashes on heterogeneous embedded systems. The authors investigate the problems associated with energy consumption when calculating password hashes and propose a distributed architecture to reduce energy consumption. The paper describes the developed algorithm that distributes the calculation of password hashes among different embedded devices, such as microcontrollers, processors, and FPGAs. Taking advantage of the heterogeneity of these systems, the algorithm distributes the computational load in a way that ensures energy-efficient password processing. The authors conduct an experimental study to compare the performance and power consumption of the developed distributed algorithm with existing methods. The results show that the proposed approach provides a significant reduction in power consumption while maintaining acceptable computational performance.

In paper [3], the authors considered a method of attacking HMAC-SHA-2 using differential power analysis, known as Carry-based Differential Power Analysis. The authors consider using this method to crack HMAC-SHA-2, one of the most widely used integrity-checking hashing algorithms. The paper explains the basic principles of CDPA and describes in detail its application to the attack on HMAC-SHA-2.

CDPA uses a hardware-based storage analysis to detect common differences in energy consumption during hashing operations. Using these differences, an attacker can obtain information about the HMAC-SHA-2 secret key. This paper presents experimental results demonstrating the success of the CDPA attack on HMAC-SHA-2.

The authors discuss the possible security implications of this attack for systems that use HMAC-SHA-2 for authentication and data integrity.

Paper [4] presents an overview and development of a JSON Web Token (JWT) implementation for authentication using the HMAC SHA-256 algorithm. The authors investigate the use of JWT for secure transmission of authentication information between a client and a server. The paper explains in detail the concept of JWT and its structure, including the header, body, and signature. The authors describe the advantages of using JWT for authentication, such as ease of integration, ease of extension, and security. The main part of the paper is devoted to the development of an implementation of the HMAC SHA-256 algorithm for generating and verifying a JWT signature. The authors explain the steps required to generate and verify a signature, including computing the HMAC SHA-256 and using a secret key.

In paper [5], the authors propose a new protocol for centralized key generation using the Shamir Secret Sharing protocol and HMAC authentication. The authors investigate the use of this protocol for generating and managing keys with a high level of security and trust. The paper explains the basic concepts and principles of the Shamir Secret Sharing protocol, which allows breaking a secret key into parts and distributing it among trusted parties. The authors also use HMAC authentication to ensure the identification and integrity verification of these parties. The paper describes a key generation protocol that combines the Shamir Secret Sharing protocol and HMAC authentication. The authors consider the steps required to generate and combine key parts, as well as the use of HMAC for authentication and integrity verification. The paper also offers a security analysis of the proposed protocol.

THEORETICAL FOUNDATIONS

To begin with, it is worth giving a general overview of the basic concept of secure storage of user passwords required for authentication and authorization, namely hashing. Hashing converts user passwords into unreadable hash values. This makes it impossible to directly read or reveal passwords in the event of improper storage or data loss. If the hash values are stolen, it is difficult or impossible to convert them back to the original password. Hashing helps prevent unauthorized access to user accounts. During login authentication, an intelligent system compares the hash value of the stored password with the hash created from the password entered by the user. If the hashes match, the system grants access. Hashing allows you to store user passwords [6] in encrypted form, which ensures the confidentiality of this data. Even the system administrator does not have direct access to the original user passwords. In general, password hashing is a necessary procedure for ensuring the security and confidentiality of accounts. Figure 1 shows a conceptual algorithm for hashing a user password.

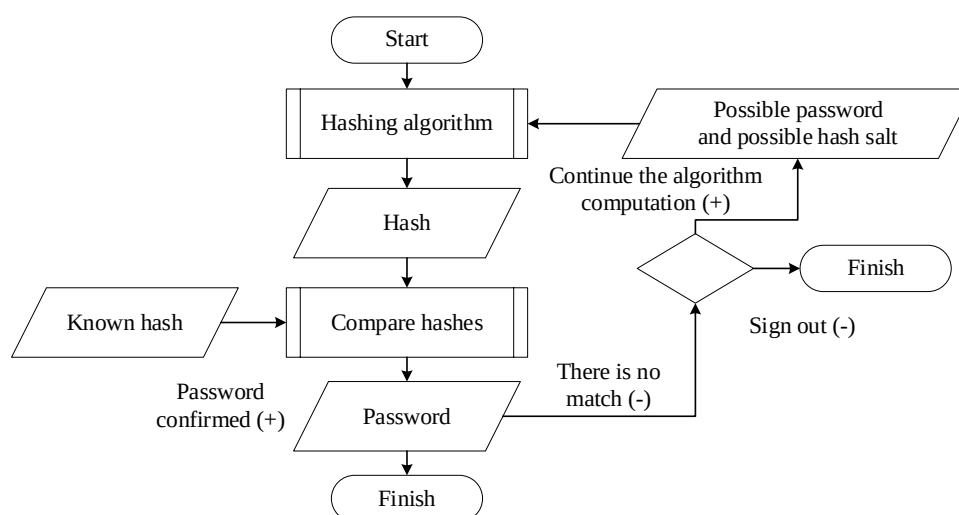


Fig. 1. Conceptual algorithm for hashing a user password

In the course of work, the SHA-512 hash algorithm will be used, which will be used together with the HMAC algorithm and plays an important role in ensuring the security of the user password. The SHA-512 algorithm establishes additional constants that determine the behavior of the algorithm. One of such constants is the output size - 512, which means the size of the output data in bits. SHA-512 has the properties necessary for ensuring data security, such as one-way, collision resistance, lack of vulnerabilities and reproducibility, and the hash width of 512 bits [7] allows you to obtain a very large number of possible hash values, which minimizes the



chance of getting a collision. This algorithm is worth considering step by step from a mathematical point of view. First, we perform preprocessing to ensure that the input data will be exactly divisible by 1024 bits. Next, you need to initialize the hash value (h) by creating 8 constants that represent the first 64 bits of the fractional parts of the square root of the first 8 prime numbers, as shown in Formula 1.

$$Ch(X, Y, Z) = (X \wedge Y) \oplus (\bar{X} \wedge Z), \quad (1)$$

After initializing the hash value, we initialize the rounded constants (k) by creating 128 constants, each of which represents the first 64 bits of the fractional parts of the cube root of the first 128 prime numbers, as can be seen in Formula 2.

$$Maj(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z), \quad (2)$$

The next step is to run the main loop, in which messages are divided into fixed-size blocks, which are processed by sequentially changing the values of the hash functions h_0-h_7 created in the previous step [8] at each iteration of the loop to obtain the final result, which can be seen in Formula 3.

$$\Sigma_0(X, Y, Z) = RotR(X, 2) \oplus RotR(X, 13) \oplus RotR(X, 22), \quad (3)$$

Further, during the operation of the algorithm, we create a message queue (w), copying the input data and adding additional words initialized with zeros, this can be seen in Formula 4.

$$\Sigma_1(X, Y, Z) = RotR(X, 6) \oplus RotR(X, 11) \oplus RotR(X, 25), \quad (4)$$

We initialize the variables $a-h$ and run a compression loop in which we change the values of the hash function variables 63 times using various operations such as bitwise AND, OR, NOT, rotations, and logical functions, as can be seen in Formula 5.

$$\sigma_0(X, Y, Z) = RotR(X, 7) \oplus RotR(X, 18) \oplus ShR(X, 3), \quad (5)$$

After performing compression, it is necessary to modify the hash values after the compression cycle by adding the corresponding variables $a-h$ to them, this can be seen in formula 6.

$$\sigma_1(X, Y, Z) = RotR(X, 17) \oplus RotR(X, 19) \oplus ShR(X, 10), \quad (6)$$

We obtain a 512-bit final hash by combining all the hash function values together. Thus, all the steps described in the formulas are performed to hash the input data and obtain the final hash [9], which represents the unique sum of this data. Figure 2 shows a diagram of the SHA-512 hash algorithm.

