Modern, Secure Application Programming Interface Implementation using RFC 6238 and RFC 7617

by Yaman Khaeruzzaman

Submission date: 10-May-2023 03:23PM (UTC+0700)
Submission ID: 2089325325
File name: Modern,_Secure_Application_Programming_Interface.pdf (797.12K)
Word count: 4816
Character count: 26589

Modern, Secure Application Programming Interface Implementation using RFC 6238 and RFC 7617

1st Nicholas Dwiarto Wirasbawa
Faculty of Engineering & Informatics
Universitas Multimedia Nusantara
Jakarta, Indonesia
nicholas.dwiarto@student.umn.ac.id

2nd Yaman Khaeruzzaman
Faculty of Engineering & Informatics
Universitas Multimedia Nusantara
Tangerang, Indonesia
yaman.khaeruzzaman@umn.ac.id

3rd Alexander Waworuntu
Faculty of Engineering & Informatics
Universitas Multimedia Nusantara
Tangerang, Indonesia
alex.wawo@umn.ac.id

***Abstract*—Application Programming Interface (API) is an interface that could be used to access or utilize services on the Internet. API has several advantages, namely being technology agnostic and becoming a middleware before a request arrives at the database. A Report from Salt Security claims that 30% of security vulnerabilities of APIs in 2021 comes from authentication problems. Two-factor authentication in the form of Time-Based One-Time Passwords (TOTP) is a method that could be implemented to achieve data integrity, secure transactions, and protection of private resources in API. A potential combination of algorithms that could be used to generate and send TOTPs are RFC 6238 and RFC 7617. An API connected to a web application is built in order to provide proof of concept which showcases that this combination is feasible to implement in an API. Technologies used are Express.js, Next.js, MariaDB, and Redis. Implementation is done in a simulation of the attendance system to find an alternative to existing attendance systems. The result of the research showcases that the whole system has been built, deployed, and conforms to OWASP Standards (metrics being Authorization, Authentication, API Security, OTP Security, and Session Security) and Snyk. The additional implementation of the simulation of the attendance system is also well accepted by application users, evidenced by an acceptance rate of 91.81% by the Technology Acceptance Model. This research has proven the algorithms used are secure without sacrificing usability and can be implemented in any API. This research also opens up possibilities of attendance systems being implemented with TOTPs.**

Keywords—Application Programming Interface, Attendance System, RFC 6238, RFC 7617, Two-Factor Authentication

I. Introduction

An Application Programming Interface (API) is an interface used to access or utilize resources on the Internet. Using an API is proven to be a scalable solution that allows for complete separation of concerns and flexibility, in addition to increased security and ease of migrations. Naturally, APIs are technology agnostic, meaning that everything and everyone could utilize them as long as the usage conforms to the right interface of that particular API [1].

Throughout 2021, Salt Security reports that API attacks have increased by 681%. The development of APIs from December 2020 to December 2021 shows drastic improvements, which means that more people starts to use APIs as the foundation of their software systems. The two most prominent reasons for using an API are development efficiencies (58%) and platform integrations (44%). Data from the same source also shows that 95% of API users have experienced security vulnerabilities in the last 12 months. Three security vulnerabilities that are the most commonly experienced are generic vulnerability (39%), authentication problem (32%), and sensitive data exposure (30%) [2].

Software developers must look beyond typical authentication methods to solve one of the most prominent vulnerabilities, the authentication problem. One way to solve the problem is to implement Two-Factor Authentication or Multi-Factor Authentication (2FA/MFA) [3] to augment 'something you know' with 'something you have' or 'something you are'. To implement multi-factor authentication in APIs, a potential combination of algorithms that could be used are RFC 6238 [4], which is a Time-Based One-Time Password (TOTP), and RFC 7617 [5], which is Basic Authentication. RFC 6238 is naturally stateless and supports the zero-knowledge protocol, which is suitable for generation, while RFC 7617 is a simple yet secure authentication schema which is good for delivery. This potential could be used to serve as a foundation for the protection of private resources in an API.

One of the implementations which could use TOTPs is attendance systems. In practice, attendance is usually performed with biometrics, radio-frequency identification (RFID) [6], and even manually. Due to the Covid-19 pandemic, these systems have to be perceived with caution as they break social distancing protocols by having to perform contact with the machine. Biometric systems also have common disadvantages, namely being able to reach false positive and false negative rates by up to 5% [7]. Another disadvantage is when the human in question is injured in the biometric location or if the human works with chemical matters, the ratio of false positive and false negative rates is highly likely to increase [8]. There has also not yet been any research about attendance systems being implemented with one-time passwords, which coincidentally is also a 'something you have' [9], the same as RFID. From these facts, the research will implement the API in a simulation of an attendance system to find out the alternative methods of attendance systems described above.

