PHISHING-RESISTANT MULTI FACTOR AUTHENTICATION

BY

KUEK EN YEE

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology
(Kampar Campus)

FEBRUARY 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Gan Ming Lee, for his invaluable guidance and support throughout the development of this project. Discussions with him on system architecture and the various components of the authentication system have shaped the direction of my work. His constant encouragement, especially during times when I felt like giving up, motivated me to push through and complete the project. Through his mentorship, I have gained a deep understanding of authentication concepts and learned how to set up both the frontend and backend of a secure system. I am sincerely grateful for his guidance.

ABSTRACT

The advent of the internet has revolutionized how people connect and interact, but it also brings various severe consequences such as data leakage when weak authentication method is implemented. Two-factor authentication(2FA) is a widely adopted method, yet vulnerabilities have been discovered to bypass it. Traditional 2FA typically combines something a user knows (like a password) with something they have (like a temporary verification code from a physical device). However, this approach is still susceptible to attacks such as phishing attacks, especially Real-Time Phishing(RTP) attack. Location-based multi authentication (MFA) methods have been proposed to mitigate RTP techniques that exploit traditional OTP-based verification. By replacing the OTP mechanism with a geolocation verification step, it add a layer of security to the authentication process. Limitations on previously proposed location-based multi-factor authentication are additional user step for adaptation, mobile-based, and additional costs due to hardware requirements. This paper suggests incorporating location as an extra security layer while maintaining the user-friendliness and seamlessness of existing two-factor authentication methods, thus creating a user-friendly and seamless multifactor authentication solution. The performance of this proposed authentication method is evaluated against various attacks, notably RTP attack.

Area of Study: Cybersecurity, Authentication

Keywords: Multi-Factor Authentication, Real-Time Phishing Attack, MERN Stack, Location Verification, Two-factor Authentication, Phishing Attack

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AS	Authentication Service
BSON	Binary JSON
cURL	Client URL
CORS	Cross-Origin Resource Sharing policy
DoS	Denial-of-Service
DOM	Document Object Model
FCM	Firebase Cloud Messaging
GPS	Global Positioning System
HMAC	Hash-based Message Authentication Code
HTML5	Hypertext Markup Language 5
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
IRS	Internal Revenue Service
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
LQI	Link Quality Indicator
MA	Mobile Application
MFA	Multi-factor Authentication
MITM	Man-in-the-Middle Attack
MPOS	Mobile Application Stores the Message

LIST OF ABBREVIATIONS

NFC	Near-Field Communication
NoSQL	Non-SQL (Not Only SQL) Databases
OTP	One-Time Password
PGT	Pre-shared Number, GPS Location, and Time Stamp
PHP	Hypertext Preprocessor
PIN	Personal Identification Number
POS	Point of Sale
POS_ID	POS's Identification Number
PUFs	Physically Unclonable Functions
RDBMS	Relational Databases Management Systems
RNDM	Identification Number and a Random Number
RSA	Rivest-Shamir-Adleman (Cryptosystem)
RSSI	Received Signal Strength Indicator
RTA	Real-Time Phishing Attack
SCDOR	South Carolina Department of Revenue
SHA	Secure Hash Algorithm
SMS	Short Message Service
SQL	Structured Query Language
TTL	Time to Live
TLS	Transport Layer Security
URL	Uniform Resource Locator
2FA	Two-factor Authentication
WA	Witness Application

CHAPTER 1

INTRODUCTION

1.1 Project Background

Authentication is crucial for protecting systems and accounts, ensuring only authorized access to sensitive data. Ineffective authentication mechanisms can be as dangerous as having no protection at all, as illustrated by major breaches like the 2012 South Carolina Department of Revenue hack. These incidents highlight the need for robust authentication methods. Two-Factor Authentication (2FA) and Multi-Factor Authentication (MFA) address these needs by combining multiple verification factors: knowledge, possession, and inherence. To counter sophisticated attacks like social engineering, incorporating location-based authentication offers an additional layer of security, further protecting against unauthorized access.

1.1.1 Overview of Authentication

Authentication is essential for safeguarding systems and accounts, ensuring that only authorized users can access sensitive information. It serves multiple purposes, including protecting data, preventing unauthorized access, maintaining accountability, preserving trust on system and so on. However, ineffective authentication mechanism can be as insecure as having no authentication at all.

1.1.2 Authentication Attacks

Most vulnerabilities in authentication mechanisms arise from two main sources: weak protection against brute-force attacks and logic flaws or poor coding in the implementation, allowing attackers to bypass authentication entirely—a phenomenon often referred to as "broken authentication."

A notable incident occurred in 2012 when a foreign hacker stole 387,000 credit card numbers and 3.6 million Social Security numbers from the South Carolina Department of Revenue (SCDOR). This breach was significant as it exposed sensitive personal information, leading to further consequences [1]. Subsequently, the Internal Revenue Service (IRS) experienced another breach in 2015, affecting over 700,000 individuals and resulting in the exposure of their personal data, including Social Security numbers and addresses. These incidents facilitated identity fraud [2]. In study conducted by Javelin Strategy & Research, at the same year of SCDOR cybersecurity incident, the total fraud losses was found to reach record high and fraud cases identified in 2016 reaches 15.4 million[3].

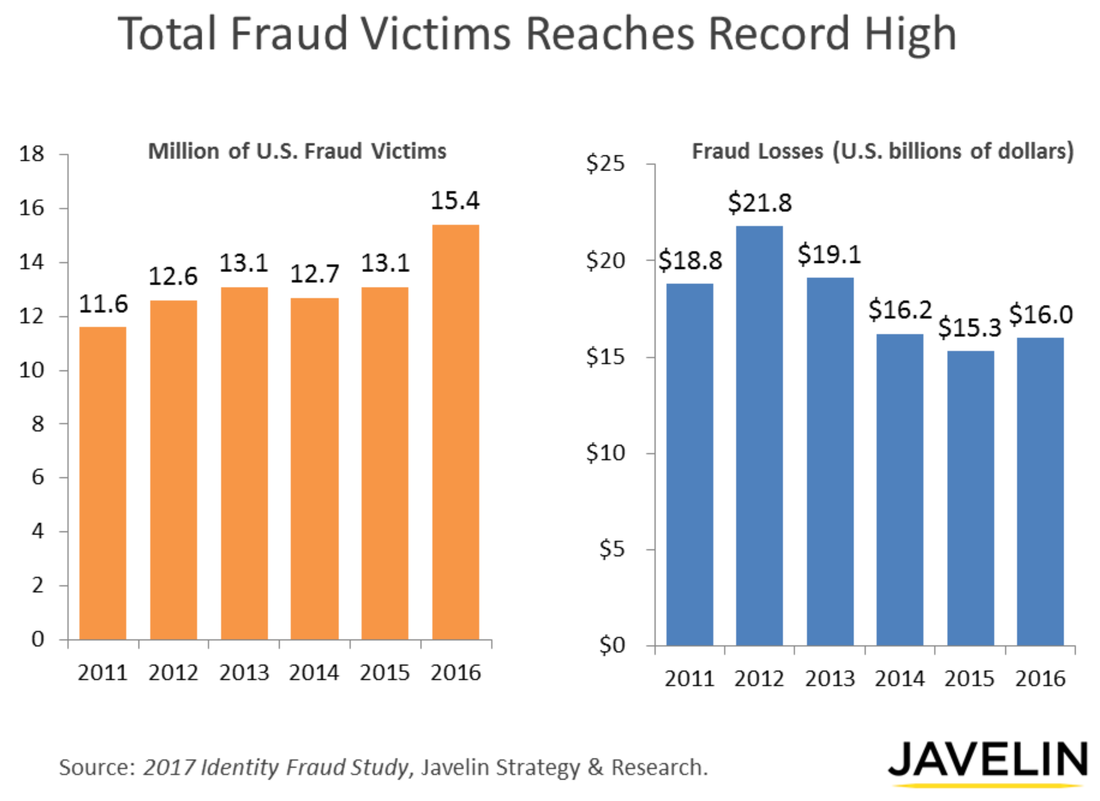


Figure 1.2.1 2017 Identity Fraud Study [3]

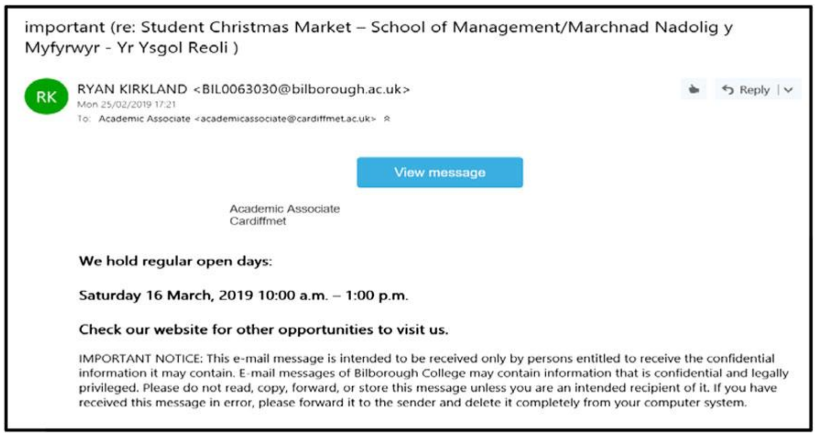
1.1.3 Two-Factor Authentication Systems

Therefore, security experts have been highly innovative in developing effective two- factor authentication (2FA) standing out as a valuable example of their effort. It is one type of multifactor authentication which combines traditional password-based authentication with another factor. The factors typically fall into one of three categories: knowledge factors, possession factors, and inherence factors.

Knowledge factors require information only the user knows, such as a password, PIN, or responds to security questions. It remains fundamentally important in modern world. Possession factors need tangible items possessed by the user, such as a smartphone, token, or smart card. This may include receiving a code through a text message or utilizing an authentication app. While effective in principle, possession factors present notable usability challenges. Research indicates 35% of users encounter availability issues with their secondary authentication devices, and approximately 25% perceive backup codes as offering no substantive security advantage over conventional passwords.

Inherence factor needs something inherent to the user, such as a fingerprint, retinal scan, or facial recognition. In traditional 2FA, there is typically a combination of something the user knows, such as a password, with something they possess, like a temporary verification code from a physical device.

However, threat actors have been sophisticated in manipulating individuals into providing both their credentials and the second factor (the verification code sent via SMS or email) through social engineering and phishing techniques. Real-time phishing (RTP) being one of the newer attacks allows attacker to gain the additional factor and thus undermine the security of systems that rely solely on transmitted data for authentication, highlighting the need for robust countermeasures.



important (re: Student Christmas Market – School of Management/Marchnad Nadolig y Myfyrwyr - Yr Ysgol Reoli)

RK RYAN KIRKLAND <BIL0063030@bilborough.ac.uk>
Mon 25/02/2019 17:21
To: Academic Associate <academicassociate@cardiffmet.ac.uk>

Academic Associate
Cardiffmet

We hold regular open days:

Saturday 16 March, 2019 10:00 a.m. – 1:00 p.m.

Check our website for other opportunities to visit us.

IMPORTANT NOTICE: This e-mail message is intended to be received only by persons entitled to receive the confidential information it may contain. E-mail messages of Bilborough College may contain information that is confidential and legally privileged. Please do not read, copy, forward, or store this message unless you are an intended recipient of it. If you have received this message in error, please forward it to the sender and delete it completely from your computer system.

Figure 1.2.2 Screenshot of a coronavirus related phishing email [4].

1.1.4 Real-time Phishing Workflow

Among contemporary phishing techniques, real-time phishing(RTP) attack is particularly a sophisticated threat vector. As illustrated in Figure below, RTP operates through a man-in-the-middle architecture where attackers intercept communications between legitimate users and target websites. The attack begins with the malicious actor delicately constructing a counterfeit website. This counterfeit domain is then distributed through conventional phishing channels to potential victims.

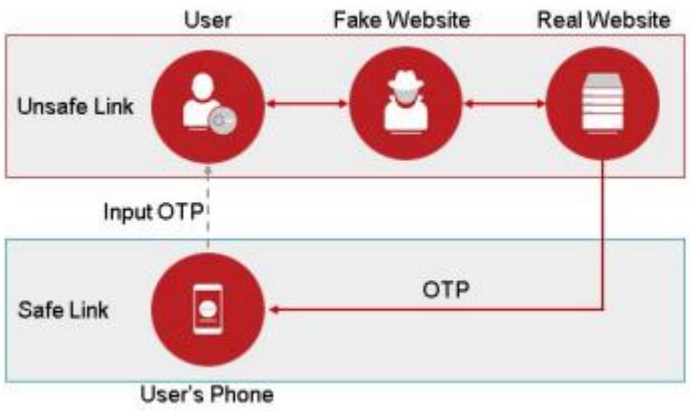


Figure 1.2.3 Real-Time Phishing workflow. [18]

The attack progresses when a user accesses the spoofed domain. Due to the fraudulent website's identical visual presentation to the legitimate website, Bob enters his username and password for the legal website on the fake phishing site. The fraudulent website immediately relays these credentials to the genuine website. Now the legitimate website establish session with the adversary which impersonates the victim. To verify the login, the legitimate website sends a onetime password (OTP) to victim's phone. The victim input OTP to the malicious website as they input their credentials. As a result, the attacker completes the authentication process with the credentials and second knowledge factor, thereby gaining full access to the victim's account. This entire compromise occurs without detection by either the user or the genuine website. [18]

Nowadays, an adversary can effortlessly establish a phishing website without any background of the web page design technique. Gophish is one of the frameworks used by educators to implement phishing exercise. It provides a web-based interface that allows users to design and customize emails and landing pages, track responses and clicks. [16]

1.2 Problem Statement and Motivation

The RTP attack exploits a critical weakness in time-based OTP implementations, where the brief validity window of authentication codes is vulnerable to immediate relay attacks. The attack proves particularly effective because it bypasses conventional security awareness measures - users interact with what appears to be a normal authentication flow, complete with expected OTP prompts. When victims reach the 2FA stage, they have typically already failed to recognize earlier phishing indicators in the email and website, leaving the 2FA service as the final potential point of security intervention. However, traditional 2FA systems lack contextual awareness about the user's actual interaction point with the service.