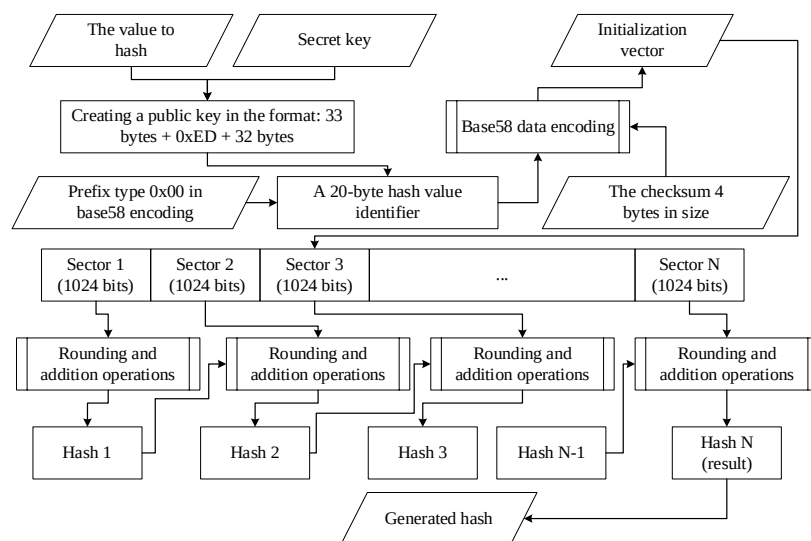


Fig. 2. Diagram of the SHA-512 hash algorithm



Having an understanding of how the SHA-512 algorithm works, it is possible to implement a more modern hashing algorithm HMACSHA-512, since it has a large number of advantages compared to just SHA-512 and it is recommended to use it when creating intelligent systems. The HMACSHA512 algorithm (Hash-based Message Authentication Code using SHA-512) combines two cryptographic algorithms - HMAC and SHA-512, to ensure the integrity and authenticity of the input message. Like all algorithms of this type, HMACSHA512 has clearly defined implementation steps and, accordingly, data distribution during the execution of the algorithm.

The most important step is to determine the secret key that will be used to calculate the HMAC. The key must be long enough and random to ensure security, and it must also be securely stored on the server in such a way that it cannot be accessed from the client part of the intelligent system and from other external sources. The next step is to prepare the data that must be hashed using the algorithm. The data for which HMACSHA512 is to be computed [10] is preprocessed. This includes adding certain bytes, padding, and general formatting of all the original data according to the specified requirements of the HMACSHA512 key.

The general construction of HMAC looks like this: to obtain the value $Gen(1^n)$, it is necessary to input 1^n and start key generation for the hash function, which will receive s , and choose $k \leftarrow \{0,1\}^n$, after which to obtain $Mac_k(m)$, it is necessary to input (s, k) and $x \in \{0,1\}^*$, calculate Formula 7 and output the result.

$$HMAC_k^s(x) = H_{IV}^s(k \oplus opad \parallel H_{IV}(k \oplus ipad \parallel x)) \quad (7)$$

Having prepared the data, we can proceed to calculating the HMAC. To calculate HMACSHA512, the internal hash function SHA-512 is used. HMACSHA512 uses the internal state of SHA-512 as the main component for calculating the message hash. Therefore, it is necessary to calculate the internal state necessary for further work. The value of the internal state of HMACSHA512 is calculated by calculating the hash of the input key using the internal starting value, known as IPAD. The resulting internal state IPAD is mixed with the key using the XOR operation. The mixed value is passed through iterations of the SHA-512 hash function. The result is a message hash, which includes the internal state of HMAC. Also important is the value of $Vrfy_k(m, t)$, which gives 1 at the output only if and only if $t = Mac_k(m)$ of the pseudo-random function. Now the first step in computing the inner hash $H_{IV}^s(k \oplus ipad \parallel x)$ is $z_1 = h^s(IV \parallel k \oplus ipad)$. Similarly, the first step in computing the outer hash is $z'_1 = h^s(IV \parallel k \oplus opad)$. Considering h as a pseudorandom function, we can conclude that $h^s(IV \parallel k \oplus ipad)$ and $h^s(IV \parallel k \oplus opad)$ are essentially different pseudorandom k_1 and k_2 . Thus, denoting $k_1 = h^s(IV \parallel k \oplus ipad)$ and $k_2 = h^s(IV \parallel k \oplus opad)$, as given in Formula 8.

$$HMAC_k^s(x) = H_{IV}^s(k \oplus opad \parallel H(k \oplus ipad \parallel x)) = H_{k_1}^s(H_{k_2}^s(x)) \quad (8)$$

Having determined the internal state, it is necessary to calculate the external state. The calculated message hash becomes the input state for calculating the external state HMACSHA512, known as OPAD. The external state OPAD is mixed with the key using the XOR operation in the same way as the internal state, and at the output we get the calculated mixed state, thanks to which we can proceed to the final step. It is worth noting that the chosen value of the external application H concerns the output of $H_{k_2}^s(x)$ and therefore is only the resulting single calculation of h^s , this is given in Formula 9.

$$HMAC_k(x) = h_{k_1}^s(H_{k_2}^s(x)) \quad (9)$$

So, given Formula 9, we assume that deriving keys k_1 and k_2 using $ipad$ and $opad$ generates two pseudorandom keys, all else remaining the same. We formally obtain this value by defining the function G knowing that it will act as a pseudorandom generator with a spread factor. So, generator G doubles its length and thus produces two pseudorandom keys of length n from one chosen random key of specified length n , as given in Formula 10.

$$G(k) = h^s(IV \parallel k \oplus opad) \parallel h^s(IV \parallel k \oplus ipad) \quad (10)$$

Finally, the mixed state is passed through iterations of the SHA-512 hash function, resulting in the final HMACSHA512. The HMAC-SHA-512 algorithm ensures the integrity and authenticity of the message, since the hash and key are calculated from the internal and external states, mixed using XOR. This makes HMACSHA512 important for password protection, password signing, data authentication, and forgery prevention. So, Figure 3 shows the main steps of the HMACSHA512 hashing algorithm.