Previous research has implemented one-time passwords in login systems [10] [11]. Both research claim that the usage of one-time passwords improves the security of the system. However, both researches do not use and elaborate on how to create an API to implement said authentication protocols. In addition, there has not yet been any research about the combination of RFC 6238 and RFC 7617, and there has not yet been any research about how to secure a system that implements these authentication protocols completely. Most famous systems are closed-source and their implementation details cannot be examined.

This research will focus on creating an API for a simulation of an attendance system that is generally secure according to Open Web Application Security Project (OWASP) metrics while implementing the RFC 6238 and RFC 7617 algorithms [12] [13] [14]. The API will interface

with a web application for users to test the simulation of the attendance system. The evaluation of the application will be done by OWASP metrics and Technology Acceptance Model (TAM) [15]. With this system, people can learn how to implement generally secure API with Two-Factor Authentication and find alternatives to existing attendance systems. This paper contains the introduction, methodology and system design, implementation, evaluation, and conclusion.

II. Methodology and System Design

This section will explain the research methodology used in this research and the algorithms that will be used, and it will also explain the system design chosen to fulfill the research objectives.

A. *Time-Based One-Time Password*

Time-Based One-Time Password (TOTP), or RFC 6238, is an algorithm used to generate time-based one-time passwords securely. It has the advantage of being stateless and naturally implements the zero-knowledge protocol. TOTP is built based on an HMAC-Based One-Time Password (HOTP) [16]. The base algorithm to generate the one-time password is described in Equation 1.

$$TOTP = Truncate(F(K,C)) \quad (1)$$

Truncate itself is a dynamic truncation function to truncate the resulting bits from the hashing algorithm into a bytes string. F is a one-way hashing algorithm. Applicable hashing algorithms are SHA−1, SHA−256, and SHA−512. K is the key of the hashing algorithm, and C is the counter value described in Equation 2.

$$C = Floor((T_1 - T_0)/X) \quad (2)$$

T_1 stands for the current time in the UNIX timestamp. T_0 stands for the initial counter time, also in UNIX timestamp (defaults to zero). X stands for the period of password validity. *Floor* is a generic mathematical floor function. Variables T_0 and X have to be initialized at the starting point of the application in order to keep everything uniform. If, after the whole process, the resulting one-time password does not have the required digits, the OTP will be padded with zero(es) until the required length is fulfilled. TOTPs can be stored in Authenticator applications due to their zero-knowledge nature and their ability to become stateless. It is possible to create one-time password authentication strings prefixed with "*otpauth://*" to store them in said applications [4].

B. *Basic Authentication*

Basic authentication, formally known as RFC 7617: The 'Basic' HTTP authentication scheme, is a method used to send credentials on the wire, usually used in APIs. It is not secure by default and must be augmented and combined with Secure Socket Layer / Transport Layer Security (SSL/TLS) protocol to achieve high-grade security. Basic authentication itself is sent over a network header, more specifically, the 'Authorization' header. This authentication scheme has several characteristics that differentiate it from other authentication schemes:

1) The authentication scheme name is 'Basic';
2) The authentication parameter 'realm' has to be filled in;
3) The authentication parameter 'charset' is optional, but if filled, it must be filled with 'UTF-8'; and
4) Verifier has to ignore other authentication parameters if it exists.

The credential itself takes the form of a Base64 encoded string [17] in the format of 'identifier:password' (split by a colon, then encoded using Base64). It is not possible for an identifier to have control characters and the colon symbol. There are no restrictions on choosing the characters of passwords. An example of basic authentication in a network header with the identifier being 'basic' and password being 'auth' is 'Basic YmFzaWM6YXV0aA=='. In conclusion, basic authentication is a secure and proper way to send credentials over the wire, as long as it is augmented with SSL/TLS, and as long as the server implements good security practices [5].