To address this vulnerability, we propose location-based multifactor authentication as an enhanced security measure. By incorporating the user's geographical location as an additional authentication factor, location-based authentication adds an extra layer of security, making it more challenging for attackers to impersonate legitimate users. This approach leverages HTML5 Geolocation API to verify the physical whereabouts of users, reducing the likelihood of successful RTP attacks and enhancing overall security.

1.3 Project Scope

The project aims to develop an advanced authentication system that integrates traditional email/password authentication with location-based MFA. This system will be designed to enhance security while ensuring user-friendliness and adaptability. The project will cover both backend and frontend development, including server-side logic, user interface design, and database management. Key components of the system include user registration, login, location verification, domain verification and email-based MFA.

1.4 Project Objectives

1. To develop a location-based MFA system that mitigates the risk of phishing attack.

   This objective will focus on designing and implementing a multi-factor authentication mechanism that prevent unauthorized access even when user's login credentials are compromised

2. To design a user-friendly interface and minimize the steps required for users to adapt to the new multi-factor authentication system, ensuring a smooth transition.

   This objective will be achieved by providing clear guide during the authentication and minimize steps for authentication.

3. To build a backend API that facilitates the integration of location-based multi-factor authentication, streamlining the implementation process.

   A backend system will be developed with API connection capability to facilitates different domain integrates the MFA authentication into their existing system, contributing to the modern cybersecurity area.

4. To integrate additional features against brute-force attempts.

   Some mechanisms will be implemented to mitigate the risk of brute-force attacks: rate-limiting, account lockout, logging.

5. To evaluate the system's security and usability.

System’s resilience against multiple attacks and user experience will be discussed.

1.5 Contributions

1. The research seeks to combat the increasing threat of phishing attacks by conducted a technical proof of concept of location-based MFA. This novel method uses geolocation data to provide an additional layer of user identity validation, thereby enhancing the overall effectiveness of phishing attack prevention.
2. The project emphasizes user experience by designing a user-friendly interface and minimizing the steps required for users to adapt to the new multi-factor authentication system.
3. The project demonstrates practical application of advanced cybersecurity concepts, such as HMAC for data integrity

1.6 Report Organization

This report is organized into seven chapters. Chapter 1 outlines the background of authentication systems, RTP attack and the motivation for developing a location-based MFA system. It likewise outlines the problem, scope, objectives, contributions, and the structure of the report. Chapter 2 is literature review, it discusses on current location-based multi-factor authentication methods, and emphasizes the advantages and drawbacks of them. This review forms the basis for proposing a more robust solution.

Chapter 3 outlines the system's methodology and design strategy, elaborating on the architecture, use cases, and technologies used in the development process. It also covers the software development principles applied in the project. Chapter 4 details the system’s functionalities, including registration, login, location verification, and the dashboard. It additionally defines the database schemas with descriptions in tables.

A more detailed technical explanation on how implemented components is offered in Chapter 5. Chapter 6 assesses the system's performance and security under different test scenarios and attacks, analyzing the system’s effectiveness, and discusses the challenges faced during project development, and evaluate the achieved objectives. Lastly, Chapter 7 concludes the report by summarizing the findings and offering future recommendations for enhancing the system.

CHAPTER 2

Literature Review

2.1 Pre-shared Number, GPS Location, and Timestamp

A new mobile-based multi-factor authentication scheme called PGT (Pre-shared number, GPS location, and Time stamp) is proposed for securing user’s web accounts like Gmail, Yahoo. It uses a modifiable pre-shared number, user's GPS location and timestamp to generate a one-time password (OTP) hash as an additional security factor. The pre-shared number is not transmitted during authentication. Attackers will be unable to generate the PGT token and login the server even when user’s credentials are obtained [6].

Initially, users authenticate themselves by entering their username and password, serving as the first factor of authentication. Subsequently, a security token is generated through the user's mobile app. A combination of factors including a pre-shared number, the device's current GPS location, and a timestamp is used for the security token generation. Upon token generation, the authentication server receiving the token will try to access the user's GPS location and timestamp from a designated GPS server. The server then generates its own token using identical parameters and compares it against the token provided by the user. Authentication is granted if the tokens match [5].

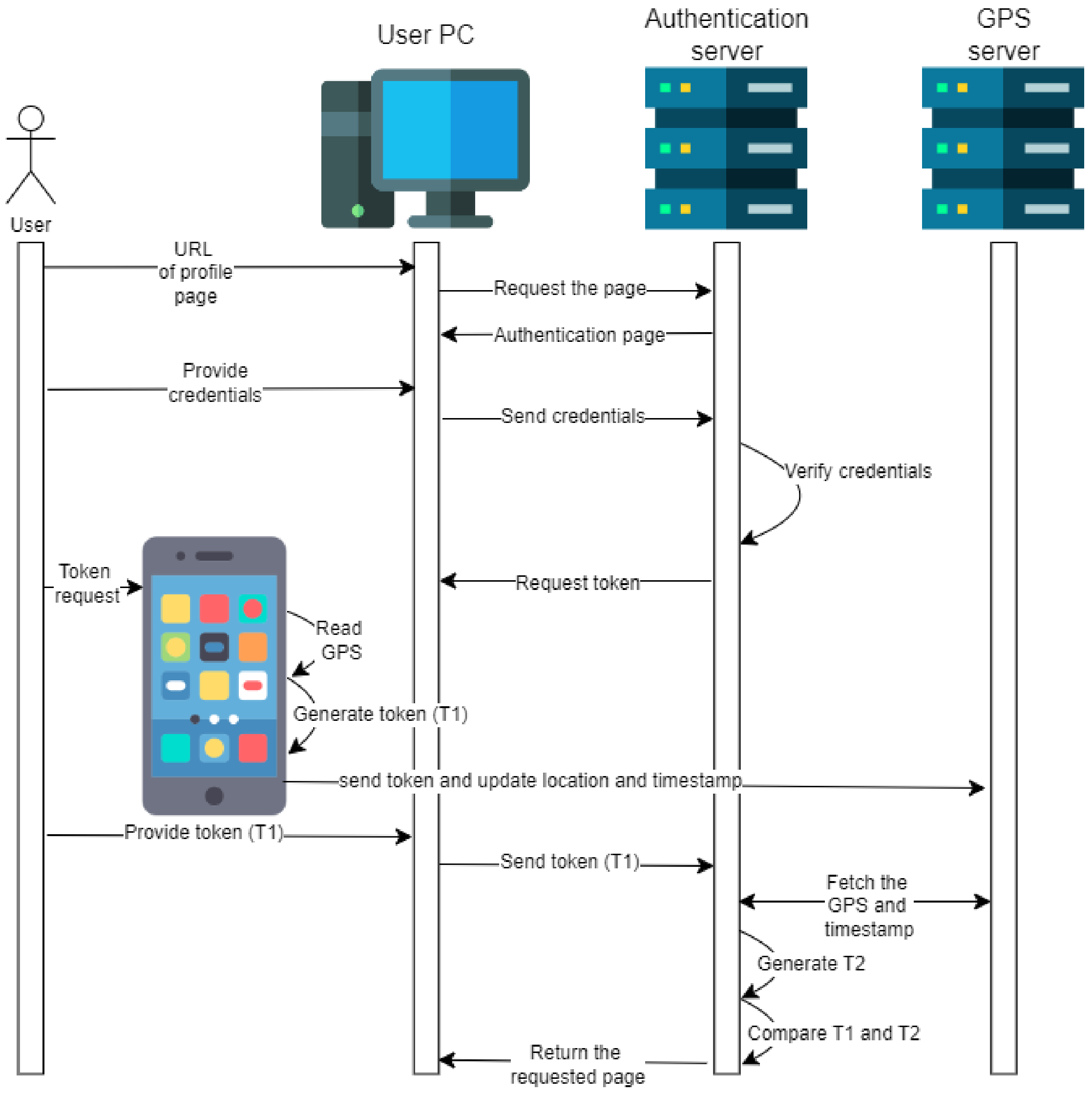


Figure 2.1.1 The flow of PGT authentication process.

2.1.1 Strengths of Pre-shared Number, GPS location, and Timestamp

The scheme utilizes multiple factors including password GPS location, pre-shared number to enhance security. Moreover, it is cost-effective as it does not require any additional hardware tokens or SMS costs. Furthermore, the scheme mitigates potential replay attacks, where an attacker could capture and later retransmits data that was exchanged between two parties, but OTP can only be used once. For duplicate-generator attack, where the victim's OTP generator can be duplicate by learning the seed and the algorithm, is also mitigated by having the PGT to be modifiable instead of using fixed seed like SecurID [6].

2.1.2 Weaknesses of Pre-Shared Number, GPS Location, And Timestamp

The system design faces several notable limitations that could impact its security and functionality. Firstly, ensuring the security of the pre-shared number poses a critical challenge, necessitating robust encryption and storage mechanisms to prevent unauthorized access. Secondly, the system is susceptible to mirroring attacks, where users unknowingly provide their credentials and generated token or pre-shared number to fraudulent websites, compromising their security. Additionally, the reliance on mobile device security introduces vulnerability, as compromised devices could expose the pre-shared number, enabling attackers to generate valid tokens. Lastly, the paper fails to elaborate on the secure reset or update procedure for the pre-shared number, posing a potential vulnerability if not managed carefully. For instance, exchanging the pre-shared number verbally between a user and a server administrator could leave it susceptible to eavesdropping attacks.

2.2 Location-Based Multi-Factor Authentication Using PUFs, RSSI And LQI

A two-factor authentication proposed by Muhammad and Biplab includes location proximity between gateway and IoT devices as another factor of authentication. It uses physically unclonable functions (PUFs) and wireless channel characteristics such as received signal strength indicator (RSSI) and link quality indicator (LQI) [7].

Some key challenges in authentication for Internet of Things (IoT) devices has been identified by the author such as the secret has to be stored on device's memory, making IoT devices vulnerable to physical attack. The physical attacks IoT devices vulnerable are, for example, side-channel attack, environmental attack and tampering with hardware. Thus, PUFs has been proposed to solve the challenges faced by IoT [6].

PUFs is for the identity verification of a chip or device. Inside a chip, there are tiny variations or imperfections that are impossible to duplicate exactly. These imperfections might be caused during manufacturing or due to the natural properties of materials. When measure these imperfections is measured, a unique pattern that's specific to that chip can be obtained. By comparing the measured pattern to the original pattern stored securely, you can confirm if it's the real chip or not. It's a clever way to add a layer of security because even if someone tries to copy the chip, they won't get the exact same PUF pattern. In addition to PUFs, the system leverages wireless channel characteristics such as received signal strength indicator (RSSI) and link quality indicator (LQI) to authenticate devices based on their proximity to a designated gateway. By comparing these values against pre-calculated thresholds for the specific area, the system can verify the device's location and ensure its authenticity [7].

2.2.1 Strengths of Location-Based Multi-Factor Authentication Using PUFs, RSSI And LQI

By utilizing physically unclonable functions (PUFs) and wireless channel characteristics, the system provides protection against physical attacks, cloning attacks, spoofing attacks, and denial-of-service (DoS) attacks. Another notable strength lies in its elimination of the need for storing secret keys on IoT devices, thus rendering them resilient to physical attacks. Moreover, the inclusion of a location verification feature adds an extra layer of security, preventing remote attackers from masquerading as legitimate IoT devices. Additionally, the system use lightweight symmetric-key cryptography, making it suitable for deployment in resource constrained IoT environments [7].

2.2.2 Limitations of Location-Based Multi-Factor Authentication Using PUFs, RSSI And LQI

However, this approach isn't suitable for web application authentication on personal computers and mobile devices. The reason being accessing RSSI from a web browser is not possible due to security and privacy restrictions. Besides, it relies on the fact that the gateway is trustable as well.

2.3 Previous Work: Location-Based Multi-Factor Authentication Using Witness Device and Application

Another location-based multifactor authentication is proposed for mobile environments. The protocol involves various components: the user's mobile application (MA), a witness application (WA) on another user's device nearby, a POS terminal, and a backend service with an authentication service (AS) which uses firebase cloud messaging (FCM) as a notification service. Point of sale (POS) refers to a device that is used to process transactions in retail business. It can be a physical device, or a checkout point in an online store. However, the paper focuses on physical POS. These components interact as shown in the Figure [8].

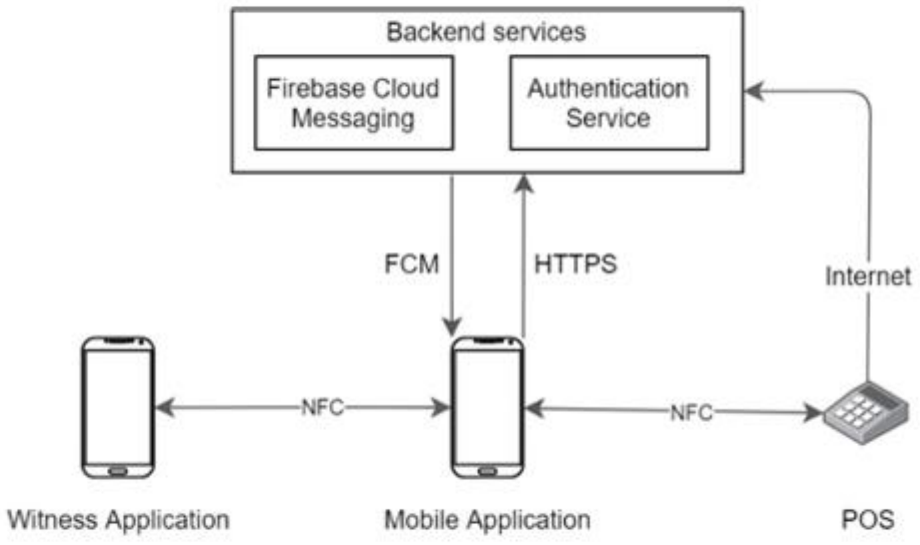


Figure 2.3.1 The system design of method using witness application.

The AS will have user identity and some keys for decryption, encryption, and verification of signatures from different components. It also has exact location of POS and will lock the location until the authentication process is finished.

For mobile application (MA) and witness application (WA), they both will generate RSA keys in registration process, one for digital signatures and another for decryption of messages sent by others. Both devices should be equipped with GPS to retrieve highly accurate location data and an NFC interface.