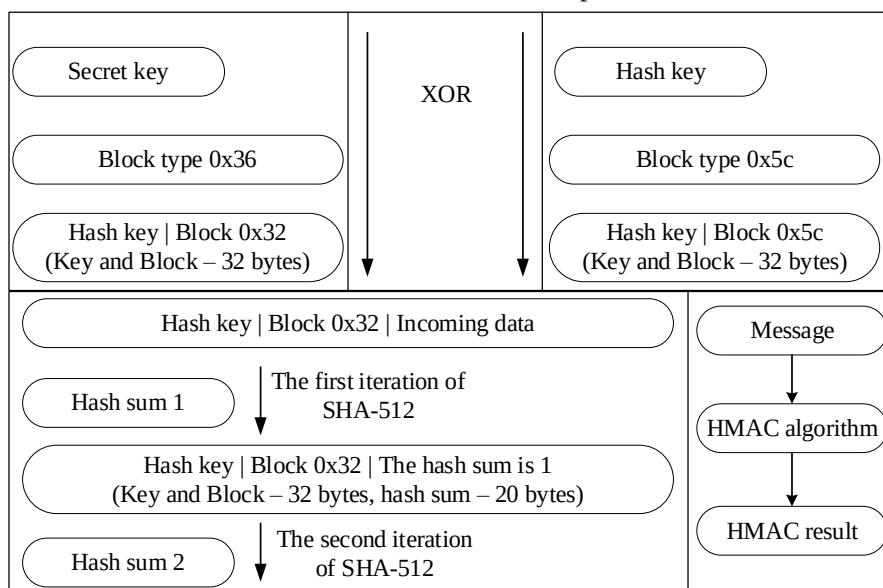


Fig. 3. Main steps of the HMACSHA512 hashing algorithm

The latest at the moment is the implementation of the password hashing algorithm in the intelligent system bcrypt, it is complex to implement and conceptually different from the HMACSHA512 algorithm, but plays a similarly important role in maximizing the security of user data and minimizing the chances of obtaining user passwords. The bcrypt hashing algorithm is one of the most reliable and stable algorithms for storing password hashes. The main feature, which is both an advantage and a disadvantage of bcrypt, is its slowness, which makes it difficult to attack using brute force. The first step is the generation of a salt, which must be created randomly. Salt generation is an important step in the bcrypt hashing algorithm.

The salt [11] is a random value that is added to the input password before hashing. It is used to ensure the uniqueness of the hashes, even if two users have the same passwords. The salt is generated randomly. This means that the salt value is unpredictable and not repeated for every password. Using a random salt makes rainbow table attacks (tables used to attack passwords that have been transformed using hash functions, allowing attackers to efficiently crack such passwords or perform attacks on ciphers using a limited amount of memory) impossible. The length of the salt varies, but it should usually be sufficient to ensure uniqueness. It is usually recommended to use a salt of 8 to 16 bytes [12], in our case we will generate a string of 16 bytes. The salt is added to the input password before hashing. This means that the actual input string to be hashed includes both the password itself and the salt. The generated salt value is stored together with the password hash in the database. This allows for future verification of the password by comparing the hashes that are calculated from the entered password and the corresponding salt value. Having the salt value generated, the input password and salt are combined into a single string. The input string is hashed using an iterative hash function that operates in a blocking mode. The iterative hashing process is repeated several thousand times, making the process resistant to brute force attacks. The resulting hash is stored together with the salt in a database. So, when a user attempts to authenticate, the stored hash and salt values are recreated. A key is derived from the entered password, hash, and salt. The user's entered password is encrypted using the resulting key. If the generated ciphertext matches the stored ciphertext, the password is valid.

It is also worth noting that the hashed user password stored in the database consists of three fields separated by the "\$" symbol and, for example, looks like this: "\$2a\$11\$Rc6XAhBsgopPRpgUFroameVYa2qg2iKurrbXmp/5Aeo4qppWFGJDC", at the beginning there is always "2a" - it determines the version of the bcrypt algorithm that was used, "11" - represents the cost factor (can have a maximum value of 16, but this will significantly affect the performance of the algorithm due to exponential complexity), which indicates the number of iterations of the key derivation function, i.e. 211 iterations. The remaining characters after the "\$" sign represent the salt and ciphertext, which are combined and encoded in Base64. The first 22 characters are decoded into a 16-byte salt value, and the remaining characters are the ciphertext used for comparison during authentication. The flowchart of the bcrypt hashing algorithm is shown in Figure 4.

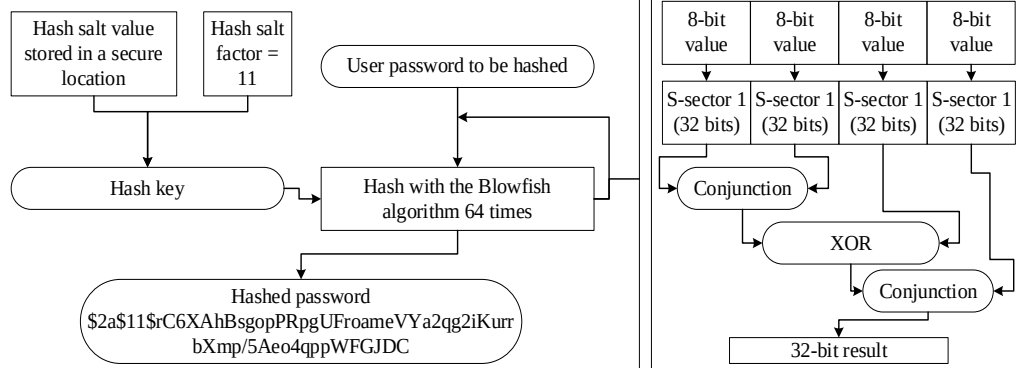


Fig. 4. Diagram of the bcrypt hashing algorithm

At this point, we have an understanding of the operation of the three main data hashing algorithms and have implemented the necessary ones for operation within the intelligent system HMACSHA512 and bcrypt. Now it is necessary to optimally integrate them within the system for maximum security of the user password. During the implementation of the user registration component, we work with the server part of the intelligent system, since it is here that the password hashing will be carried out and it is the server that will communicate with the database, which will store and subsequently retrieve the user data required in accordance with the request sent to the server. First of all, to create a general password hashing algorithm, you need to understand what data will be submitted for input, for now we will neglect the high-level implementation of checking the user password and its validation. Let us assume that at the input to the method responsible for hashing the password we have a string password, which contains the user's password, which is currently registered in the system and contains from 8 to 25 text characters, which need to be hashed and the final result saved in the database [13], which contains all the entered data about the user for his further identification. The main point is that hashing itself should provide the basis for data security, since even if a cyberattack is carried out on the database and the attackers have certain user data - the passwords will be in the form of hashes, which consist of a set of characters that have no meaning for the attackers, and which cannot be returned to the standard form and used as the user's original password in any way.

So we create a new HMACSHA512 algorithm object and initialize it using a salt stored in a secure place on the server, which is not accessible from the client or from other IP addresses that are not allowed. We encode the user password and salt into bytes, then calculate the hash using the HMACSHA512 algorithm, encode the resulting hash into a Base64 string, hash the resulting string using the bcrypt algorithm, and store the final hash as the resulting string in the database. Having implemented password hashing algorithms, you can continue to implement the user authentication logic in the system at a higher level, that is, implement user registration and authorization mechanisms. UML sequence diagrams are convenient to use to display these interaction mechanisms between the user, server and database. The created user registration sequence diagram is shown in Figure 5.