C. *Use-Case Diagram*

Fig. 1 shows the use-case diagram of the system. There are three entities. A normal user has the ability to refresh authenticator secrets (a common feature in authenticator applications), post their own attendance, check their own attendance report, see and edit their own profile, see and invalidate their sessions, deactivate their account, and register. An administrator possesses the ability to perform create, read, update, and delete operations on users, and also the ability to see all cumulative attendances in the system. The system itself also has the ability to send check-out reminders and send emails regarding security (on password changes or if the user fails to input the TOTP three times). Several of the use-cases require the user to be authenticated with multi-factor or two-factor authentication (MFA/2FA) by verifying their TOTP. Some use-cases do not require the user to be authenticated with MFA, and normally signing in is enough.

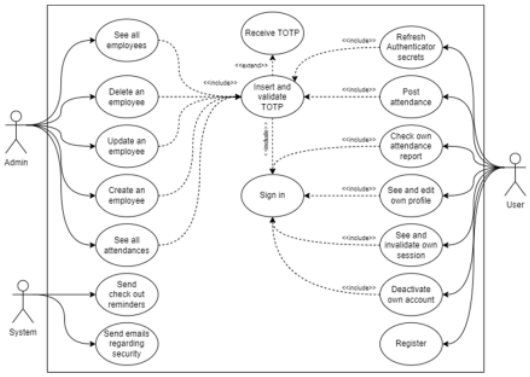
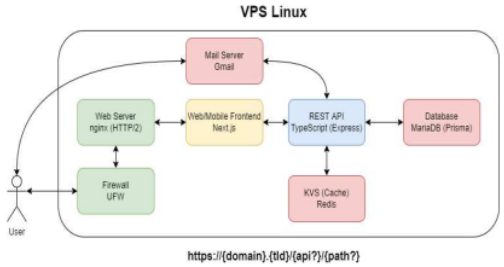


Fig. 1. Use-case diagram of the system.

D. *System Architecture*

Fig. 2 showcases the architecture of the system. Before a request can be successfully made, the request has to pass through the Uncomplicated Firewall (UFW) and the web server / reverse proxy (Nginx). The API itself is built with Representational State Transfer (REST) architecture [18], powered by Express.js, and it will act as the middleware to process requests before they reach the related infrastructures. API will only accept JavaScript Object Notation (JSON) as its content type, and will respond to requests with JSON only.

The website part of the system is powered by Next.js with ChakraUI [19] as its user-interface framework. The website will communicate with the API to 'hydrate' its components,



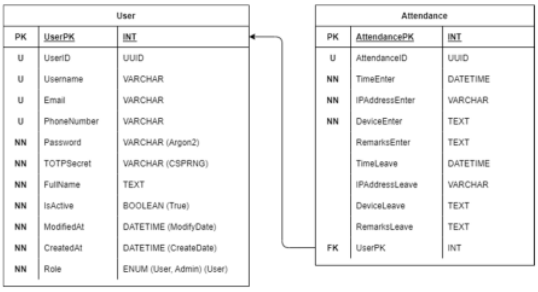
essentially making the website rely on the API.

Fig. 2. The architecture of the whole system

The databases used in this system are Redis [20] as the key-value store (also serving as a cache to store volatile data) and MariaDB [21] as the primary database to store non-volatile data. Databases are backed up periodically using cron jobs. To send TOTPs, the system implements an Email service (Gmail) and Authenticator applications. The whole application lives inside a Linux server, with the operating system being Debian 11 for servers. The transport layer security (TLS) certificate is using Let's Encrypt for secure HyperText Transfer Protocol Secure (HTTPS) transport. The system is accessible via a domain name.

E. Database Schema

Fig. 3 shows the SQL database schema of the system. It is divided into two entities: User and Attendance. In the schema, PK is the primary key, FK is the foreign key, U is a unique constraint, and NN is not null constraint. The relationship between the tables is one user has many attendances, and one attendance is related to a single user.



User

PK	UserPK	INT
U	UserID	UUID
U	Username	VARCHAR
U	Email	VARCHAR
U	PhoneNumber	VARCHAR
NN	Password	VARCHAR (Argon2)
NN	TOTPSecret	VARCHAR (CSPRNG)
NN	FullName	TEXT
NN	IsActive	BOOLEAN (True)
NN	ModifiedAt	DATETIME (ModifyDate)
NN	CreatedAt	DATETIME (CreateDate)
NN	Role	ENUM (User, Admin) (User)

Attendance

PK	AttendancePK	INT
U	AttendanceID	UUID
NN	TimeEnter	DATETIME
NN	IPAddressEnter	VARCHAR
NN	DeviceEnter	TEXT
	RemarksEnter	TEXT
	TimeLeave	DATETIME
	IPAddressLeave	VARCHAR
	DeviceLeave	TEXT
	RemarksLeave	TEXT
FK	UserPK	INT