During the first phase, trusted location point is encrypted and sent by MS to AS. The location point is trusted after confirmation by witness applications (WS). MS and WS will exchanges their location data via NFC interface which limits the distance between the user performing the authentication process and the witness application of another registered user.

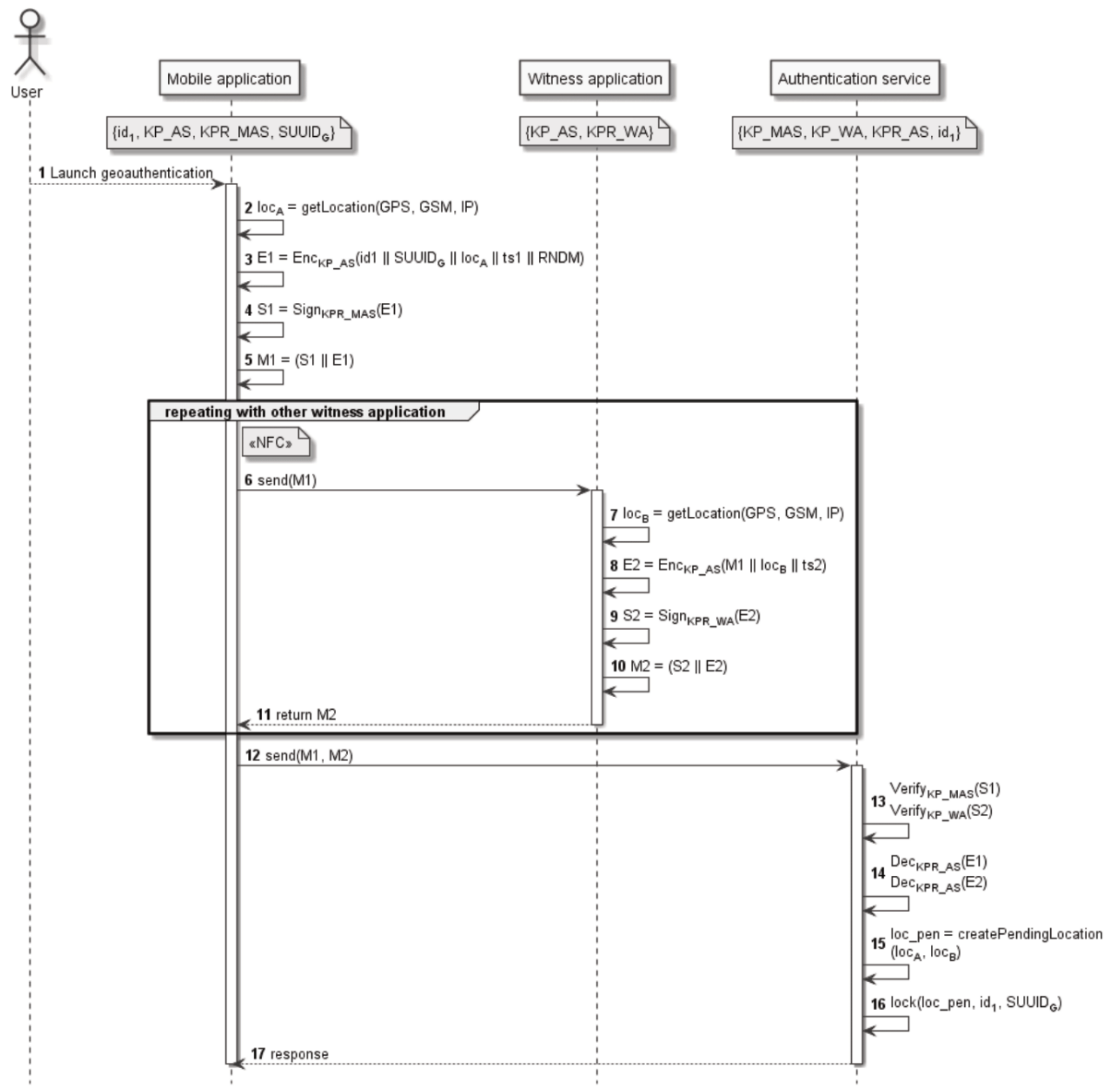

User
Mobile application
Witness application
Authentication service
{id1, KP_AS, KPR_MAS, SUUIDG}
{KP_AS, KPR_WA}
{KP_MAS, KP_WA, KPR_AS, id1}
1 Launch geoauthentication
2 locA = getLocation(GPS, GSM, IP)
3 E1 = EncKP_AS(id1 || SUUIDG || locA || ts1 || RNDM)
4 S1 = SignKPR_MAS(E1)
5 M1 = (S1 || E1)
repeating with other witness application
«NFC»
6 send(M1)
7 locB = getLocation(GPS, GSM, IP)
8 E2 = EncKP_AS(M1 || locB || ts2)
9 S2 = SignKPR_WA(E2)
10 M2 = (S2 || E2)
11 return M2
12 send(M1, M2)
13 VerifyKP_MAS(S1) VerifyKP_WA(S2)
14 DecKPR_AS(E1) DecKPR_AS(E2)
15 loc_pen = createPendingLocation (locA, locB)
16 lock(loc_pen, id1, SUUIDG)
17 response

Figure 2.3.2 The system design of method using witness application.

In the second phase, the declared user location will be confirmed with the POS participates in confirming the user's declared location. The process begins with the mobile application discovering nearby POS using Hello messages. Once a POS is discovered, it sends its identification number and a random number (RNDM), signed as MPOS, to the mobile application. This random number addition helps prevent replay attacks. After the mobile application stores the message (MPOS) and responds with a success message, the POS starts the geo-authentication process. The mobile application encrypts and signs the

MPOS, along with its ID and session identifier, and sends them to the POS. The POS forwards this message to the AS. Following verification of data integrity using signatures and timestamps, the AS confirms the locations of the mobile application and POS. The mobile application's location is obtained in the first phase while the location of the POS identified by its identification number is stored in the AS's storage. If the two locations match, a success message is sent back via FCM [8].

2.3.1 Strengths of Location-Based Multi-Factor Authentication Using Witness Device and Application

The authentication system designed addresses the critical issues of location spoofing and privacy attacks through a combination of multiple data sources and encryption techniques. Additionally, formal verification demonstrates the system's robustness against man-in-the-middle attacks, replay attacks, and other forms of undesirable behaviour, ensuring the integrity and security of the authentication process. Furthermore, performance evaluations indicate that the system can be efficiently implemented without significantly impacting the duration of the authentication process, making it a practical and effective solution for authentication at POS [8].

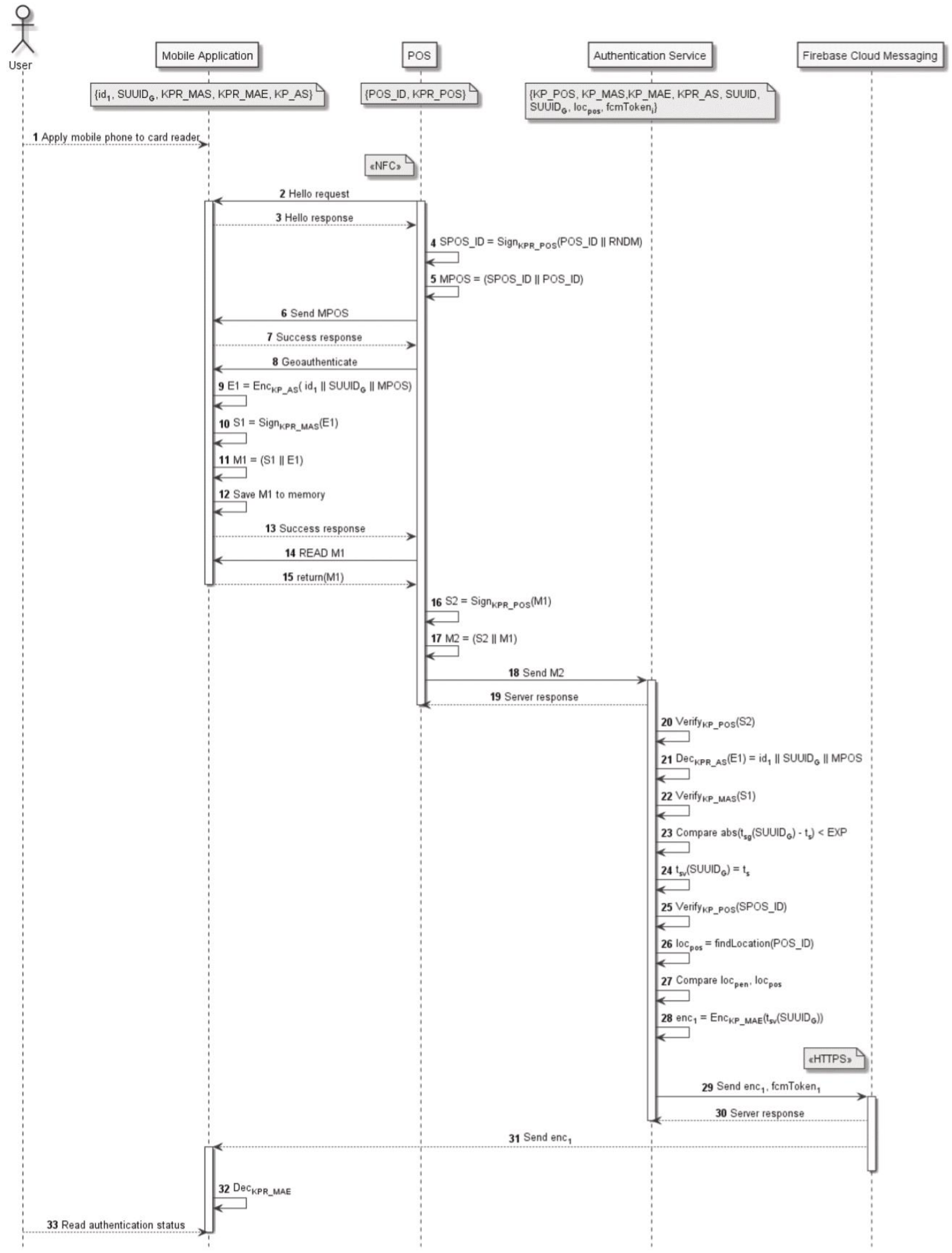

User
Mobile Application
POS
Authentication Service
Firebase Cloud Messaging
{id_1, $SUUID_G$, KPR_MAS, KPR_MAE, KP_AS}
{POS_ID, KPR_POS}
{KP_POS, KP_MAS, KP_MAE, KPR_AS, SUUID, $SUUID_G$, loc_{pos}, $fcmToken_i$}
1 Apply mobile phone to card reader
«NFC»
2 Hello request
3 Hello response
4 SPOS_ID = $Sign_{KPR\_POS}$(POS_ID ‖ RNDM)
5 MPOS = (SPOS_ID ‖ POS_ID)
6 Send MPOS
7 Success response
8 Geoauthenticate
9 E1 = Enc_{KP_AS}(id_1 ‖ $SUUID_G$ ‖ MPOS)
10 S1 = $Sign_{KPR\_MAS}$(E1)
11 M1 = (S1 ‖ E1)
12 Save M1 to memory
13 Success response
14 READ M1
15 return(M1)
16 S2 = $Sign_{KPR\_POS}$(M1)
17 M2 = (S2 ‖ M1)
18 Send M2
19 Server response
20 $Verify_{KP\_POS}$(S2)
21 Dec_{KPR_AS}(E1) = id_1 ‖ $SUUID_G$ ‖ MPOS
22 $Verify_{KP\_MAS}$(S1)
23 Compare abs(t_{sg}($SUUID_G$) - t_s) < EXP
24 t_{sv}($SUUID_G$) = t_s
25 $Verify_{KP\_POS}$(SPOS_ID)
26 loc_{pos} = findLocation(POS_ID)
27 Compare loc_{pen}, loc_{pos}
28 enc_1 = Enc_{KP_MAE}(t_{sv}($SUUID_G$))
«HTTPS»
29 Send enc_1, $fcmToken_1$
30 Server response
31 Send enc_1
32 Dec_{KPR_MAE}
33 Read authentication status

Figure 2.3.3 The flow of verification of location data by point-of-sales (POS).

2.3.2 Limitations of Location-Based Multi-Factor Authentication Using Witness Device and Application

The first limitation of the proposed approach is performance overhead. Multiple rounds of communication between MA and WA are needed to avoid GPS spoofing. Furthermore, the appearance of WA may not always be possible. For avoid the risk of replay attack, RNDM is introduced in every exchange of POS's identification number (POS_ID). If a number is used twice, the query of POS_ID is not processed, this method introduces overhead as the necessary of recording each issued RNDM in the POS to prevent reuse.

The second phase of the protocol relies on trusted POS terminals to confirm the user's location. If these terminals are compromised or unavailable, the authentication process may be hindered. The protocol is designed specifically for mobile environments and may not be easily adaptable to other scenarios where location-based multifactor authentication is desired (e.g., desktop computers, IoT devices). Users are required to install the application for doing authentication and witness others' authentication, which may be reluctant to do so. Additionally, the availability of WA may not always be guaranteed. Additionally, only one user can undergo the geo-authentication process at a location, while others are blocked until the process finishes.

2.4 Limitations of Previous Works

The previous methods of using location-based multi-factor authentication had some drawbacks. First, they added an extra step to the authentication process, which could be confusing for users and make it harder for them to adopt the method. Second, since these methods relied on GPS technology, they only worked on devices that had GPS, leaving out devices without GPS capabilities. Moreover, implementing this kind of authentication could require extra hardware, which would increase the overall cost. This could be a barrier for organizations or users who didn't want to spend more money on authentication. Overall, these limitations show the need for simpler and more affordable authentication methods that work well across different types of devices.

2.5 Proposed Method

The proposed method aims to address the limitations of traditional location-based multi-factor authentication by introducing a more flexible and user-friendly approach. Instead of relying on GPS technology, the proposed method incorporates HTML5 Geolocation API, with combined location determination methods such as Wi-Fi triangulation or IP geolocation, making the authentication method not limited to GPS-capable devices. This ensures that users can authenticate from a wider range of devices, including those without GPS capabilities, thereby enhancing accessibility and usability. Additionally, the proposed method simplifies the authentication process by minimizing the number of steps required, making it more intuitive for users to adopt. Furthermore, to mitigate the need for additional hardware components and reduce costs, the proposed method leverages existing features of devices with browser wherever possible, such as utilizing HTML5 Geolocation for location verification. Overall, the proposed method aims to offer a more versatile, cost-effective, and user-friendly approach to location-based multi-factor authentication for mobile and personal computer (PC) environment.