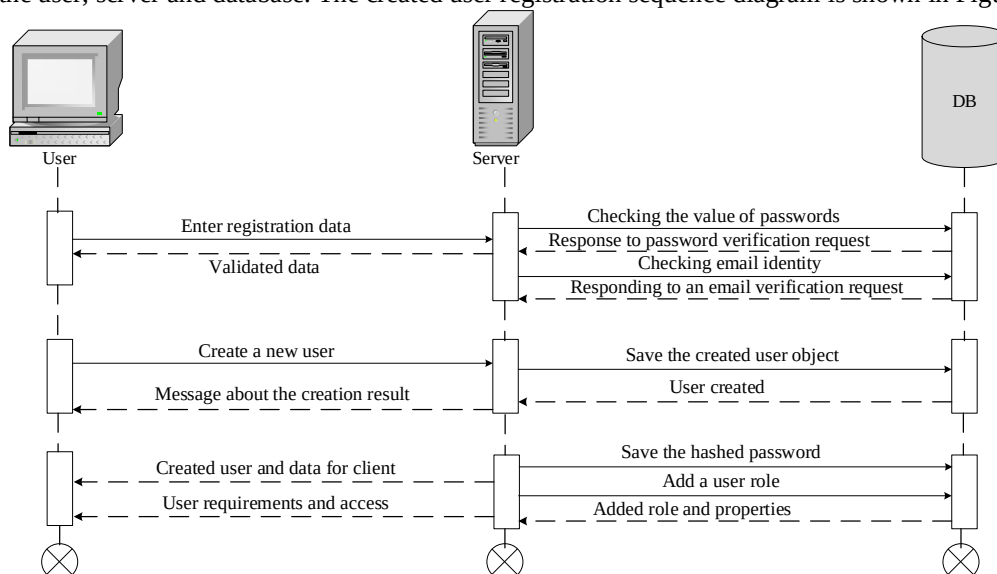


Fig. 5. Diagram of the user registration sequence

The next step is to implement the user login mechanism, this process is closely tied to the interaction between the client and the server and the transfer of JWT Access and Refresh tokens between them. The created user authorization sequence diagram is shown in Figure 6.

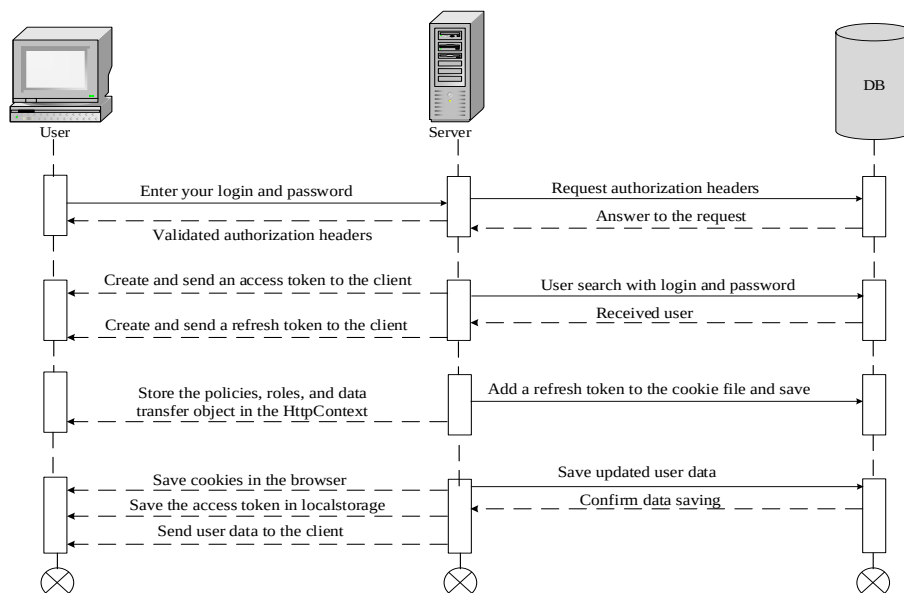


Fig. 6. Diagram of the user authorization sequence

The mechanism of user login and his work in the system is implemented using access and refresh tokens. In general, JSON Web Token is an open standard RFC 7519, which is used for secure transmission of information in JSON format between the client and the server in the form of a small token. JWT consists of three main parts: header, payload, and signature. The header contains the token type and the algorithm used for signing. The payload contains the data transmitted by the secure token, such as user ID, role, and token validity period. The signature is generated using a secret key, which allows you to verify the integrity and authenticity of the token. JWT tokens are compact [14], can be transmitted over the network easily and securely. Also, all the necessary information is contained in the token itself, which simplifies processing and verification. In addition, we implement a one-way authentication mechanism, verifying its authenticity without processing an external authentication source. However, it is important to consider some risks when using JWT tokens. For example, if the token is compromised or stolen, it can lead to the compromise of the intelligent system, so it is important to handle and store the secret key carefully and take security measures to protect the token, namely, in addition to the access token, use a refresh token. The block diagram of the entire authentication mechanism of the intelligent system is shown in Figure 7.

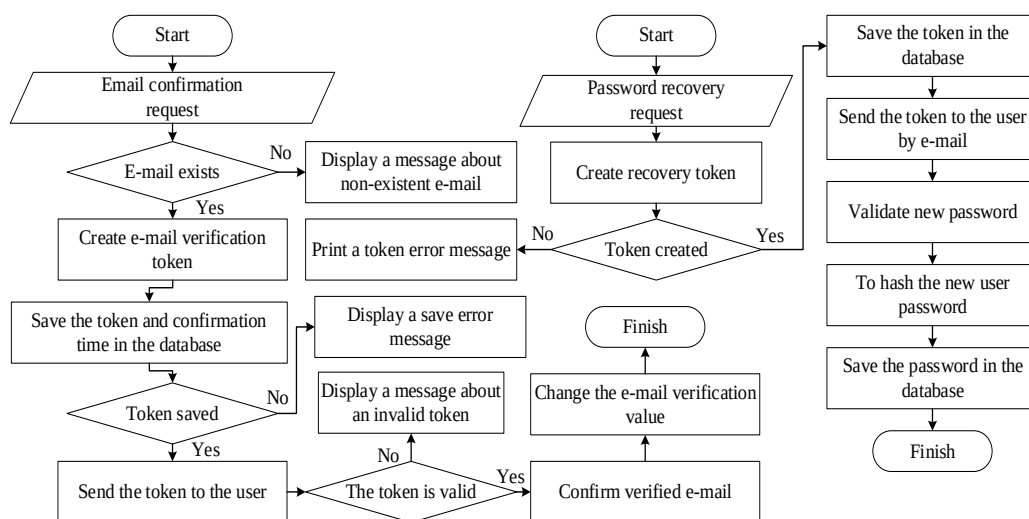


Fig. 7. Diagram of the entire authentication mechanism of the intelligent system



A JWT refresh token is a special type of token that is used to refresh JWT access tokens without the need to re-authenticate the user. Refresh tokens have a longer validity period than access tokens, which allows them to be used to obtain new access tokens after the old tokens expire. Refresh tokens are stored in a secure location, namely in an http-only cookie in the client browser. Http-only cookies are not accessible to JavaScript, so they are protected from XSS attacks (cross-site scripting attacks). In addition, these cookies are automatically transmitted with each request to the server, which allows you to conveniently and securely store the refresh token and ensure continuous user authentication without the need to manually transmit tokens with each request, which makes the access token renewal process transparent to the user. In addition, using http-only cookies reduces the risk of token theft through CSRF attacks (cross-site request forgery). We store refresh tokens in a secure location and take certain security measures to protect against unauthorized access, so we use cookie lifetime limitation mechanisms and refresh tokens once an hour, when the access token lives only 10 minutes.

So, a user login begins with entering a username and password, after which the server queries the database for authorization headers and sends them to the user's client. The server also searches the database to see if the user with the username and password exists, if so, creates an access and refresh token [15], adds the access token to the JSON response, and the refresh token to the http-only cookie. After that, the client stores policies, roles and the data transfer object in the HttpContext, and also sends an access token and a cookie with a refresh token to the client in response to the request, which are stored in localStorage and in the cookies storage, respectively. As can be seen from the diagram, in addition to registration and authorization, the intelligent system contains such important and necessary functions as email confirmation and password recovery. Thanks to the confirmed email, the user will be able to restore access to his account at any time in case of password loss or fraud, and the user can also change the password at any time.