Fig. 3. SQL database schema of the system

The attributes that are stored in the User database are as follows:

1) 'UserPK' is the user's primary key and is stored as an integer.
2) 'UserID' is the user's public ID and is stored in the form of UUID V4 [22].
3) 'Email' is the user's email.
4) 'PhoneNumber' is the user's phone number.
5) 'Password' is the user's password and is stored in a hashed form using Argon2 key-derivation function algorithm [23].
6) 'TOTPSecret' is a random seed to generate the user's TOTP tokens.
7) 'FullName', which is a user's full name.
8) 'IsActive', which is a user's activation status.
9) 'ModifiedAt', which is a user's last data modification date.
10) 'CreatedAt', which is a user's creation date.
11) 'Role', which is a user's role, whether they are an admin or a normal user.

The attributes that are stored in the attendance database are as follows:

1) 'AttendancePK' is an attendance's primary key and is stored as an integer.
2) 'AttendanceID' is an attendance's public ID and is stored in the form of UUID V4 [22].
3) 'IPAddress', 'DeviceEnter', 'RemarksEnter,' and their 'Leave' counterparts, which are the user's internet protocol (IP) address, the user's device from the user agent network header, and optional comments.
4) 'UserPK' is a foreign key to connect a single attendance to a single user ID.

The public ID is used in order not to expose the primary key to the outside environment. Furthermore, in the REST API, data considered sensitive will not be revealed in the API response. Data considered sensitive are hashed passwords, TOTP secrets, usernames, and primary keys.

F. Cache Schema

Fig. 4 presents the data structure used to store the volatile data of the system or to store the data in the cache. The data structure used in Redis is a key-value type that has a certain time to live (TTL). The TTL in the schema means the time that a key has left before it is deleted from the cache. In other words, The TTL is the time that the data remains in the cache before being deleted automatically when it has passed the existing TTL.

Redis Key-Value

TTL	Key	Value
30s	AskedOTP:{UserID}	IsTrue
2m	BlacklistedOTP:{UserID}:{OTP}	UserID
24h	OTPAttempts:{UserID}	Count
15m	OTPSession:{SessionID}	UserID
2h	Sess:{SessionID}	SessionData
15m	SlowDown:{IPAddress}	Count
15m	{Domain}:RateLimit:{IP}	Count
15m	Email-Lock:{UserID}	IsTrue

Fig. 4. Cache schema of the system

The data stored in the cache are as follows:

1) 'AskedOTP:{UserID}' is to know whether a user has recently asked for an OTP to prevent SPAM.
2) 'BlacklistedOTP:{UserID}:{OTP}' is to know whether a user has recently verified an OTP to prevent replay attacks.
3) 'OTPAttempts:{UserID}' is to know the number of attempts a user has tried and failed to verify their OTP.
4) 'OTPSession:{SessionID}' is to store the integrity of a special session token that is received after the user has validated their OTP code. The special session ID takes the form of a secure opaque (random) string, and on the client-side, it is stored in a JSON Web Token [24], more precisely in JWT ID (JTI) attribute.
5) 'Sess:{SessionID}' is to store the normal session token gained after successfully authenticating with username and password combination (initial authentication).
6) 'SlowDown:{IP Address}' is used to store the internet protocol address to throttle if the IP address makes too many requests.
7) '{Domain}:RateLimit:{IP}' is used to store rate limiters to prevent an IP address from making too many requests.
8) 'EmailLock:{UserID}' is used to prevent SPAM in user emails by accident.

G. *One-Time Password Verification*

Fig. 5 showcases how the system performs a one-time password verification procedure by combining RFC 6238 and RFC 7617 algorithms. The flowchart proposes a methodology and procedure to safely verify a one-time password, which is as follows:

1) The server should verify that the user has a valid session.
2) The server should fetch the 'Authorization' network header and ensures that the header is valid.
3) The server should then fetch the 'identifier' and 'password' part of the Basic Authentication.
4) The 'identifier' part of the basic authentication should fetch a user identifier from the main database. In this application, the user ID is used. The 'password' part of the basic authentication should be a one-time password.
5) The server should fetch from the cache, if the user has exceeded the login attempt limitations, the server should reject the request and send a security alert notification to that specific user.
6) The server should increment the attempts count by one and then perform a check to know if the OTP has been blacklisted or not. This is to prevent replay attacks.
7) The server should validate the one-time password based on the server and user parameters. This includes the secret, the allowed time drift, the digits, and the hashing algorithm (SHA-1/SHA-256/SHA-512).
8) Once the validation process is completed, the server should send a special session token to keep track of the special authorized session, meaning a session that the user has gained after performing two-factor authentication. The server should return a proper status code at this point (200 OK is recommended).
9) As an additional note, the server should return the proper status codes on failed requests, such as 400 Bad Request, 401 Unauthorized (on failed verification), 410 Gone (on blacklisted one-time passwords), and 429 Too Many Requests.
10) It is recommended that the server set this endpoint as HTTP PUT, as it is semantically correct (replacing a resource in a server). Nonetheless, any other HTTP method is fine as long as the cache control is not set to prevent caching.
11) The server should not log session cookies and authorization headers.

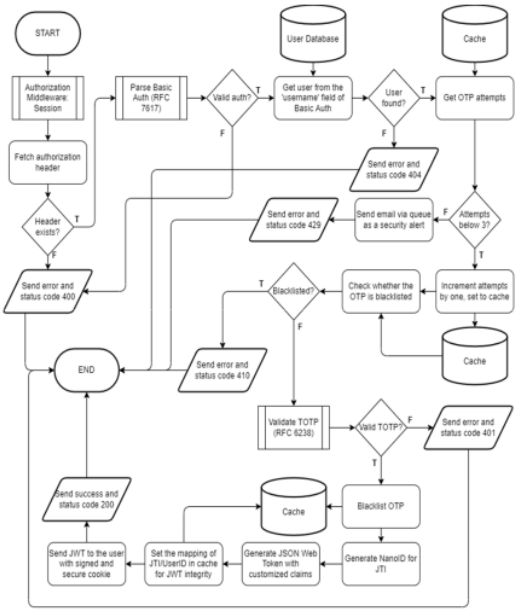


Fig. 5. Secure OTP verification process

III. Implementation

We have successfully implemented our REST API and web application with the following specifications.

A. *Application Programming Interface*

The API itself conforms to JSON:API Standards [25] format in its response body. The API will perform several preprocessing steps before the request is executed, as shown in Fig. 6.

The API will validate a request based on several metrics, such as the 'Accept' header, the 'Content-Type' header, the cross-site request forgery (CSRF) prevention header, the rate limit and slowdowns, the IP address, the 'Authorization' header, and session cookies. The API will return the relevant

status code errors if it encounters any errors. The request will be logged and stored in a log file for audit purposes. An example of the successful API response is shown in Fig. 7

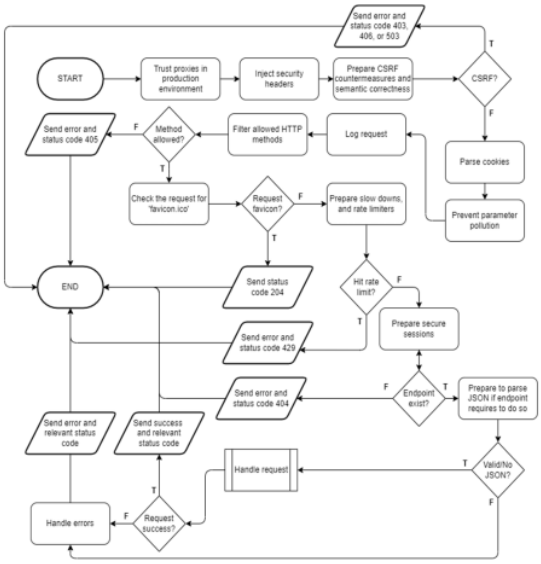


Fig. 6. API request processing steps

```
{
    "status": "success",
    "message": "Welcome to Attendance API!",
    "data": [],
    "type": "general",
    "meta": {
        "copyright": "© 2022 – Nicholas",
        "authors": [
            "Nicholas"
        ]
    },
    "jsonapi": {
        "version": "v1"
    },
    "links": {
        "self": "http://192.168.1.2/api/v1"
    }
}
```

Fig. 7. API success response

B. *Website*

The website is created as a media to interface with the API. The website is coupled with the API, so it is essentially a blank page without data to 'hydrate' it. Web pages created include the homepage, profile page, attendance page, and administrator page. The website is designed to be responsive and supports the best practices of Web Accessibility Standards (a11y). The homepage of the website is shown in Fig. 8.