Category	**Pre-shared Number, GPS Location, and Timestamp**	**Location-Based Multi-Factor Authentication Using PUFs, RSSI And LQI**	**Location-Based Multi-Factor Authentication Using Witness Device and Application**	**New Approach**
Security Risks	- Pre-shared number security challenges (encryption, storage). - Susceptibility to mirroring attacks. - Vulnerability due to mobile device security.	- Reliance on the gateway being trustable, which may not always be the case.	- Reliance on trusted POS terminals, which may be compromised.	- Mitigates brute-force attacks, replay attacks, and website mirroring using additional features.
Usability Issues	- Lack of secure reset or update procedure for the pre-shared number. - Verbal exchange of pre-shared number could be vulnerable to eavesdropping.	- Not suitable for web applications on personal computers and mobile devices due to RSSI access limitations.	- Users required to install apps and witness authentication, leading to potential reluctance. - Only one user can authenticate at a time, blocking others.	- No pre-shared number required. - HMAC is generated using the website's secret key. - No reliance on RSSI. - Ensure user-friendly flow and interface
Performance/Overhead	N/A	N/A	- Performance overhead due to multiple communication rounds	- Potential latency and network overhead due to the possible use of a cloud

			between MA and WA to avoid GPS spoofing. - Overhead from recording issued RNDM to prevent reuse.	solution (needs to be examined).
Adaptability/Applicability Limitations	N/A	- Not adaptable to personal computers and mobile devices due to security restrictions on accessing RSSI from a web browser.	- Limited adaptability to non-mobile environments (e.g., desktop, IoT).	- Adaptable to any device that supports a browser and email service.

Table 2.5.1 Comparison between new project and previous work.

CHAPTER 3

System Methodology/ Approach

The system implements a MFA process that includes location verification as an additional security layer. The login process critically involves a process which backend generates a verification URL, and the user must click to confirm their physical location with where the login was initiated. Besides, the precision of latitude and longitude is reduced to a grid-based level to ensure that small variations in location data do not cause verification failures. After verifying the integrity of the location data, the system uses haversine formula to determine if the user is within an allowable distance from the original login location. If all checks pass, a JWT token is generated for users to access the website's protected routes, and the login process is completed. A verified user can then perform API key management in their dashboard, offering a interface for them to adapt the multi-factor authentication in their system easily.

The system integrates three main components — frontend interface, backend API, and database — along with a necessary service: HTML5 Geolocation API. The development is inspired by agile software methodology to build a public API multi-factor authentication service in iterative cycles, modularly, securely. Internal testing is conducted after each cycle to ensure the system stability and security.

3.1 System Use Case

There are several use cases in the system, including registration, verification of email, login, verification of location, domain verification and lastly, logging.

There are two main actors supported: visitor and registered user. A visitor is an unregistered individual who interacts with the API request interface to register for an account. Visitor are required to provide their email, and password. The password is hashed before being stored using Bcrypt library for data confidentiality. A unique "urlKey" is generated for each user and appended to the link sent to their email. Users can only verify their email using the URL given, which will direct users to a frontend verification page. Additionally, if the user does not validate their account within 10 minutes, the registration is automatically invalidated and pending user record is deleted from database.

A registered user may initiate a login request through the system. Required data are email, password and location. The system first verifies the user's credentials and checks the presence of valid location data. If the credentials are correct, a “urlKey” is generated and appended to the link sent to the user's email for location verification. The location data is also used to generate HMAC and stored in database for later verification. The user must open this URL on the same or nearby device to verify their location.

Once the users open the URL, the URL will be invalidated and no longer accessible. Afterwards, the system will match the HMAC of the location where user verify their location with the stored HMAC and that the provided location is within an acceptable proximity to the stored location. Lastly, the system logs the success or failure of this login attempt for auditing and monitoring purposes.

If the user is verified successfully, they are granted access to the system and can then request an API key. To prevent abuse, the system enforces domain-based restrictions. Website owners have to verify their website ownership through file upload and incoming requests are validated via the Origin or Referrer headers.

The use case diagram below summarizes the system’s users and their interactions with it. It illustrates overview of the relationship between use cases, actors, and system. The two actors in the system, Visitor and Registered User, are represented by a stick figures. The oriented ovals represent the actions or services the system provides to fulfil user goals. The line between actors and use cases indicates which actors participate or involve with the use case.

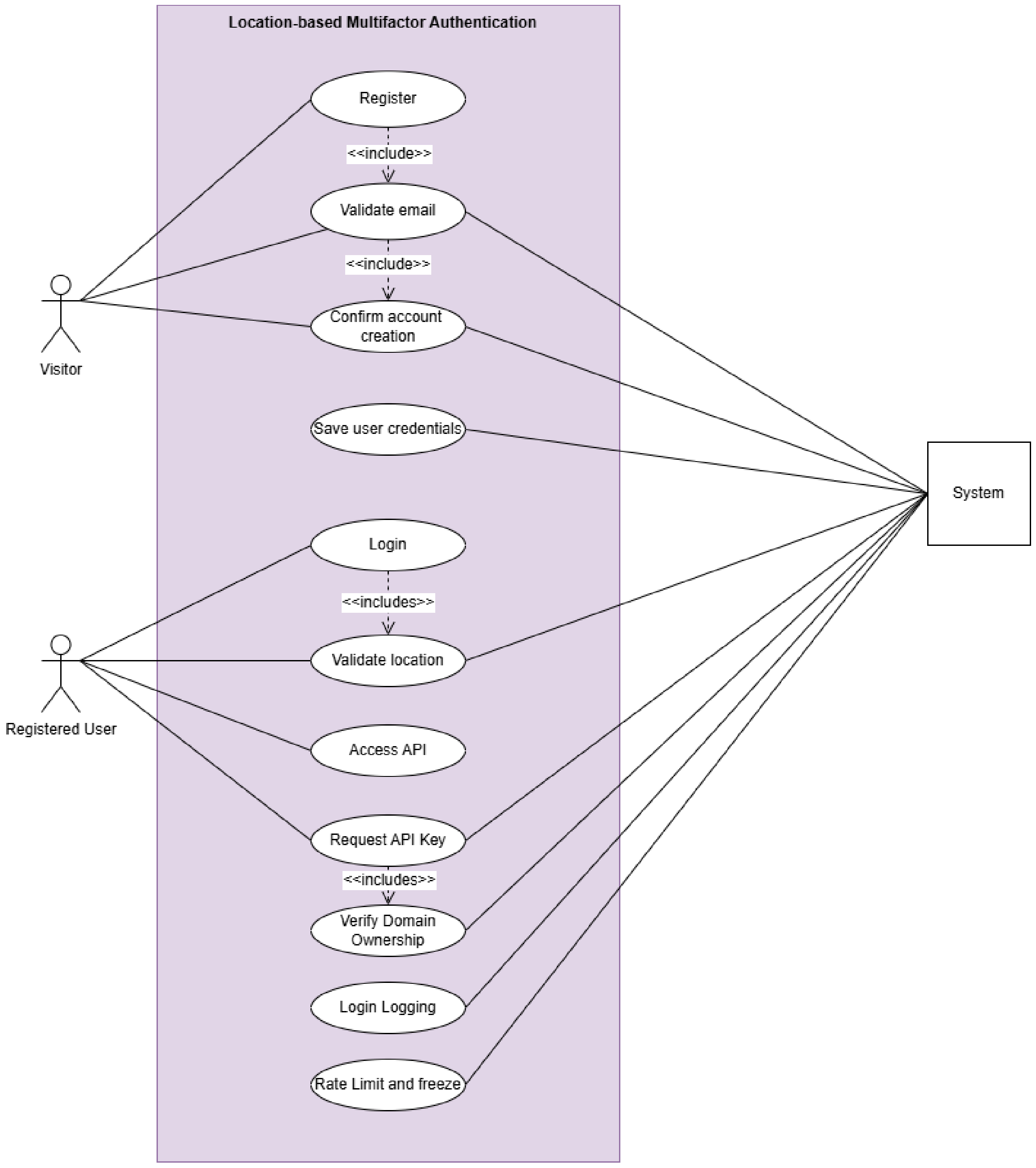


Figure 3.1.1 System Use Case Diagram

3.2 System Architecture

The Location-Based MFA System's key components include the frontend interface, backend API, geolocation service, and database. The following diagram provides an overview of the system's architecture. Each of the components will be further described in detail in section 3.2.1.

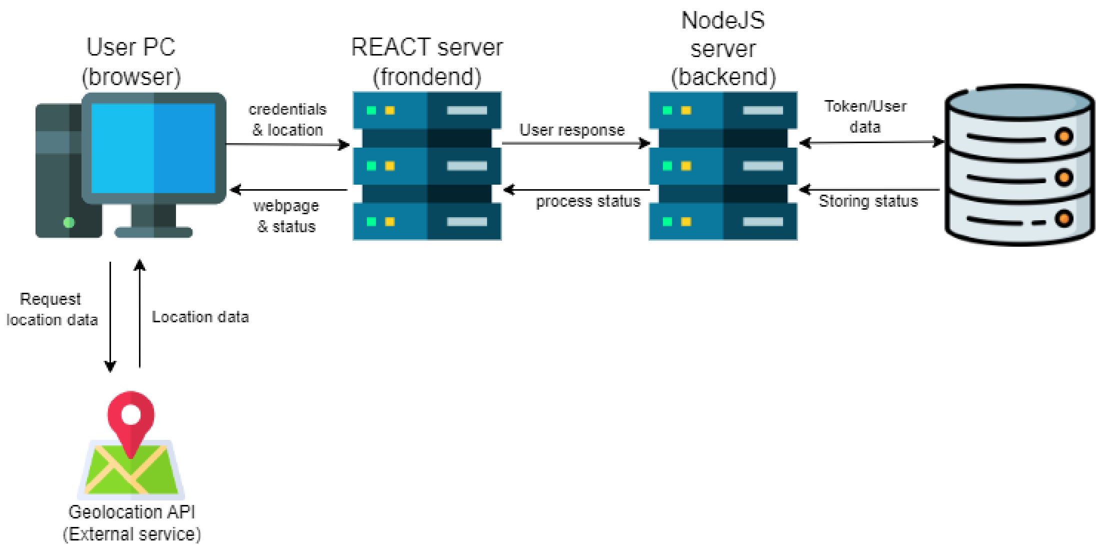


Figure 3.2.1 The diagram of MFA authentication system architecture

3.2.1 System Components

First, the browser is where webpages are rendered for users to enter their credentials, show progress of authentication and allow access to their location data during the login process. This interface collects the necessary information and transmits it to the frontend, which subsequently interacts with the backend for additional processing. The Backend API is vital in handling requests. It authenticates Gmail accounts, examines geolocation data, generates and verifies secure HMAC codes to confirm data integrity, and checks whether the user's current location aligns with the expected parameters. Additionally, it interfaces with other services, particularly the database.

The HTML Geolocation Service is essential for determining whether the user's present location falls within an acceptable proximity to the anticipated location. It is invoked via a call to 'navigator.geolocation', which prompts the user's browser to seek permission for accessing their location data. Upon receiving permission, the browser employs the most precise method available on the device, such as GPS, to obtain this information. Conversely, if permission is refused, the verification process cannot continue.

Finally, the Database stores all important information, including domain and API keys, user credentials, HMAC values, and verification codes. It also keeps records of login attempts for security and audit purposes. The backend API interacts with the database to manage and update user data as part of the authentication process.

3.3 System Development Methodologies

The system's fundamental functions include user registration, email verification, domain verification, location-based login verification, and API key management. All were handled by separate but connected modules. Several system development concepts are applied to develop an authentication system that handles sensitive user data and access controls securely.

3.3.1 Modular Development

The system was relied on several independent but interconnected modules, each responsible for a core functionality. Isolate development of each module ensured that each part could be implemented, tested, and secured separately,

3.3.2 Agile and Iterative Development

Inspired by Agile practices, development was done in short iterative cycles. Every cycle concentrated on developing features into place or improving existing ones, such secure token creation and validation, WebSocket-based verification, location distance computation, or CORS enforcement. After every iteration, internal testing was done to evaluate security, performance, and accuracy. Conducting the evaluation iteratively helped in gradually refining the system while minimizing the risk of breaking system's logic features after updates.

3.3.3 Emphasis on Security

Security is embedded into every phase of development. This involved implementing multifactor verification, short-lived session and token handling, domain verification prior to issuing API keys, rate-limiting, logging and monitoring, domain verification, and CORS validation. In accordance with contemporary secure development lifecycle (SDL) methodologies, every security element was handled as a fundamental necessity rather than an afterthought.

In conclusion, the development process delivered a safe, scalable, and reliable multifactor authentication system that can be integrated by various third-party domains by combining modular design, agile principles, and security-first thinking.

3.4 Development Specification

In this section, the hardware and software specifications for system development and the minimum system requirement for application execution are listed. Besides, the technology involved for the project development are discussed in terms of the reason for selection.

3.4.1 Hardware Specification

The hardware involved in this project includes a computer and a mobile device with browser and email service support. The computer is used for development, while the mobile device is used for testing the application's functionality, particularly the location-based services and email verification process. The table below shows the hardware specifications during the development phase.

Components	Requirements
Computer System	Windows 11

RAM	16GB
Processor	AMD Ryzen 7 5800H with Radeon Graphics
Storage	128GB Samsung MZVL2512HCJQ-00BOO

Table 3.4.1 The computer specifications for development.

For the mobile device, it can be any device that has GPS functionality and browser that support HTML5 geolocation API installed.

3.4.2 Software Specification

In this project, a variety of software tools and platforms were utilized to ensure efficient development, testing, and deployment of the application. The primary components are detailed below.