RESEARCH RESULTS

The implementation of optimized authentication and authorization algorithms in an intelligent system is an important and difficult task, and first of all, it is necessary to correctly choose the technologies that are best suited to perform the necessary tasks, therefore, the C# programming language and its ASP .NET Core framework were chosen to create the backend part of the system, the Typescript language and the Angular framework will be used to create the frontend part of the program, MS SQL Server and Entity Framework Core will be used to store data, as an opportunity to conveniently and quickly work with data as OOP objects, and not with the SQL itself directly, which, although it minimally reduces the speed of the program, increases the productivity of creating new program functions and increases the security of the program, since you do not need to worry about such methods of attacks on the database as SQL injection. First of all, backend programs and all necessary API requests are written and tested and tested in various conditions. For this, it is convenient to use Swagger, a technology for documenting the entire REST-API within an intelligent system that provides a convenient user interface.

The system architecture is organized into multiple controllers, each of which handles a specific set of endpoints - such as those related to administration, authentication, user management, and email functionality. This modular separation ensures that each controller is responsible solely for its respective group of requests, making it easier to maintain and extend the system. It also streamlines communication with repositories and service layers, promoting a clear and logical structure. An important aspect of the architecture is the use of Data Transfer Objects (DTOs). These serve as intermediate data structures that allow the system to decouple internal database models from the data exposed externally. DTOs make it possible to tailor the returned data to the needs of individual requests without exposing or relying on the full database schema. This abstraction enhances both security and flexibility, ensuring that each endpoint returns only the relevant data.

When developing the server-side logic, it is essential to account for the state of the application at any given moment, as well as to ensure proper separation of concerns. Different components of the system are responsible for distinct functionalities, which also influences how testing procedures and error handling strategies are distributed. For instance, the core functionality related to token handling and email communication is encapsulated within dedicated services. The token service is tasked exclusively with generating access and refresh tokens and decoding them when needed. By limiting its responsibilities in this way, the service avoids unnecessary complexity and minimizes the risk of errors. The service accepts the necessary cryptographic keys for token generation as input and returns the resulting token strings as output. The processes for creating access and refresh tokens are fundamentally similar, with the primary distinction being the use of different secret keys for each type. The token creation begins with the definition of user claims - in this case, the user ID and their role.

Once the claims are defined, they are signed using a secret key to generate a token descriptor, which includes metadata such as the user's claims, the validity period of the token (10 minutes for access tokens), and a list of allowed audiences that can use the token. The refresh token, unlike the access token, is created directly within the controller logic. It is stored in an HTTP-only, secure cookie, which enhances security by preventing



client-side scripts from accessing it. Finally, testing of API endpoints, including registration and login flows, as well as validation of token generation and cookie handling, is performed using Postman. This tool allows developers to simulate various client requests and inspect the responses, tokens, and cookie headers. For example, Figure 10 illustrates how user registration and login requests are tested using Postman to verify the correctness of the authentication flow.

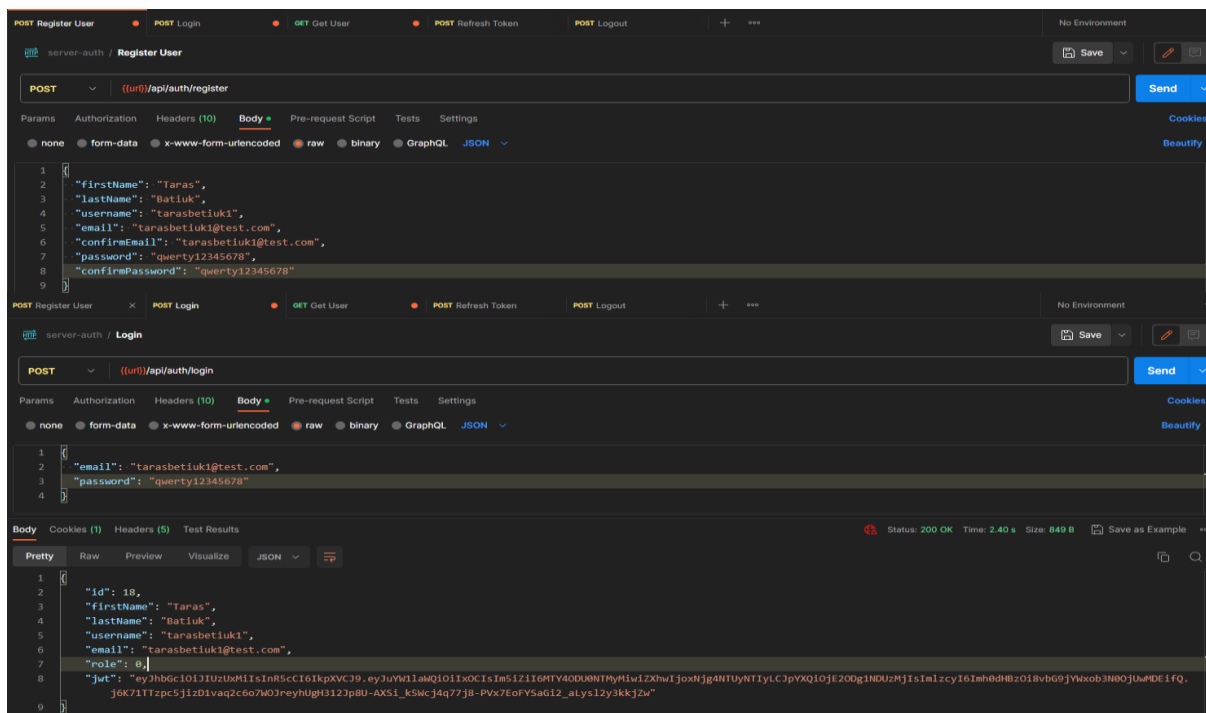


Fig. 8. Testing user registration and login requests in the Postman application

As you can see, the request consists of a long link to the server address and an endpoint, which consists of the controller name and the corresponding method. Also, the result shows that the server response status is 200 OK, which means that the response to the request was successfully returned in the form of data in JSON format, including the system user role and the created JWT access token, Figure 9 shows testing token update and user search requests.

In the token refresh request, we see that the previous access token was successfully deleted and a new access token was created using the refresh token stored in cookies, which will be used for all requests, such as a request for user information containing certain data, a hashed password, and email verification status. So, having implemented the backend part of the intelligent system, it is necessary to create a frontend part to have a full-fledged client-server system, the very first thing is user registration and authorization, so you need to create forms for entering user data and a basic post-authorization page.

Forms are implemented using Reactive Forms technology, which allows you to create a separate form object and process each field as its property. So, validators were created for each form field, a separate regular expression was created for the email field, which checks the correctness of data entry and field matching, and a separate password validator checks the identity of the entered passwords before sending a request to the server for authentication or registration using the service.

The main system pages were also created, such as the user profile, the user profile update page, where you can change all available user data and also his email and password, but this requires additional confirmation using an email. If the user has the administrator role, he accordingly has access to a separate admin page, where you can view information about all existing users, change roles, give or take away possible policies from them and, if necessary, delete the account, this is shown in Figure 10.