When a user has successfully logged in, they can validate themselves by Two-Factor Authentication, as shown in Fig. 9. After authenticating themselves with the one-time password, the user can post their attendance, whether it is checking in or checking out.

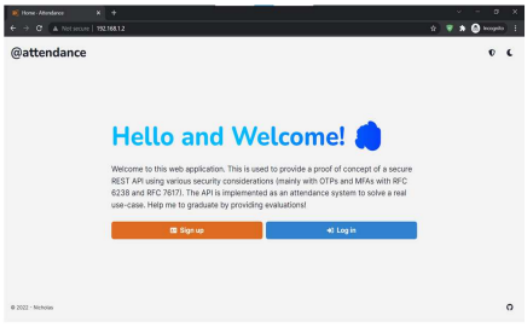


Fig. 8. Homepage of the website

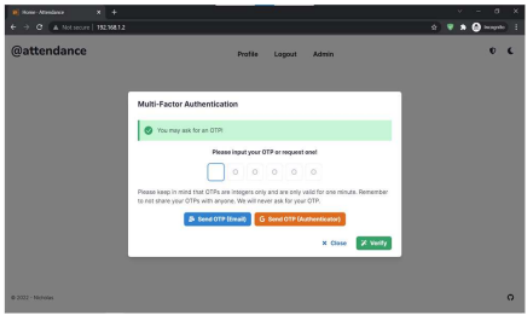


Fig. 9. One-time password verification modal

The profile page allows the user to see and manipulate their own data and also see their attendance. Users are also able to change their passwords securely (meaning that all current sessions are deleted, and they have to log in again with their new password). Users are also able to refresh their authenticator secrets, and users also have control over their own sessions. The administrator is able to perform create, read, update, and delete operations on users, and also see all attendances data. The administrator page is shown in Fig. 10.

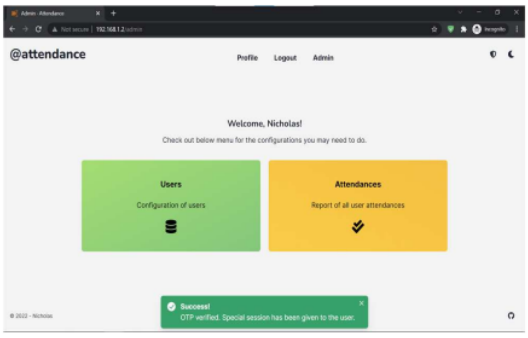


Fig. 10. Administrator page

Finally, the whole application is error-tested, meaning that if a user tries to access a non-existent page and an unauthorized page, the website will show the proper error page. As this research is about the implementation details, the source code of this research may be found at https://github.com/lauslim12/attendance.

IV. EVALUATION

For the evaluation part of the application, we performed white-box and black-box testing to test and analyze the source code and the live version of the application. Evaluation of application security used OWASP metrics, specifically authentication testing, authorization testing, API security, OTP security, session security, and Zed Attack Proxy. Technology Acceptance Model (TAM) is used to test the acceptance rate of the system.

A. Functionalities Evaluation

From the functionalities evaluation, everything is working fine, and there are no bugs in the application. All features conform to the Use-Case Diagram as described in Fig. 1.

B. Security Evaluation

For the security evaluation part of the research, the system has been tested with OWASP metrics, which are authentication testing, authorization testing, API security, OTP security, session security, and Zed Attack Proxy. The whole application passed all of the security penetration tests. Several parts of the application also use a random identifier named NanoID [26]. It is a cryptographically secure random string generator, and it is tested with the simplex method; by generating thousands of IDs and then trying to find any duplicates. Additionally, Snyk [27] is used to check for vulnerabilities in the application and to assess the code style. All parts of the application have proven themselves to be secure.

C. Acceptance Evaluation

The attendance application has been tested by users. A total of 59 respondents helped to fill out questionnaires. The respondents are students from Universitas Multimedia Nusantara and surrounding universities. Table I shows the list of questionnaire opinion statements that should be answered using the Likert scale [28]. The questionnaire is designed to assess the acceptance rate of the application. According to the Technology Acceptance Model, three metrics are used: ' Ease of Use,' 'Usefulness,' and 'Security and Usability.' The survey reveals that the ease of use ratio is 94.37%, the usefulness ratio is 89.63%, and the security and usability ratio is 94.23%. On average, the acceptance rate is 91.81%. This shows that the application is well-accepted by respondents and opens up the potential to use one-time passwords as an alternative to existing attendance systems.