The project is built on the Node.js runtime environment, which provides a robust platform for developing fast and scalable server-side applications using JavaScript. Several essential Node.js modules are used, including jsonwebtoken for handling JWT, crypto for creating HMACs, geolib for handling geographical calculations, mongoose for object data modeling with MongoDB, cors to prevent cross-site errors, axios for making API calls, nodemon for restarting the server when file changes, and nodemailer for sending emails. These modules, along with the Express.js framework, form the backbone of the project's backend.

React.js is used for building the frontend of the application, providing a powerful and flexible library for creating interactive user interfaces. The development environment is further enhanced by using Visual Studio Code, a lightweight yet powerful Integrated Development Environment (IDE) that offers excellent support for JavaScript, Node.js debugging, and React.js development.

A web browser is essential for accessing online resources, debugging using Browser Developer Tools, viewing the frontend, and testing the integration of the front and backend. The browser's developer tools are particularly useful in debugging and optimizing website development. Besides, tools like Postman and cURL are also utilized in debugging and testing of the system. Commonly used browsers in this project include Google Chrome, Microsoft Edge, and Mozilla Firefox.

Version control is managed using Git and GitHub. Git is used to track changes in the project, ensuring a clear history of development. GitHub also serves as a platform for deploying the code to cloud services like Vercel, streamlining the process from local development to production.

The project utilizes MongoDB Atlas as a cloud-based database, chosen for its ease of integration and scalability. MongoDB Atlas allows for seamless access to the database over the network and simplifies the implementation of API calls. The `mongoose` module is employed for object data modeling, further enhancing the interaction between the Node.js backend and the MongoDB database.

For development and deployment, Vercel and Render are used. They are both cloud hosting platforms that help global availability of a web application. Vercel is particularly effective for deploying frontend applications and static content. Render, while slightly more costly, offers more backend support and WebSocket compatibility. Both platforms provide seamless integration with MongoDB Atlas, easy rollback, HTTPS connections, analysis and code deployment from GitHub's repository, ease the hassle of web application development.

Other tools used includes cURL, HTML 5 Geolocation API, and Gmail. cURL is a command-line tool, used to make API calls, testing connections and ensure APIs are functioning correctly. HTML5 Geolocation API is used to obtain the user's location. Gmail was used for sending verification codes and notifications in the project due to its popularity. However, any email client is applicable.

Components	**Requirements**
Integrated Development Environment (IDE)	Visual Studio Code
Version control	Git and Github
Web browser	Google Chrome, Edge, Firefox

Debugging tools	Postman, cURL, Browser developer tools
Backend runtime	Node.js (v18.18.0)
Web framework	Express.js (^v4.18.2)
Frontend library	React (v18.3.1)
Database	MongoDB Atlas (with mongoose ^v6.10.1)
Deployment Platform	Vercel, Render
Email Service	Gmail
Package management	npm (v9.8.1)

Table 3.4.2 The software required for development.

3.5 TECHNOLOGIES/SERVICES INVOLVED

3.5.1 HTML Geolocation API

HTML Geolocation API is used to obtain reasonably accurate location data. Its location determination methods include GPS, Wi-Fi positioning, cell tower triangulation and IP geolocation. For personal computer, which typically lacks built-in GPS module, the Browser Geolocation API primarily relies on Wi-Fi positioning and IP geolocation for location determination.

3.5.2 Node.js

Node.js is a cross-platform, open-source JavaScript runtime environment, providing developers with benefit of "JavaScript everywhere", allowing developers to use JavaScript for both client-side and server-side development. Therefore, developers can focus on JavaScript for development, eliminating the need to switch between different programming languages and ecosystems [9]. Additionally, Node.js offers powerful concurrency handling capabilities. This is beneficial for the project which involving multiple tasks such as credential verification, session validation, and database access. Its non-blocking I/O model allows tasks to execute concurrently without waiting for I/O operations to complete, enhancing the efficiency and responsiveness of authentication processes when in comparison to PHP and Python-web [9], [10]. npm is used to manage project dependencies and libraries of NodeJS. It simplifies version control, updates, and script automation during development.

3.5.3 Express.js

Express.js is a lightweight framework for Node.js, simplifies the development of web applications [11]. It simplifies the process of defining routes and provides middleware for handling HTTP requests. For example, it intercepts incoming requests and executes functions such as converting request bodies from URLs to JSON. Furthermore, the routing middleware it provides makes the request management modular and efficient, developers can manage the endpoints effortlessly.

3.5.4 React

React’s component-based architecture enables the creation of reusable UI components, which enhances modularity and simplifies maintenance. Its extensive ecosystem, boosted by a vast of libraries, tools, and strong community support, further amplifies its versatility and utility. Furthermore, React’s virtual DOM optimizes performance by only updating the parts of the user interface that have changed, rather than the entire page. This efficiency is particularly advantageous for dynamic applications where frequent updates to the UI are required.

3.5.5 Socket.io

Socket.io is a WebSocket library used for enabling real-time, bidirectional communication between the client and server. It plays a critical role in the login verification process, allowing the frontend to receive asynchronous event updates emitted by the backend for location verification result. The frontend listens for the event in real-time, react immediately upon receiving the verification status to minimize user effort and delay during the authentication process.

3.5.6 MongoDB Atlas

MongoDB's seamless integration with Vercel and other tech stacks of the project also simplifies development and deployment. MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called Binary JSON (BSON). Unlike traditional relational databases, MongoDB allows for dynamic schema design, documents can be stored with varying structures. This flexibility makes it ideal for applications that require scalability, performance, and ease of integration with modern technologies like Node.js. The schema can be designed to automatically expire tokens after a certain period using MongoDB's TTL (Time-To-Live) indexes, which automatically remove expired documents, enhancing security and reducing storage overhead.

3.5.7 JSON Web Tokens (JWT)

In the project, after a user successfully logs in, a JWT is generated and used as proof of the user's identity for subsequent requests. This token is commonly stored in the client's local storage or cookies and is included in the headers of future HTTP requests, allowing the user to remain authenticated without needing to log in again. Additionally, the token is refreshed before it expires using a refresh token. [13], [14].

3.5.8 Hash-based Message Authentication Code (HMAC)

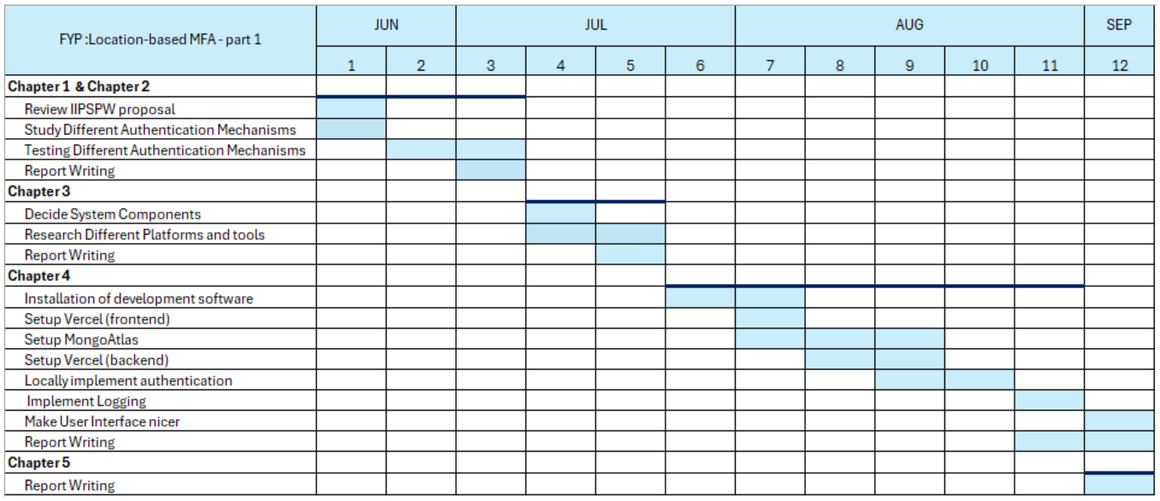
HMAC is a cryptographic technique used to ensure the integrity and authenticity of a message. It combines a cryptographic hash function (like SHA-256) with a secret key to produce a unique hash value for each message. The resulting HMAC is sent along with the message, and the server can use the same key and hash function to verify that the message hasn't been tampered with. If the calculated HMAC matches the one received in the login phase, the message is considered authentic.

3.5.9 Google Maps JavaScript API & Google Geocoding API

The system uses vis.gl/react-google-maps to visually display the expected and current user location when location mismatch occurs during login. The library simplifies Google Maps integration in React by offering map components and hooks. The Geocoding API is used to convert geographic coordinates into human-readable addresses. Maps provide real-time world maps in a simple, intuitive manner. They teach about the world by displaying the sizes and shapes of countries, points of interest, and distances between countries [15]. Rather than simply showing coordinates in number, both are used to provide more effective, visual feedback to users.

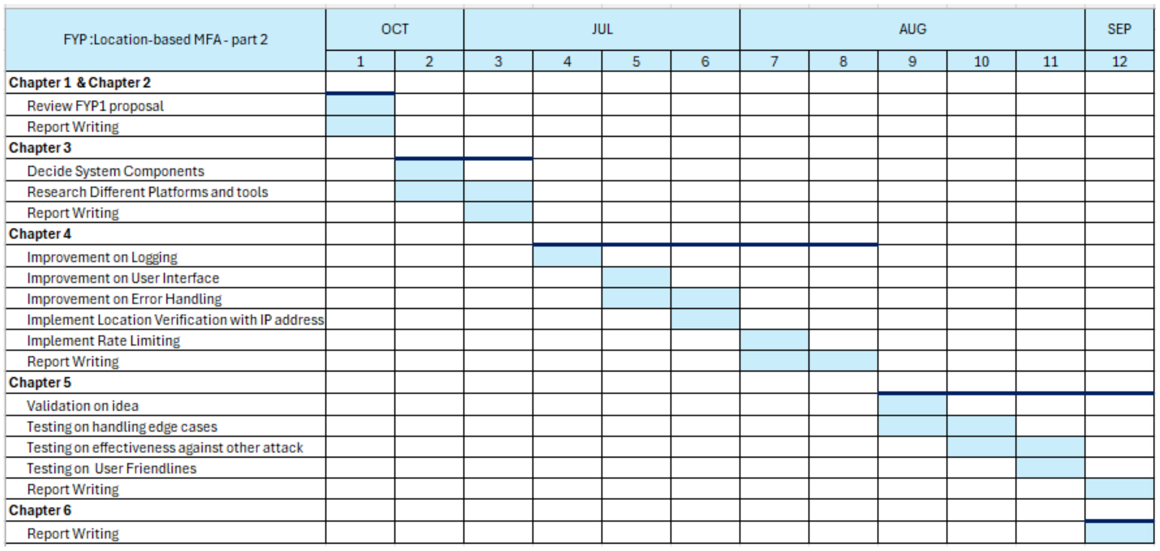
3.3 Timeline

A Gantt chart is provided as below to guide the project onwards. For project 1, it is important to ensure the practicality of the solution, hence the implementation phase consumes significantly more time than others. However, for project 2, system testing is equally important as enhancing functionalities, so both phases will consume pretty much the same time resources.



FYP :Location-based MFA - part 1	JUN		JUL			AUG						SEP
	1	2	3	4	5	6	7	8	9	10	11	12
Chapter 1 & Chapter 2	▬	▬	▬									
Review IIPSPW proposal	🟦											
Study Different Authentication Mechanisms	🟦											
Testing Different Authentication Mechanisms		🟦	🟦									
Report Writing			🟦									
Chapter 3				▬	▬							
Decide System Components				🟦								
Research Different Platforms and tools				🟦	🟦							
Report Writing					🟦							
Chapter 4						▬	▬	▬	▬	▬	▬	
Installation of development software						🟦	🟦					
Setup Vercel (frontend)							🟦					
Setup MongoAtlas							🟦	🟦	🟦			
Setup Vercel (backend)								🟦	🟦			
Locally implement authentication									🟦	🟦		
Implement Logging											🟦	
Make User Interface nicer												🟦
Report Writing											🟦	🟦
Chapter 5												▬
Report Writing												🟦

Table 3.5.1 Project 1 timeline



FYP :Location-based MFA - part 2	OCT		JUL				AUG					SEP
	1	2	3	4	5	6	7	8	9	10	11	12
Chapter 1 & Chapter 2												
Review FYP1 proposal	▨											
Report Writing	▨											
Chapter 3												
Decide System Components		▨										
Research Different Platforms and tools		▨	▨									
Report Writing			▨									
Chapter 4												
Improvement on Logging				▨								
Improvement on User Interface					▨							
Improvement on Error Handling					▨	▨						
Implement Location Verification with IP address						▨						
Implement Rate Limiting							▨					
Report Writing							▨	▨				
Chapter 5												
Validation on idea									▨			
Testing on handling edge cases									▨	▨		
Testing on effectiveness against other attack										▨	▨	
Testing on User Friendlines											▨	
Report Writing												▨
Chapter 6												
Report Writing												▨

Table 3.5.2 Project 2 timeline.