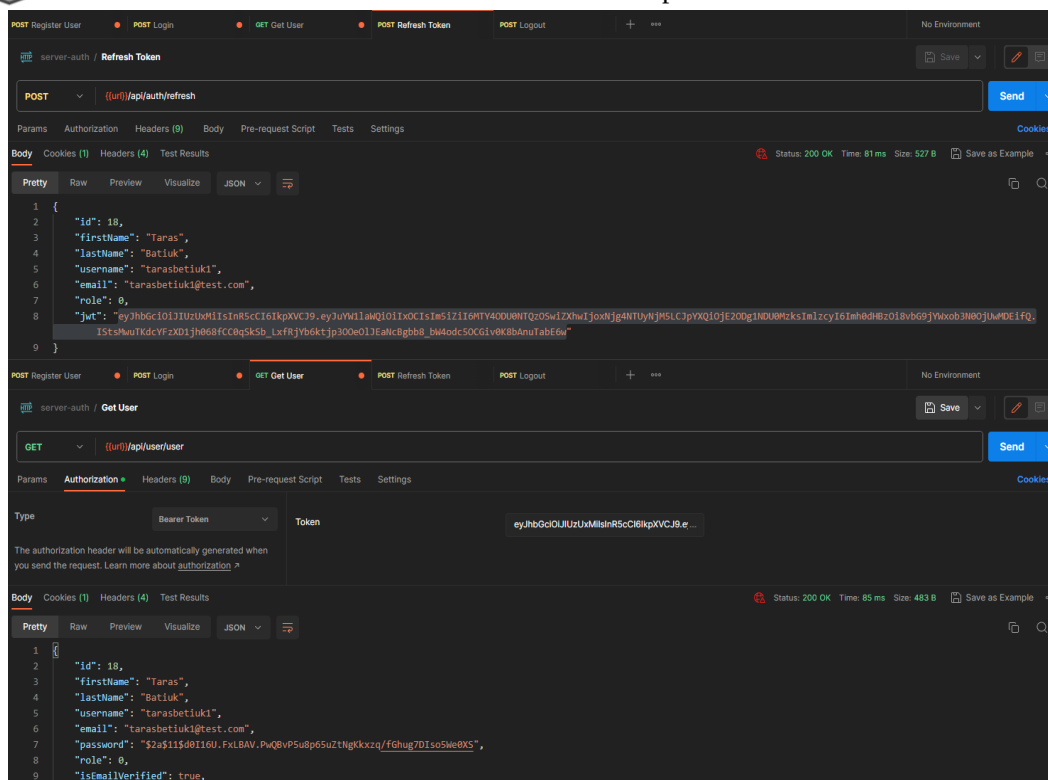


Fig. 9. Testing token and user information update requests in the Postman application

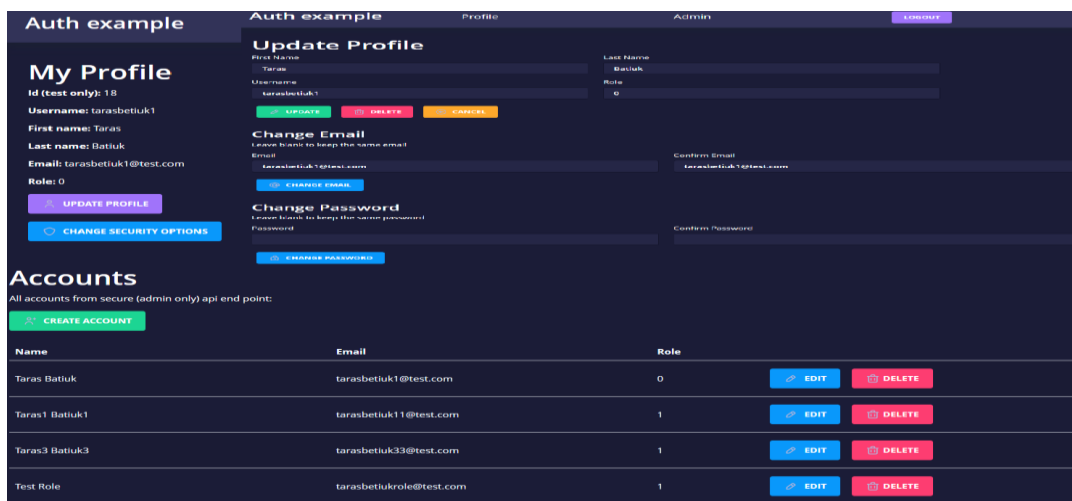


Fig. 10. Main pages of the intelligent system available to a registered user

Also, an important task was to ensure the correctness of the code using the mechanisms of the browser where the client is running, in our case it is Google Chrome, thanks to the means of checking the intelligent system and network requests, the correctness of storing the access token in localstorage and the cookie containing the refresh token in the storage and being http-only was checked, this is shown in Figure 11. In addition, all network requests were tested, thanks to which you can be sure of the security of requests, the correctness of headers and data payload, this is shown in Figure 12.

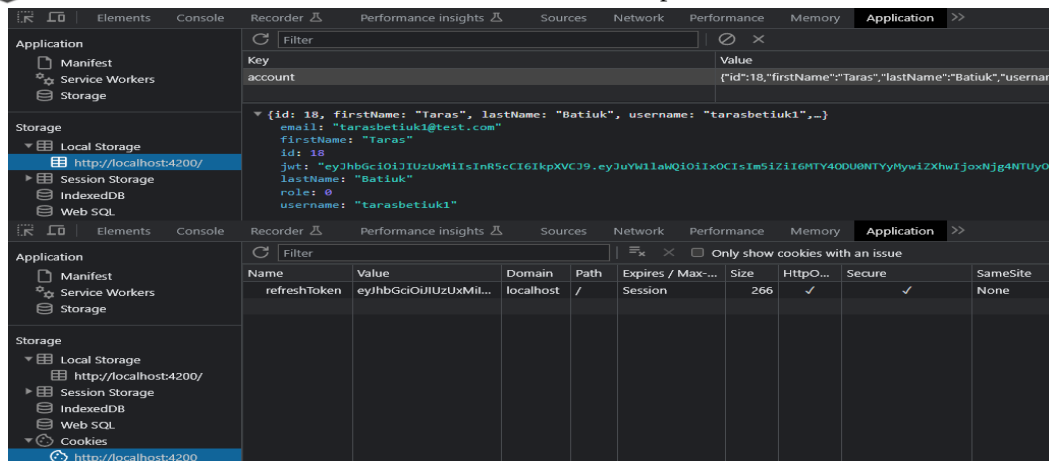


Fig. 11. Storing access and refresh tokens in localStorage and cookies, respectively

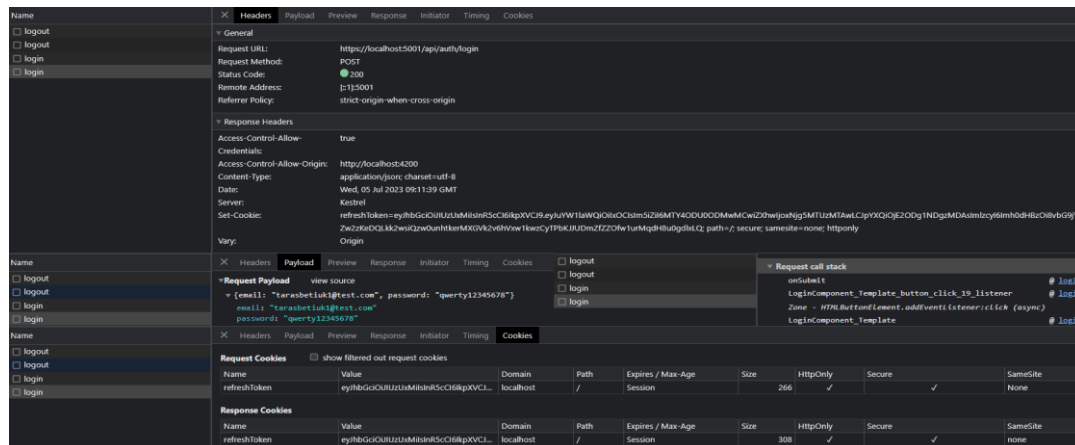


Fig. 12. Testing the correctness of all requests in the intelligent system network