TABLE I. LIST OF QUESTIONNAIRE OPINION STATEMENTS

No.	Opinion Statement	Metric
1	The attendance application is easy to use.	Ease of Use
2	The attendance application has a good and flexible display or user interface.	Ease of Use
3	The attendance application can perform every function and features well.	Ease of Use
4	The attendance application made me realize that conventional attendance systems can be converted into digital-based systems.	Usefulness
5	The attendance application intrigued me to make attendance using an OTP.	Usefulness
6	The attendance application can provide an alternative to the existing conventional attendance system.	Usefulness
7	The attendance application can send an OTP relatively fast through the available methods.	Ease of Use
8	The attendance application has good performance.	Ease of Use
9	The attendance application makes it easy to do presence in everyday life.	Usefulness
10	The attendance application can save you time when making attendance.	Usefulness
11	The attendance application makes you interested in implement an OTP-based attendance system for the future (if you are responsible for the attendance of an institution, event, and so on).	Usefulness
12	The Attendance application has an OTP feature using an authenticator application, such as Google Authenticator, and you will find that using this application is very practical for OTP delivery problems.	Usefulness
13	After using the Attendance application, you feel the level of security is very good when using it.	Security and Usability

V. CONCLUSION

A. Summary

In this research project, we have successfully built a secure API with Two-Factor Authentication via RFC 6238 and RFC 7617. We have also built a web application that functions as a media to interface / communicate with the API. The application is built with Express.js, Next.js, MariaDB, and Redis, with the main programming language being TypeScript. The application has been deployed securely in a Debian 11-powered Linux server, complete with TLS certificate and a domain name. From the results of the Open Web Application Security Project and Snyk evaluations, the application is declared to be secure. From the results of the Technology Acceptance Model, the application nets an acceptance rate of 91.81%, meaning that the application is well-accepted by the users. According to the respondents, this application can also serve as an alternative to already existing attendance systems.

B. Future Development

For future research, several additional suggestions are provided as follows:

1) The API has proven itself to be secure. It is recommended to 'extract' the API authentication and authorization part to be a standalone Identity Provider.
2) Add additional protocols, such as OAuth2, SAML, and/or OpenID Connect to allow external clients to delegate their authentication and authorization to the constructed API.
3) Implement additional implementation details and improve upon existing ones, such as migrating the system into a microservices-based architecture to increase scalability and allow replication for high availability (HA).
4) Perform evaluations in a real-life use-case scenario of an attendance system to know the objective effectiveness of the system.

ACKNOWLEDGMENT

The authors would like to thank Universitas Multimedia Nusantara for the support and facilities given for this research project.

REFERENCES

[1] N. D. Wirasbawa, D. R. Wibawanto, A. Kosasi, and S. Hansun, "Scalable building management system for offices and co-working spaces," Indian Journal of Economics and Business, vol. 20, no. 2, 2021.

[2] S. Security, "State of api security q1 2022," Mar 2022. [Online]. Available: https://salt.security/api-security-trends

[3] U. Khushlani, "The importance of otp and 2 factor authentication for customer security," 9 2020. [Online]. Available: https://msg91.com/guide/the-importance-of-otp-and-2-factor-authorization-for-customer-security/

[4] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "Totp: Time-based one-time password algorithm," Internet Requests for Comments, RFC Editor, RFC 6238, May 2011, https://www.rfc-editor.org/rfc/rfc6238.txt. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6238.txt

[5] J. Reschke, "The 'basic' http authentication scheme," Internet Requests for Comments, RFC Editor, RFC 7617, September 2015.

[6] T. Lim, S. Sim, and M. Mansor, "Rfid based attendance system," 2009 IEEE Symposium on Industrial Electronics and Applications, 2009.

[7] P. Wadhwa, "Attendance system using android integrated biometric fingerprint recognition," International Research Journal of Engineering and Technology, vol. 4, no. 6, 2017.

[8] R. Saini, "Comparison of various biometric methods," International Journal of Advances in Science and Technology (IJAST), vol. 2, no. 1, Mar 2014.

[9] V. Maty´a˘s and Z. ˘R´ıha, "Biometric authentication — security and usability," Advanced Communications and Multimedia Security, p. 227–239, 2002.

[10] W. S. Raharjo, I. D. E. Ratri, and H. Susilo, "Implementasi two factor authentication dan protokol zero knowledge proof pada sistem login," Jurnal Teknik Informatika dan Sistem Informasi, vol. 3, 4 2017.