CHAPTER 4

SYSTEM APPLICATION

4.1 Functionalities

4.1.1 Registration

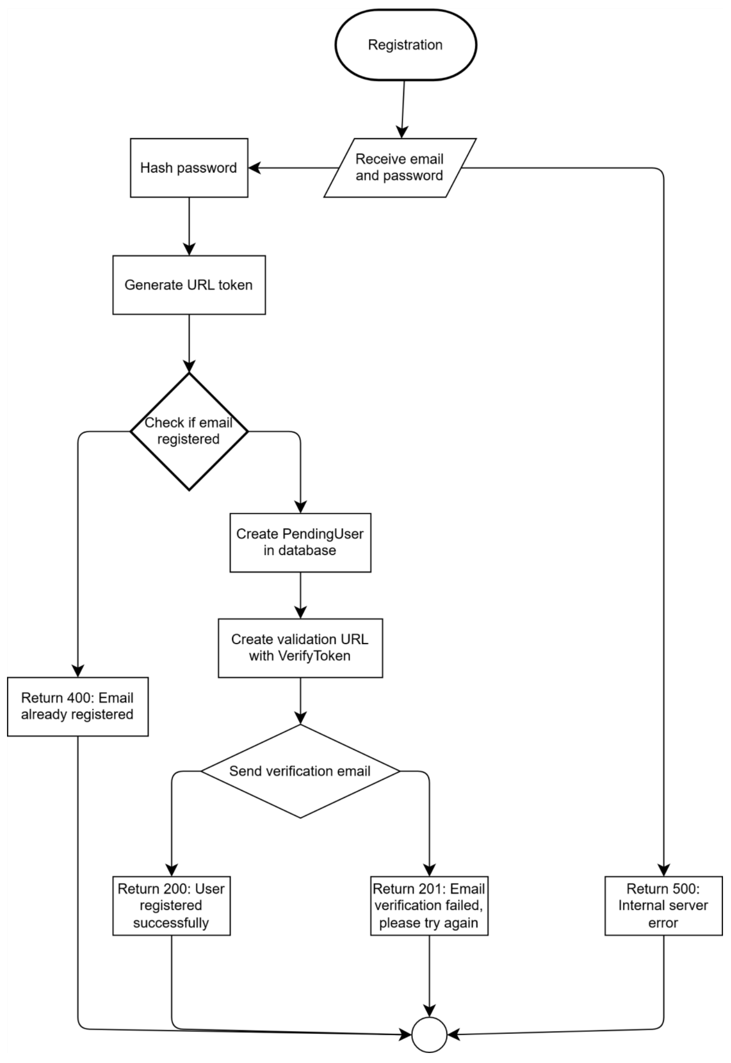

Registration
Receive email and password
Hash password
Generate URL token
Check if email registered
Create PendingUser in database
Create validation URL with VerifyToken
Return 400: Email already registered
Send verification email
Return 200: User registered successfully
Return 201: Email verification failed, please try again
Return 500: Internal server error

Figure 4.1.1 Registration Flow Chart

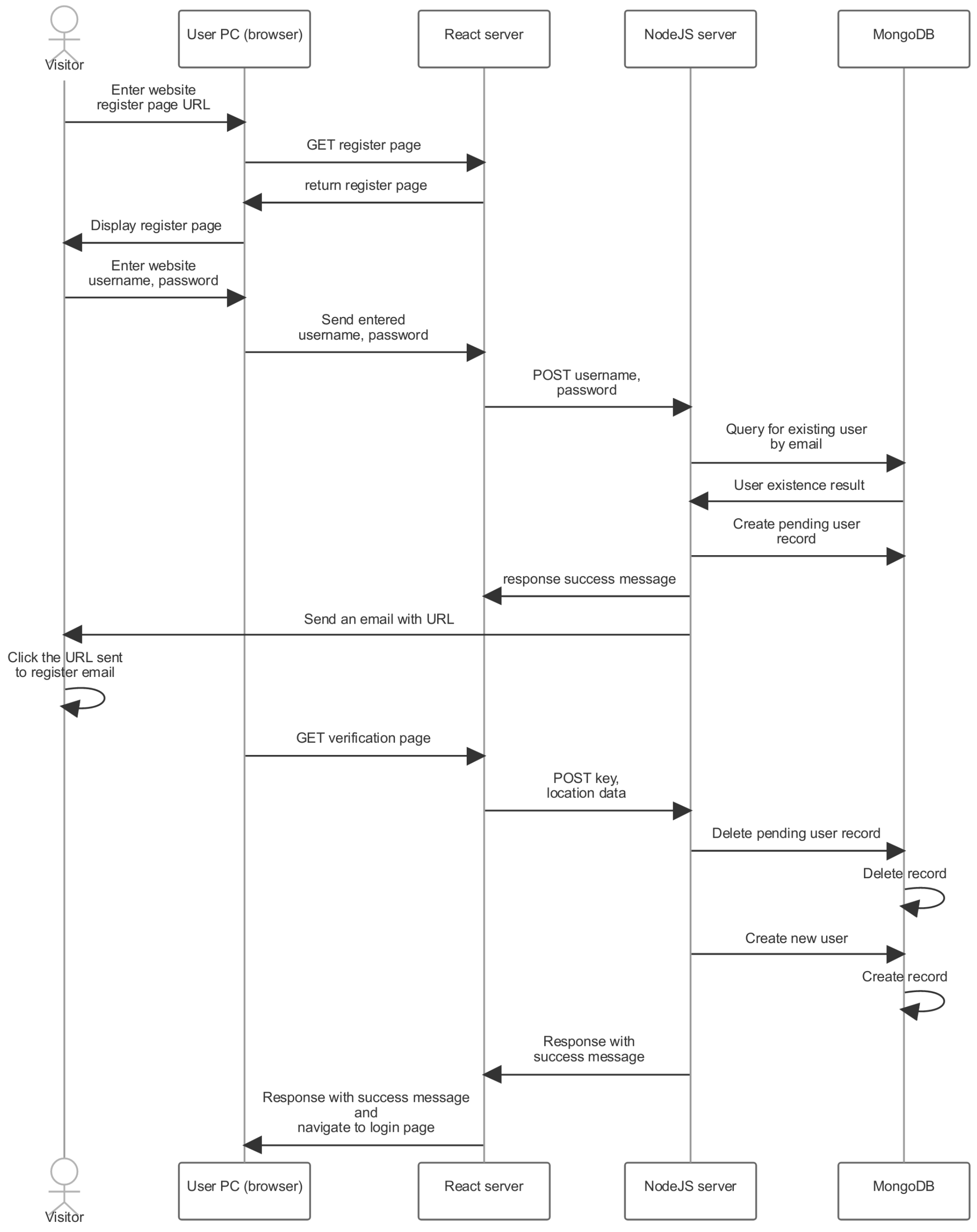

Visitor
User PC (browser)
React server
NodeJS server
MongoDB
Enter website register page URL
GET register page
return register page
Display register page
Enter website username, password
Send entered username, password
POST username, password
Query for existing user by email
User existence result
Create pending user record
response success message
Send an email with URL
Click the URL sent to register email
GET verification page
POST key, location data
Delete pending user record
Delete record
Create new user
Create record
Response with success message
Response with success message and navigate to login page

Figure 4.1.2 Registration sequence diagram

The journey begins when the user types the website profile URL into his or her browser. This initiates a GET request to the frontend React server. The React server responds with the registration page. The user's browser then parses the HTML and displays the content, showing the registration form to the user.

After the registration page appears, the user types in their email, password. Upon submission, React server check for the presence and validity of the data before sending them to the server. This is done to improve user experience, eliminate unnecessary server requests.

Then the React server makes the POST request containing the user's credentials to the NodeJS server for backend operations. The backend, initially, checks if the email already exists in the User collection. If so, it returns a 400 error. If not, it creates a new PendingUser document with a randomly generated username, hashed password, and an extra unique URL key for verification. At the same time, it also sends an email confirmation to the user's email address containing a link that has the uniquely generated URL key.

When the user clicks on the email verification link, an open frontend verification page with the key as query parameter. The frontend makes a GET request to '/api/auth/verifyEmail' with this URL key. The backend finds and deletes the PendingUser record with that key, adds a new User record with the same data, and returns a success response. When the React server receives the successful registration status, it returns the status to the user's browser and the browser displays the registration result to the user, informing the user of the success of the registration attempt. If the user does not click the link within 15 minutes, the PendingUser record will expire automatically and registration will fail.

Once the registration status is shown, the frontend redirects the user to the login page. When the page is received, the browser processes the HTML and shows the login interface to the user.

4.1.2 Login

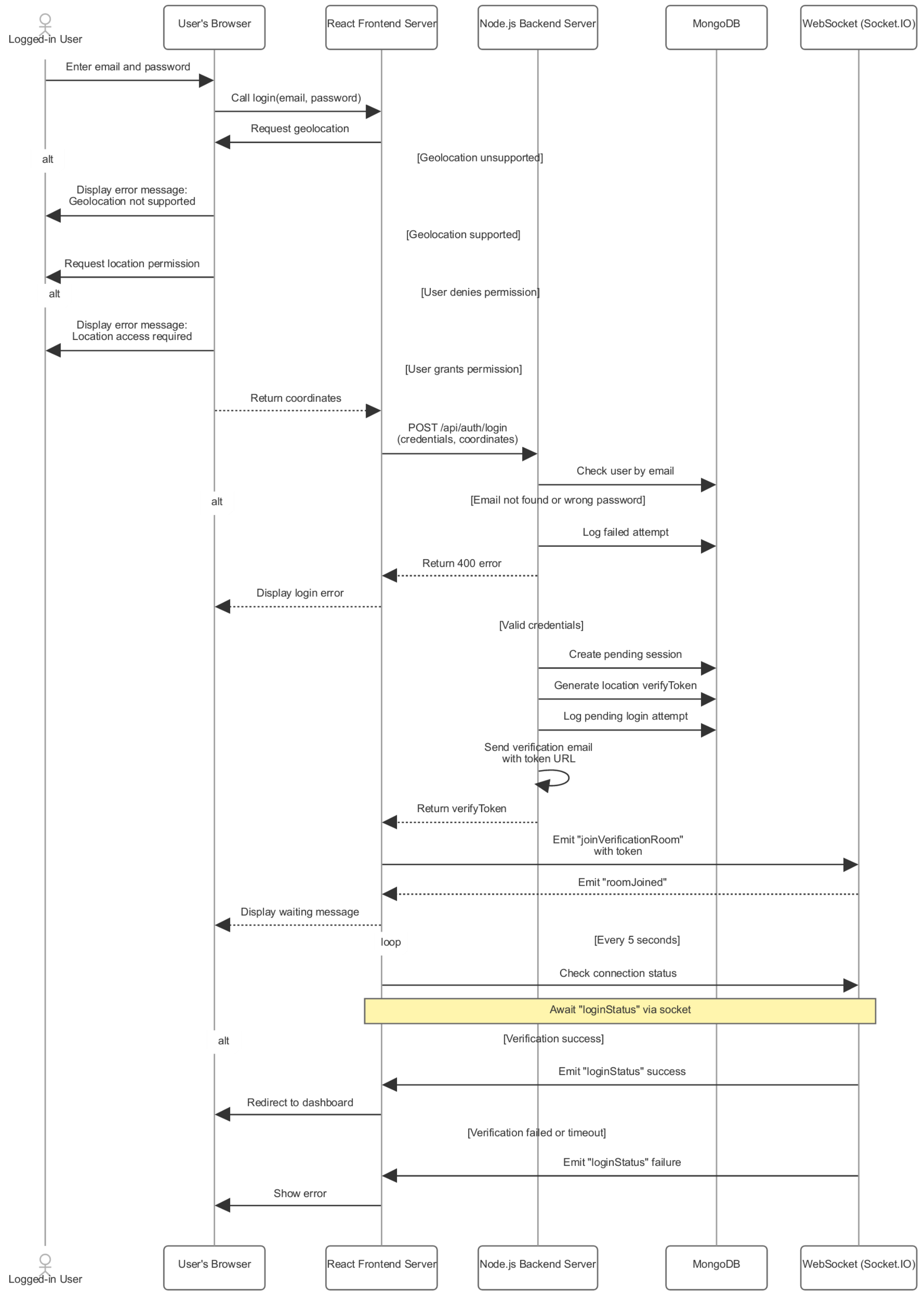


Figure 4.1.3 Login sequence diagram.

This sequence diagram illustrates the user login process, which is the same as registration process from beginning until user input their username and password. Once the user submits this information, the browser invokes the navigator.geolocation API to request the user's current location. If geolocation is not supported by the browser, or the user denies permission, the frontend immediately halts the login process and displays an error message.

If the user grant location access, the browser extracts their coordinates, package it with the email and password, and forward it to frontend, which then send a POST request to the backend API endpoint /api/auth/login.

Upon receiving the request, Node.js backend initiates authentication. It matches the given credentials against the records of User collection. If the user does not exist or the password is incorrect, login failure is recorded in LoginAttempts collection and error 400 is returned.

However, if the credentials are valid and geolocation data is present, the backend generates a verification token, stores it in VerifyToken collection and sends it to user's email for them to click and verify their location. The token is tied to the session and the user's location, as a result, the backend can query the login entry back.

Simultaneously, the backend returns the token to frontend for the frontend to establish a WebSocket connection. Frontend will fire a joinVerificationRoom event with the token to subscribe to the appropriate verification channel. Once joined, the frontend shows a "Waiting for verification" message and establishes a 5-second interval to monitor the WebSocket connection status. A 10-minute timeout is also established in frontend so that it will not wait endlessly.

The user is required to check their email and click the verification link to trigger a location-based verification and proceed with their login. The detailed flow for location verification is depicted in 4.1.3. If it is confirmed, the backend updates user status, emits a loginStatus event of success flag and a JWT token via the WebSocket channel. When this event is received, the frontend cancels the interval and timeout, stores the token, and redirects the user to the dashboard. Upon verification failure or time out, the user will be notified through an error message.