Having a complex intelligent system with a large number of requests and updating data in cookies, such as a refresh token, it is important to monitor the large number of errors that may occur during data transfer between the client and the server, first of all, an SSL certificate was activated so that the connection to the web server was secure using the https protocol. At the end, such functionality as email verification was implemented, which is implemented by sending a generated code to the user's email using the SMTP protocol, as a verification code that the user must use to confirm in the form of an intelligent system. All necessary fields and a template for displaying the final message for user were configured. The Ethereal service was used to test user interaction and send emails, email verification and a confirmed a current user record in the database are shown in Figure 13.



Fig. 13. Email verification and recording of a verified user in the database



The last step is to implement the password recovery mechanism. First, the user specifies the email address where the recovery token should be sent. The created token is valid for only 30 minutes, is sent to the user, and is stored in the database table for password recovery tokens. The user follows the password recovery link, inserts the created token, and enters a new password. If the token is still valid and belongs to the current user, the token is deleted from the database, and the hashed new password is stored instead of the lost password, as shown in Figure 14.

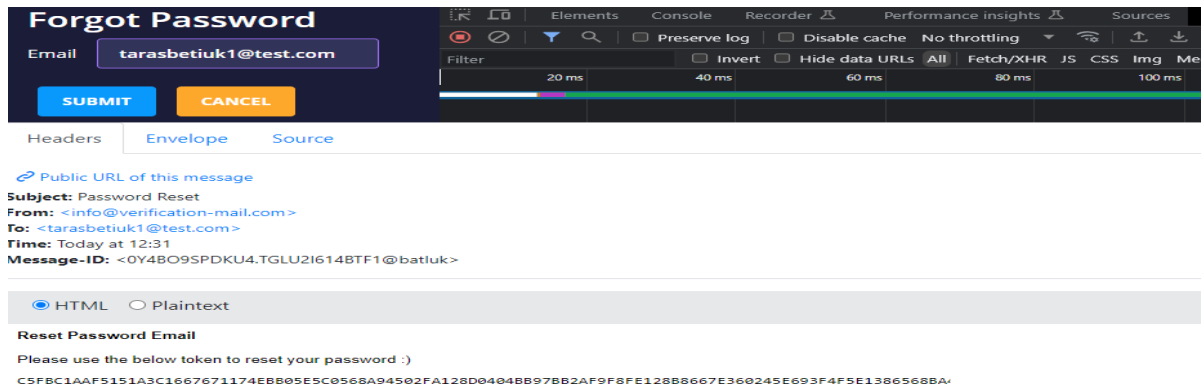


Fig. 14. Recovering a user password and saving it to the database

So, by implementing the basic authentication and authorization algorithms, creating mechanisms for changing, viewing and restoring user data and dividing users into roles with distributed access, a full-fledged authentication and authorization service was created in the intelligent system.

CONCLUSIONS AND PROSPECTS FOR FURTHER RESEARCH

During the work, optimized authentication and authorization algorithms were implemented in the intelligent system, thanks to which system users can register and authorize with secure password storage, confirm email as a means of communication with the system and recover a lost password. First, an analysis of existing research and publications was carried out, which determined which algorithms should be used for hashing user passwords, their advantages and disadvantages.

Modern mechanisms for authentication and saving user sessions using JWT access and refresh tokens and their correct implementation to ensure the maximum possible system security were also considered. Next, a system analysis was carried out, namely, using flowcharts and diagrams, the implementation of user password hashing algorithms was conceptually designed, the implementation of each algorithm was considered step by step with possible use cases, advantages and disadvantages. Also, using UML diagrams, the main mechanisms of the system's operation were designed, such as user request endpoints, which described all the main components of the intelligent system and their interaction with each other.

In the process of implementing the intelligent system, the basics of hashing and the SHA-512 algorithm were considered, the HMACSHA-512 and bcrypt hashing algorithms were also implemented, which provide reliable protection of the user password, thanks to double hashing and a conceptually different hash implementation system, due to which it is practically impossible to find out the user's password either by brute force or by using rainbow tables due to the use of the advantages of both algorithms for hashing user data. Next, JWT access and refresh tokens were implemented, thanks to which the mechanism for saving a secure user session works, where the access token with user data is stored in localStorage for only 10 minutes and is replaced with a new token thanks to the refresh token, which is stored in an http-only cookie file, which makes it impossible to steal or replace the refresh token. In addition to the basic algorithms, an intelligent system with authentication and authorization functionality was generally implemented through the interaction of two backend and frontend programs using client-server architecture. The possibilities of working in the system for a registered user in accordance with his role and the policies provided to him were implemented, thanks to which the user can change any data. Also, work with e-mail was implemented using the SMTP protocol, which allowed the system user to confirm the account and change the password quickly and safely. As a result, reliable optimized algorithms for authentication and authorization of users in the intelligent system were implemented.



REFERENCES

1. Lee, K., & Yim, K. (2023). Analysis of key storage mechanism of asymmetric key-related functions in CNG crypto library. *Applied Sciences*, 13(11), 6510. <https://doi.org/10.3390/app13116510>
2. Pervan, B., Knezović, J., & Guberović, E. (2022). Energy-efficient distributed password hash computation on heterogeneous embedded system. *Automatika*, 63(3), 399–417. <https://doi.org/10.1080/00051144.2022.2042115>
3. Belenky, Y., et al. (2023). Carry-based differential power analysis (CDPA) and its application to attacking HMAC-SHA-2. *Transactions on Cryptographic Hardware and Embedded Systems*, 1–29. <https://doi.org/10.46586/tches.v2023.i3.1-29>
4. Ramdani, F. C., Rahmatulloh, A., & Shofa, R. N. (2023). Implementation of JSON web token on authentication with HMAC SHA-256 algorithm. *SISTEMASI*, 12(1), 194. <https://doi.org/10.32520/stmsi.v12i1.2450>
5. Abdel Hakeem, S. A., & Kim, H. (2022). Centralized threshold key generation protocol based on Shamir secret sharing and HMAC authentication. *Sensors*, 22(1), 331. <https://doi.org/10.3390/s22010331>
6. Hirose, S., & Shikata, J. (2023). Aggregate entity authentication identifying invalid entities with group testing. *Electronics*, 12(11), 2479. <https://doi.org/10.3390/electronics12112479>
7. Lalem, F., Laouid, A., Kara, M., Al-Khalidi, M., & Eleyan, A. (2023). A novel digital signature scheme for advanced asymmetric encryption techniques. *Applied Sciences*, 13(8), 5172. <https://doi.org/10.3390/app13085172>
8. Zgurovsky, M. Z., Yefremov, K., Gapon, S., & Pyshnograiev, I. (2023). Research of food security problems of the war-torn regions of Ukraine using geomatics methods. *System Research and Information Technologies*, (1), 7–22. <https://doi.org/10.20535/srit.2308-8893.2023.1.01>
9. Melnykov, S., Malezhyk, P., Gasanov, A., & Bidyuk, P. I. (2022). Methodological aspects of operative control system intellectualization for dynamic objects. *System Research and Information Technologies*, (4), 44–57. <https://doi.org/10.20535/srit.2308-8893.2022.4.04>
10. Pozdniakova, M. (2025). Strategy for protecting personal data in machine learning systems. *Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique»*, 3(31), 414–423. <https://doi.org/10.28925/2663-4023.2025.31.1016>
11. Tkachenko, K., Tkachenko, O., & Tkachenko, O. (2025). Intelligent learning systems: Transforming tabular data of educational content in the ontological model. *Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique»*, 3(31), 512–527. <https://doi.org/10.28925/2663-4023.2025.31.1043>
12. Zou, J., et al. (2023). Trusted access authentication technology for large-scale heterogeneous terminals in a vehicle charging network system. *Electronics*, 12(8), 1882. <https://doi.org/10.3390/electronics12081882>
13. Otta, S. P., Panda, S., Gupta, M., & Hota, C. (2023). A systematic survey of multi-factor authentication for cloud infrastructure. *Future Internet*, 15(4), 146. <https://doi.org/10.3390/fi15040146>
14. Čučko, Š., Keršič, V., & Turkanović, M. (2023). Towards a catalogue of self-sovereign identity design patterns. *Applied Sciences*, 13(9), 5395. <https://doi.org/10.3390/app13095395>
15. Diao, Z., & Sun, F. (2023). A deep-learning neural network approach for secure wireless communication in the surveillance of electronic health records. *Processes*, 11(5), 1329. <https://doi.org/10.3390/pr11051329>