[11] H. Setiawan, D. Sartika, and B. G. Ramadhan, "Implementasi time-based one time password (totp) pada sistem two-factor authentication (2fa)," Jurnal Teknologi, vol. 13, 6 2020.

[12] F. OWASP, "Wstg - stable," 2021. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/

[13] ——, "Owasp application security verification standard," October 2022. [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/

[14] Z. Banach, "The owasp api security top 10," Aug 2020. [Online]. Available: https://www.invicti.com/blog/web-security/owasp-api-security-top-10/

[15] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," MIS Quarterly, vol. 13, no. 3, p. 319, 1989.

[16] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "Hotp: An hmac-based one-time password algorithm," Internet Requests for Comments, RFC Editor, RFC 4226, December 2005.

[17] S. Josefsson, "The base16, base32, and base64 data encodings," Internet Requests for Comments, RFC Editor, RFC 4648, October 2006, https://www.rfc-editor.org/rfc/rfc4648.txt. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4648.txt

[18] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[19] ——, "Create accessible React apps with speed," October 2022. [online]. Available: https://chakra-ui.com/

[20] ——, "Redis." October 2022. [online]. Available: https://redis.io/

[21] ——, "About MariaDB Server," October 2022. [online]. Available: https://mariadb.org/about/

[22] P. J. Leach, M. Mealling, and R. Salz, "A universally unique identifier (uuid) urn namespace," Internet Requests for Comments, RFC Editor, RFC 4122, July 2005, https://www.rfc-editor.org/rfc/rfc4122.txt. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4122.txt

[23] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," 2016 IEEE European Symposium on Security and Privacy (EuroSP), 2016.

[24] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," Internet Requests for Comments, RFC Editor, RFC 7519, May 2015, https://www.rfc-editor.org/rfc/rfc7519.txt. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7519.txt

[25] Y. Katz, "Json:api," 2015. [Online]. Available: https://jsonapi.org/

[26] A. Sitnik, "Ai/nanoid: A tiny (130 bytes), secure, url-friendly, unique string id generator for javascript," 2020. [Online]. Available: https://github.com/ai/nanoid

[27] ——, "Open source risk management made for developers," October 2022. [online]. Available: https://snyk.io/product/open-source-security-management/

[28] Likert, R., "A technique for the measurement of attitudes," Archives of Psychology, 1932.

Modern, Secure Application Programming Interface Implementation using RFC 6238 and RFC 7617

ORIGINALITY REPORT

15%	14%	4%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	kc.umn.ac.id Internet Source	7%
2	www.ashwinanokha.com Internet Source	2%
3	ejournals.umn.ac.id Internet Source	1%
4	informatica.si Internet Source	1%
5	docs.aws.amazon.com Internet Source	<1%
6	www.coursehero.com Internet Source	<1%
7	www.researchgate.net Internet Source	<1%
8	Fransiskus Aprilion Aric, Alexander Waworuntu. "Android-based Decision Support System in Laptop Selection Using ELECTRE	<1%

	Method", 2021 6th International Conference on New Media Studies (CONMEDIA), 2021 Publication	
9	www.amrita.edu Internet Source	<1 %
10	www.turcomat.org Internet Source	<1 %
11	"Table of Contents", 2023 International Conference on Smart Computing and Application (ICSCA), 2023 Publication	<1 %
12	Submitted to Academic Library Consortium Student Paper	<1 %
13	Minoli, . "Security Mechanisms and Approaches", Security in an IPv6 Environment, 2008. Publication	<1 %
14	repository.tudelft.nl Internet Source	<1 %
15	Matthew Baker. "Secure Web Application Development", Springer Science and Business Media LLC, 2022 Publication	<1 %
16	link.springer.com Internet Source	<1 %

17	Edwin Handoko, Dennis Gunawan. "Parabolix: Educational Simulation Game on Classical Mechanics Based on Virtual Reality and Perlin Noise Algorithm", 2019 5th International Conference on New Media Studies (CONMEDIA), 2019 Publication	<1 %
18	diglib.tugraz.at Internet Source	<1 %
19	M. Lycett. "Understanding 'variation' in component-based development: case findings from practice", Information and Software Technology, 2001 Publication	<1 %
20	github.com Internet Source	<1 %
21	ntnuopen.ntnu.no Internet Source	<1 %
22	projekter.aau.dk Internet Source	<1 %
23	whaagmans.nl Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 7 words
Exclude bibliography	On		