4.1.3 Verify Location

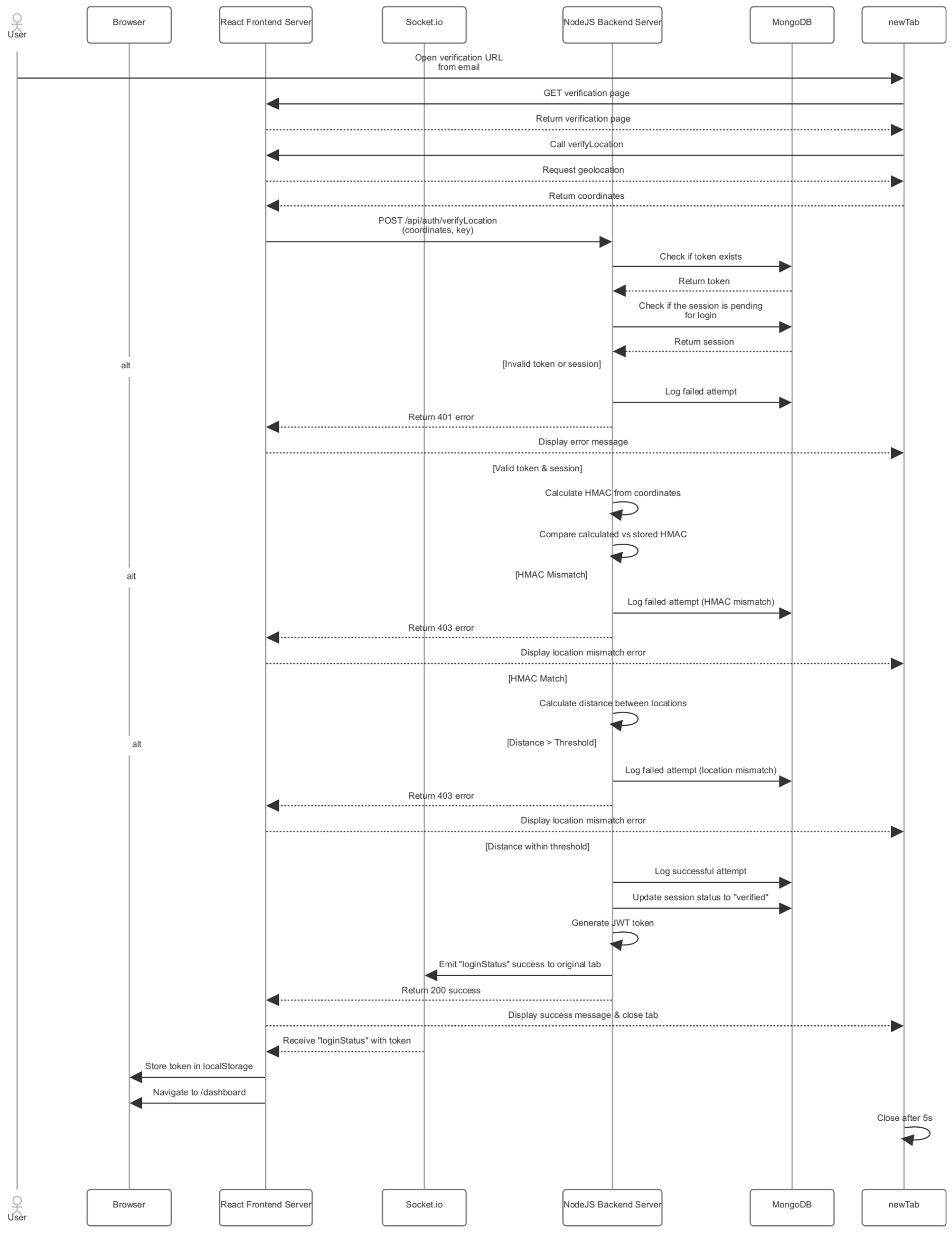

User
Browser
React Frontend Server
Socket.io
NodeJS Backend Server
MongoDB
newTab
Open verification URL from email
GET verification page
Return verification page
Call verifyLocation
Request geolocation
Return coordinates
POST /api/auth/verifyLocation (coordinates, key)
Check if token exists
Return token
Check if the session is pending for login
Return session
alt
[Invalid token or session]
Log failed attempt
Return 401 error
Display error message
[Valid token & session]
Calculate HMAC from coordinates
Compare calculated vs stored HMAC
alt
[HMAC Mismatch]
Log failed attempt (HMAC mismatch)
Return 403 error
Display location mismatch error
[HMAC Match]
Calculate distance between locations
alt
[Distance > Threshold]
Log failed attempt (location mismatch)
Return 403 error
Display location mismatch error
[Distance within threshold]
Log successful attempt
Update session status to "verified"
Generate JWT token
Emit "loginStatus" success to original tab
Return 200 success
Display success message & close tab
Receive "loginStatus" with token
Store token in localStorage
Navigate to /dashboard
Close after 5s

Figure 4.1.4 Location verification sequence diagram

The location verification process takes place in a separate browser tab or window, which the user initiates by clicking the verification link sent to their email.

When the user loads the verification link, the frontend will request permission to the user's location, and, if the user consented to share his or her location, then needed coordinates (either GPS or IP coordinates) were recorded from the user. If the two conditions were satisfied, the front-end constructs a POST request sending the user's location coordinates and verification key to the backend endpoint /api/auth/verifyLocation.

On the back-end the server receives the verification token and the user's location. The server will lookup the token from the database, where it should also be associated with a session pending for location verification. The location verification can only continue when the given token exists in the database and the session linked to the token is pending for location verification. If those were not satisfied, the verification will be considered invalid, and the server will respond with a 401 error.

If both of those were satisfied, then the backend will now verify the user's location. Initially, the backend will create a HMAC (although the standard HMAC will have the user's actual coordinates). Using the coordinates originally captured at the time he or she was verifying his or her location, it will then compare that to what was generated and saved at the time of login for that session. If the HMACs do not match, the request is immediately rejected.

Even in the case of the HMACs being equal, the server takes it a step further by verifying the physical distance between the login location of the user and the verification location. If the distance exceeds the threshold 2000m, the verification has failed due to either a spoofing or an unauthorized login attempt. Every failure, whether due to an invalid session, mismatched HMAC, or distance violation, is logged in the database along with the reason for failure, ensuring transparency and auditability.

If the verification is successful, the system updates the login session’s status to “verified.” At this point, the backend generates a new JWT token for the authenticated user and emits a loginStatus event via the established WebSocket connection. This event is directed to the original login page, still open in the user's browser.

Meanwhile, the verification page displays a success message and automatically closes after a short delay, typically around three seconds, providing a seamless user experience. Back on the original login page, the frontend receives the loginStatus event containing the JWT token. This token is stored in the browser’s local storage, and the user is automatically redirected to the dashboard, the login process is completed.

4.1.4 Rate Limiter

Your system implements a two-layer defense mechanism to prevent brute-force login attacks. The first layer uses IP-based rate limiting, restricting each IP address to a maximum of five login attempts within a 15-minute window. If this threshold is exceeded, the system logs the incident, blocks further attempts from the IP temporarily, and optionally increments the failure count for the associated user account (if an email is provided). The second layer involves account-based protection. If a specific user account experiences repeated failed login attempts, the system temporarily locks the account after five failed attempts and notifies the user. This dual-layer strategy helps mitigate both automated brute-force attacks from a single IP and targeted attacks against individual accounts.

4.1.5 CORS, API Key Validation and Domain Verification

Cross-Origin Resource Sharing (CORS) is an HTTP-header-based security mechanism that allows a web page to access resources from a server on a different domain [19]. Traditionally, the frontend and backend are hosted on the same domain, allowing them to communicate without cross-origin restrictions. However, the system is designed to be a public authentication service to ease the adoption of location-based multifactor authentication, reducing cyber threats. Websites will be hosted on different domain with the authentication server, their browsers must perform cross-origin requests to use interact with the authentication system.

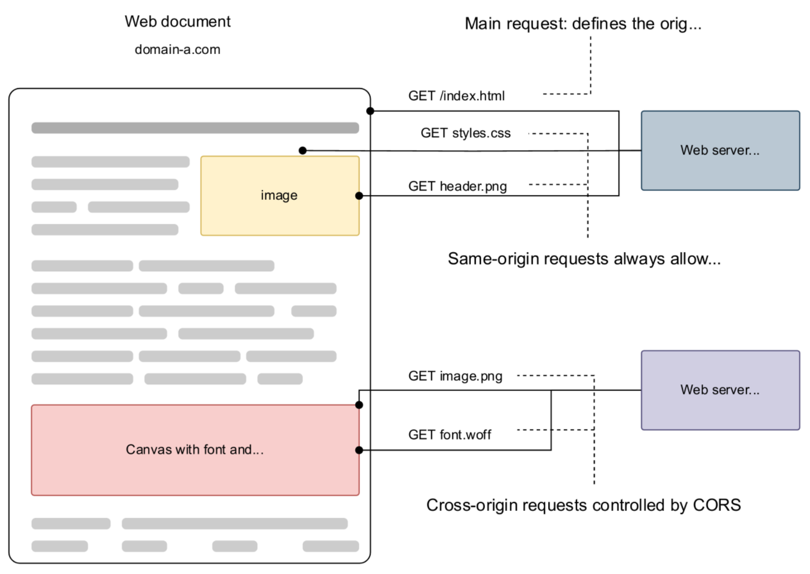


Figure 4.1.5 The CORS mechanism[19].

4.1.5.1 Cross-Origin Resource Sharing (CORS) Enforcement

The system has a middleware restricting domain access to API. Upon receiving a request, it retrieves approved domains from the database, checks against the Origin header of the request and only sets Access-Control-Allow-Origin header to the approved domain. Other unauthorized domains receive a CORS error and are blocked before reaching the API logic.

4.1.5.2 API Key Validation:

To be qualified as a trusted domain, domains must obtain API key and perform domain verification from backend. The API key is then included in each request, multifactor-auth-api-key in the request headers to request to protected API endpoints. The backend server then checks the API key to validate the consumer's identity before returning the requested data. If the API key is missing, invalid, or associated with an inactive website, the request is rejected.

API keys also play a crucial role in rate limiting on the system, which is the practice of controlling the number of requests made to an API by a specific client in a certain period of time. Rate limiting helps prevent resource exhaustion and protects the API from security threats.

By combining strict CORS policies and API key authentication, the system effectively mitigates Cross-Site Request Forgery (CSRF) attacks. Even if a user is tricked into making a request, it would lack the correct API key or originate from an unauthorized domain, and therefore would be rejected.

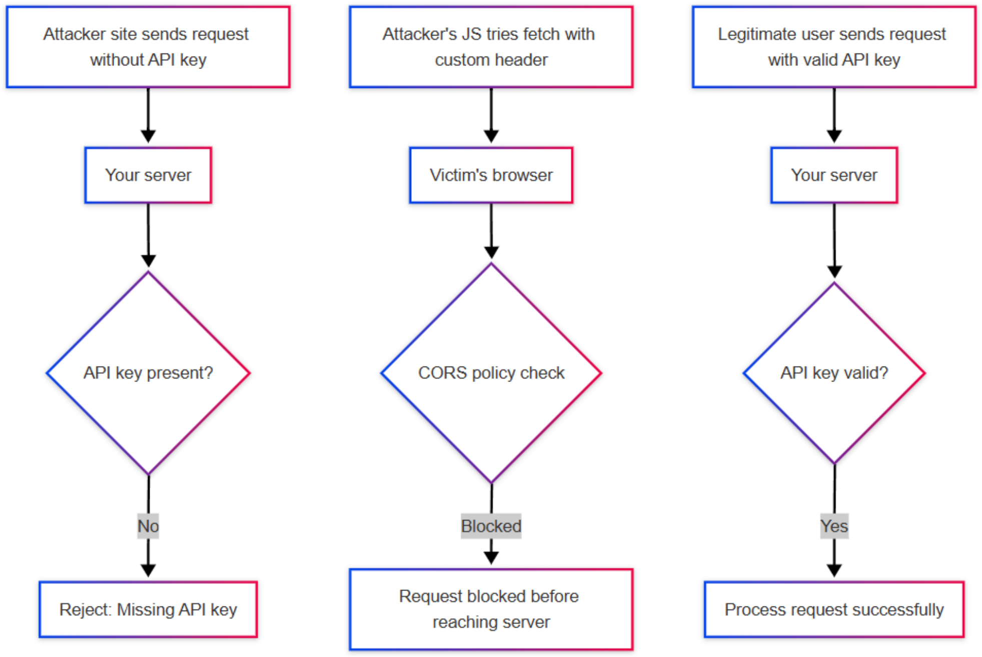

Attacker site sends request without API key
Your server
API key present?
No
Reject: Missing API key
Attacker's JS tries fetch with custom header
Victim's browser
CORS policy check
Blocked
Request blocked before reaching server
Legitimate user sends request with valid API key
Your server
API key valid?
Yes
Process request successfully

Figure 4.1.6 API Checking Protection

4.1.5.3 Domain Verification

The authentication system requires domain owners to verify their ownership of the domain to obtain an API key. Without domain verification, any malicious users could easily register an API key, thereby allowing them to impersonate trustworthy websites or abuse the API. The integrity of the authentication system would be jeopardized by these acts. Therefore, domain verification is implemented for owners requesting API key to demonstrate their control over the domain, reducing the risk of credential theft, API abuse, or unauthorized use of authentication services.

Various verification methods can be used for domain verification. To determine the most effective method, multiple approaches are compared. The evaluation factors include their security levels, ease of implementation, and compatibility with cloud-based hosting platforms including Vercel, GitLab Pages, and Netlify.

Methods	**Security Level**	**Ease of Implementation**	**Compatibility with Cloud**	**Description**
DNS TXT Record Verification	High	Medium	No (Limited)	Secure but requires access to DNS records, which some cloud hosts don't support.
DNS Resolution	Medium	Hard	No (Limited)	Verifies domain ownership but requires DNS control, limiting usability on some cloud hosts.
Pattern Matching	Low	Easy	Yes	Can block suspicious domains but is easily bypassed.
Referer or Origin Header Checking	Low-Medium	Medium	Yes	Useful but can be manipulated (e.g., by CURL requests).
File Upload Verification	High	Medium	Yes	Websites upload a verification file to confirm ownership. Works without a custom domain.
Email Confirmation	Medium	Easy	Yes	Ensures legitimate domain ownership if using a verified email (e.g., admin@domain.com).

Table 4.1.1 Comparison on methods for domain verification

Considering security, ease of use, and compatibility with cloud-hosted services, the system implements the combination of file upload verification and checking

Referrer/Origin headers. This solution provides secure the system moderately while eliminating the need for DNS configuration, allowing websites without custom domains to adopt the system. Finally, the combination supports automated validation of domain through API endpoints.

4.1.6 Dashboard

Dashboard is used for registered users to perform API key management and domain verification. It has strict access control and disable unauthenticated users from accessing it with automatic redirection to login page. When a session is expired, an alert is triggered and the user is redirected to login page upon confirmation.

4.1.7 Verify Token

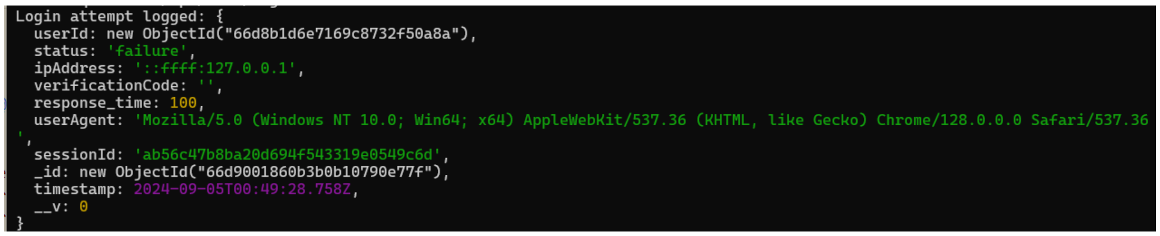
It is important to note that, the verification URL sent to users contains the verifyToken, which is used by the server to identify the user currently performing email or location verification. It is a risky action if a user is lured into exposing the verification URL when the attackers are performing website mirroring or phishing attack. In such cases, if attackers obtain both users credentials and the verification URL, they could easily bypass the location-based multifactor authentication by making the expectedLocation and currentLocation matches.