**Батюк Тарас Миронович**

доктор філософії, асистент кафедри «Інформаційні системи та мережі»

Національний університет «Львівська політехніка», Львів, Україна

ORCID: 0000-0001-5797-594X

taras.m.batiuk@lpnu.ua

Ларук Юрій Валерійович

к.т.н., асистент кафедри «Інформаційні системи та мережі»

Національний університет «Львівська політехніка», Львів, Україна

ORCID: 0000-0001-5514-6312

yurii.v.laruk@lpnu.ua

ІМПЛЕМЕНТАЦІЯ ОПТИМІЗОВАНИХ АЛГОРИТМІВ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ В ІНТЕЛЕКТУАЛЬНІЙ СИСТЕМІ

Анотація. Запропоновано імплементацію алгоритмів bcrypt та HMACSHA512, а також використання JWT Access та Refresh токенів для забезпечення надійної автентифікації та авторизації в інтелектуальній системі. Розглянуто роль цих алгоритмів у підвищенні безпеки паролів, захисті від підробки даних та забезпеченні ефективного керування доступом. Було докладно описує процес імплементації кожного алгоритму та їхнє використання в різних сценаріях. Виокремлено важливість використання потужних алгоритмів для забезпечення безпеки та конфіденційності в інтелектуальних системах, особливо з урахуванням зростаючих загроз кібербезпеці. Описано особливості створення, обробки та зберігання JWT Access та Refresh токенів на сервері зі збереженням закодованої та захешованої інформації в базі даних та в клієнтській частині програми зі збереженням даних в локальному сховищі та з використанням http-only cookies. Усі наведені алгоритми описано за допомогою формул та концептуальних діаграм, після чого наведено програмну реалізацію автентифікації та авторизації в інтелектуальній системі.

Ключові слова: bcrypt; HMACSHA512; автентифікація; авторизація; JWT Access та Refresh токени.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lee, K., & Yim, K. (2023). Analysis of key storage mechanism of asymmetric key-related functions in CNG crypto library. *Applied Sciences*, 13(11), 6510. <https://doi.org/10.3390/app13116510>
2. Pervan, B., Knezović, J., & Guberović, E. (2022). Energy-efficient distributed password hash computation on heterogeneous embedded system. *Automatika*, 63(3), 399–417. <https://doi.org/10.1080/00051144.2022.2042115>
3. Belenky, Y., et al. (2023). Carry-based differential power analysis (CDPA) and its application to attacking HMAC-SHA-2. *Transactions on Cryptographic Hardware and Embedded Systems*, 1–29. <https://doi.org/10.46586/tches.v2023.i3.1-29>
4. Ramdani, F. C., Rahmatulloh, A., & Shofa, R. N. (2023). Implementation of JSON web token on authentication with HMAC SHA-256 algorithm. *SISTEMASI*, 12(1), 194. <https://doi.org/10.32520/stmsi.v12i1.2450>
5. Abdel Hakeem, S. A., & Kim, H. (2022). Centralized threshold key generation protocol based on Shamir secret sharing and HMAC authentication. *Sensors*, 22(1), 331. <https://doi.org/10.3390/s22010331>
6. Hirose, S., & Shikata, J. (2023). Aggregate entity authentication identifying invalid entities with group testing. *Electronics*, 12(11), 2479. <https://doi.org/10.3390/electronics12112479>
7. Lalem, F., Laouid, A., Kara, M., Al-Khalidi, M., & Eleyan, A. (2023). A novel digital signature scheme for advanced asymmetric encryption techniques. *Applied Sciences*, 13(8), 5172. <https://doi.org/10.3390/app13085172>
8. Zgurovsky, M. Z., Yefremov, K., Gapon, S., & Pyshnograiev, I. (2023). Research of food security problems of the war-torn regions of Ukraine using geomatics methods. *System Research and Information Technologies*, (1), 7–22. <https://doi.org/10.20535/srit.2308-8893.2023.1.01>
9. Melnykov, S., Malezhyk, P., Gasanov, A., & Bidyuk, P. I. (2022). Methodological aspects of operative control system intellectualization for dynamic objects. *System Research and Information Technologies*, (4), 44–57. <https://doi.org/10.20535/srit.2308-8893.2022.4.04>



10. Pozdniakova, M. (2025). Strategy for protecting personal data in machine learning systems. *Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique»*, 3(31), 414–423. <https://doi.org/10.28925/2663-4023.2025.31.1016>
11. Tkachenko, K., Tkachenko, O., & Tkachenko, O. (2025). Intelligent learning systems: Transforming tabular data of educational content in the ontological model. *Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique»*, 3(31), 512–527. <https://doi.org/10.28925/2663-4023.2025.31.1043>
12. Zou, J., et al. (2023). Trusted access authentication technology for large-scale heterogeneous terminals in a vehicle charging network system. *Electronics*, 12(8), 1882. <https://doi.org/10.3390/electronics12081882>
13. Otta, S. P., Panda, S., Gupta, M., & Hota, C. (2023). A systematic survey of multi-factor authentication for cloud infrastructure. *Future Internet*, 15(4), 146. <https://doi.org/10.3390/fi15040146>
14. Čučko, Š., Keršič, V., & Turkanović, M. (2023). Towards a catalogue of self-sovereign identity design patterns. *Applied Sciences*, 13(9), 5395. <https://doi.org/10.3390/app13095395>
15. Diao, Z., & Sun, F. (2023). A deep-learning neural network approach for secure wireless communication

Отримано редакцією журналу / Received: 10.06.26

Прорецензовано / Revised: 22.07.26

Схвалено до друку / Accepted: 26.10.26



This work is licensed under Creative Commons Attribution-noncommercial-sharealike 4.0 International License.