While binding token verification to a user's IP address can help prevent unauthorized reuse of the link, this approach has a significant drawback: it restricts verification to a single IP. If a legitimate user switches networks or uses a different device, the IP address changes, potentially causing the verification to fail.

Therefore, to balance usability and security, several measures are implemented. Firstly, Mongo TTL index is used to ensure the tokens are short-live (10 minutes). Second, each token can only be used once. Furthermore, alert warning users not to share the verification link is included in email. Lastly, the link is embedded as an HTML button to reduce the risk of copying and pasting, particularly on platforms like Gmail mobile, which typically restricts copying of the link address of a button.

4.1.8 Logging

The logging functionality logs important information such as the user ID, status of the attempt, IP address, response time, session ID, and user agent of each login attempt. The LoginAttempts schema is described in section 4.2.4. Below shown a logging example of failed attempt due to an invalid token provided.



```
Login attempt logged: {
  userId: new ObjectId("66d8b1d6e7169c8732f50a8a"),
  status: 'failure',
  ipAddress: '::ffff:127.0.0.1',
  verificationCode: '',
  response_time: 100,
  userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36
',
  sessionId: 'ab56c47b8ba20d694f543319e0549c6d',
  _id: new ObjectId("66d9001860b3b0b10790e77f"),
  timestamp: 2024-09-05T00:49:28.758Z,
  __v: 0
}
```

Figure 4.1.7 Logging of failure attempt printed on backend server.

4.2 Database Schema

In section 3.2.5, an overview of MongoDB Atlas was provided, and here we will focus on important concepts related to NoSQL databases, MongoDB, and the design of the data schema used in this project.

NoSQL databases, such as MongoDB, are particularly suited for projects where the relationships between stored data are not of primary importance. In scenarios where data is unstructured or changes over time, a rigid schema can become a significant limitation. MongoDB's adaptability to evolving data requirements is crucial during project development, where data structures are continuously refined until the final deliverable. Its capability to efficiently manage large volumes of data at high access speeds is also vital, especially as web applications increasingly store and retrieve growing amounts of information. It also allow cloud-based solution and can be integrated into Vercel in minimal effort.

While MongoDB offers many advantages, it is important to note that it does not fully support complex transactions, constraints, or joins, which are common in traditional relational databases (RDBMS). However, these features are not required for this project, as the relationships between the data are not complex.

4.2.1 User Schema

Three schemas are used in this project, with the first being the user schema. The User schema is a crucial component of the authentication system, responsible for storing essential user data, including the username, email, password, URL key, verification code, last login, and last login time. The username is unique and serves as the primary identifier for locating user records in the database, while the email field is mandatory for communication purposes. The URL key and verification code are dynamically updated with each login attempt, ensuring that the system can effectively track and authenticate users. The URL key helps in identifying users during the email verification and login process when they click the verification link. Additionally, the last login and last login time fields record the timestamp of the user's most recent login, enabling the system to manage user sessions effectively.

4.2.2 PendingUser Schema

This schema stores user registration data before email or link verification. If the user doesn't verify in time, the record will expire and be removed automatically. The purpose of having a PendingUser collection is to prevent the creation of actual user accounts without email verification. This adds a layer of security and protects the system from spam or fake registrations. It ensures that only users who confirm their email address are allowed to own an account.

Field Name	Type	Required	Default Value	Description	Example
username	String	Yes	N/A	Stores the user's unique username.	user001
email	String	Yes	N/A	Stores the user's email.	kuekisabella@gmail.com
Password	String	Yes	N/A	Stores the hashed password for user authentication	1234567890
verificationCode	String	No	N/A	Stores the verification code for email verification and login.	FhQAvB
urlKey	String	No	N/A	Appended behind URL to identify user identity across devices and platform.	393fd25b41c5957d1f1 78ec48666e07163fd7d e5f757e6d82cbaa0ea9 a66ee03f97306860b04 895efcc45c6b158d067 470149d1185f6cd61fb e705af1a389f7e
isValidated	Boolean	No	false	Indicates whether user email is validated.	true
Timestamps	createdAt, updatedAt	No	Date.now	Automatically added fields for record creation and last update timestamps.	createdAt: 2024-09-04T20:20:01.804Z; updatedAt: 2024-09-04T20:20:01.804Z

Table 4.2.1 The Description of User Schema.

Field Name	Type	Required	Default Value	Description	Example Value
_id	ObjectId	Yes	Auto-generated	Unique identifier for the session record	68089d1c696f2a5163930207
sessionToken	String	Yes	N/A	Unique token identifying the session	4e9642b6b18d6c243d0bb19d9ad690f5
userId	ObjectId (ref)	Yes	N/A	References the user who is logging in	67e92f48368752c1d196735a
websiteId	ObjectId (ref)	Yes	N/A	References the website associated with the session	67ce87bb605fc1827b4ec916
status	String (enum)	No	"pending"	Current session state: "pending" or "verified"	verified
verifyTokenId	ObjectId (ref)	No	N/A	Link to the verify token used during login	68089d1c696f2a516393020b
expiresAt	Date	No	15 minutes from creation	When the session expires (automatically deleted after this time)	2025-04-23T08:11:12.075Z
createdAt	Date	No	Auto via timestamps	Session creation timestamp	2025-04-23T07:56:12.079Z
updatedAt	Date	No	Auto via timestamps	Last update timestamp	2025-04-23T07:56:55.165Z

Table 4.2.2 The Description of PendingUser Schema.

4.2.3 VerifyToken schema

VerifyToken schema plays a vital role in securely handling all relevant data during the login verification process. short-lived verification tokens. It stores the URL key, associated geolocation data, and HMAC of the location. The schema also includes a createdAt field, which automatically deletes the record after 10 minutes, MongoDB TTL index to ensure tokens are short-lived and secure. ensuring that sensitive information is not retained longer than necessary. This field, along with standard timestamps, allows the system to track when the token was created, providing a clear record for each login attempt.

4.2.4 LoginAttempt schema

The LoginAttemptsSchema is the schema for logging login attempts. It has various details related to each login attempt, including the user and website where the request is from, the status of the attempt, the IP address, and optional details like the reason of failure. This schema is useful for tracking and analysing login attempts, which can aid in monitoring security and troubleshooting issues related to authentication.

4.2.5 UserSession schema

This schema stores temporary session data when a user is logging in or verifying their identity. It helps keep track of session status and expiry time for security purposes.

4.2.6 PendingDomainSchema

The PendingDomain schema is used to temporarily store information about domains undergoing verification. It is created when a website owner requests for apiKey and submits a domain. Once verified (via token), the domain is moved to the Website collection.

4.2.7 WebsiteDomain

Once a domain is verified, it is stored in the Website schema. This schema contains information about verified domains and their associated API keys.

Field Name	Type	Required	Default Value	Description	Example Value
_id	ObjectId	Yes	Auto-generated	Token identifier auto-generated by MongoDB	ObjectId("68089d1c696f2a516393020b")
verifyToken	String	Yes	N/A	The one-time token appended after verify URL.	"a8f5ebc6a294d59abab0fe72dd13681329377c523f829c838331a4e51a572c88"
locationData	Object	Yes	N/A	Contains the latitude and longitude coordinates associated with the token.	{ latitude: 4.3253646, longitude: 101.1298997}
hmac	String	Yes	N/A	Stores the HMAC hash generated from the location data and URL key.	Binary.createFromBase64('N2VmNWJiMWI2YjIwNmU4MmFkZTg1YTRhZGQwNjdiZWNiNWZmM…")
createdAt	Date	No	Date.now()	Time of creation; auto-expires in 10 minutes	2025-04-23T07:56:12.229Z
updatedAt	Date	No	Auto via timestamps	Last modified time	2025-04-23T07:56:12.229Z

Table 4.2.3 The Description of VerifyToken Schema.

Field Name	Type	Required	Default Value	Description	Example Value
_id	ObjectId	Yes	Auto-generated	LoginAttempt identifier generated by MongoDB	ObjectId("67eb1449666a1dabdc343c64")
userId	ObjectId	No	N/A	ID of the user trying to log in. It isn't a required field as some login attempts fail to find the user.	ObjectId("66d8b1d6e7169c8732f50a8a")
websiteId	ObjectId	Yes	N/A	ID of the website where the login attempt was made.	ObjectId("67ce87bb605fc1827b4ec916")
sessionId	String	No	N/A	ID of session associated with this login attempt.	ObjectId("67eb1449666a1dabdc343c62")
status	String	Yes	N/A	Outcome of the attempt: success, failure, or pending.	"success"
ipAddress	String	Yes	N/A	IP address from which the login attempt was made.	"::ffff:127.0.0.1"
timestamp	Date	No	Date.now()	Time the login attempt occurred.	2025-03-31T22:16:41.410+00:00
responseTime	Number	No	N/A	Time (in ms) it took to process the login request.	14993
userAgent	String	No	N/A	Browser/device details from which the attempt was made.	"Mozilla/5.0 (Windows NT 10.0...)

reasonOfFailure	Enum	No	N/A	Reason the login failed (if any). Enum includes things like timeout, etc.	"other"

Table 4.2.4 The Description of LoginAttempt Schema

.

Field Name	Type	Required	Default Value	Description	Example Value
sessionToken	String	Yes	N/A	Unique token representing the session.	abc123sessiontoken
userId	ObjectId (ref: User)	Yes	N/A	References the user who initiated the session.	609e125d0e9f1b0015e7a9a0
websiteId	ObjectId (ref: Website)	Yes	N/A	References the website associated with the session.	609e12810e9f1b0015e7a9a5
status	String	No	"pending"	Indicates whether the session is pending or verified.	verified
verifyTokenId	ObjectId (ref: VerifyToken)	No	N/A	References the one-time verification token used for location/MFA verification.	60af938c93d45b0027a9fd13
expiresAt	Date	No	15 minutes from creation	TTL field for auto-deleting expired sessions after 15 minutes.	2025-05-05T13:15:00.000Z

Table 4.2.5 The Description of UserSession.

Field Name	Type	Required	Default Value	Description	Example Value
_id	ObjectId	Yes	Auto-generated	Unique ID for the website entry.	ObjectId (“661c9955c3b7e4e8b8c5a33f”)
name	String	Yes	N/A	A human-readable label for the API key. Useful if a user manages multiple keys.	Testing for multifactor
apiKey	String	No	uuidv4()	A unique API key assigned to this verified domain. Indexed for faster lookup.	“085427ec11cbf2af472fb495e417f5fedff2ee39d4553711ac69e10c9cb48abd”
domain	String	Yes	N/A	The verified domain name. Regex validation ensures the domain format is correct.	“https://fyp-frontend-five.vercel.app/”
createdAt	Date	No	Auto via timestamps	Automatically recorded creation date.	2025-04-15T09:00:00.000Z
updatedAt	Date	No	Auto via timestamps	Automatically updated whenever the record is modified.	2025-04-15T10:00:00.000Z

Table 4.2.6 The Description of WebsiteDomain Schema.

Field Name	Type	Required	Default Value	Description	Example Value
_id	ObjectId	Yes	Auto-generated	Pending domain identifier generated by MongoDB	ObjectId("67eb1449666a1dabdc343c64")
userId	ObjectId	Yes	N/A	ID of the user who submitted the domain for verification	ObjectId("66d8b1d6e7169c8732f50a8a")
domain	String	Yes	N/A	Domain submitted for verification	"https://fyp-frontend-five.vercel.app/"
verifyToken	String	Yes	N/A	Token used to verify domain ownership	"a1b2c3d4e5f6g7h8"
createdAt	Date	No	Date.now()	Time the record was created. Entry auto-expires after 24 hours.	2025-04-15T09:00:00.000Z

Table 4.2.7 The Description of PendingDomain.

CHAPTER 5

SYSTEM IMPLEMENTATION

The application is developed based on the system design that was discussed in Chapter 4. The discussion is separated according to each module – Domain verification, Register, Logging and Rate-limiting, Location Verification.

5.1 Register

As illustrated in section 4.1, the Register module allows users to create an account. It has some additional features to secure the registration process. First, password toggle feature prevents shoulder surfing, allowing users to hide or show their password input for better usability and privacy. Besides, Register module enforces a secure password policy as shown in Figure 5.1.1. These requirements are suggested by Kaspersky, a Malaysia cybersecurity company, to protect the system against brute-force attacks and credential stuffing [17].

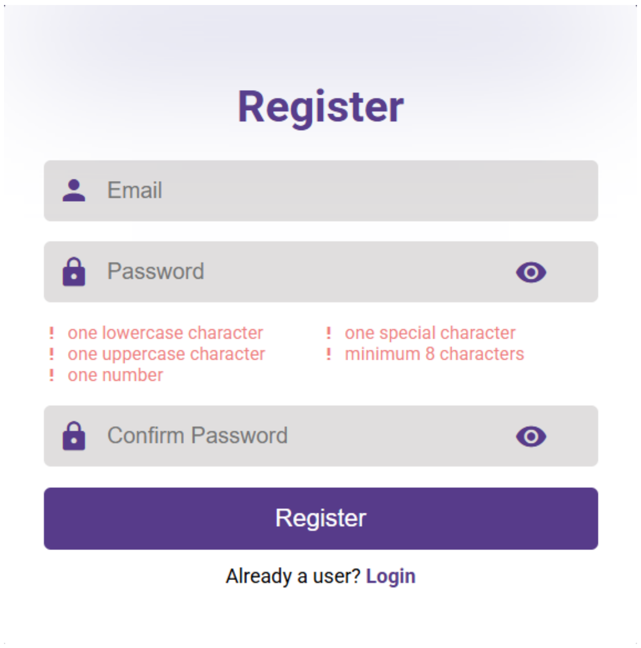
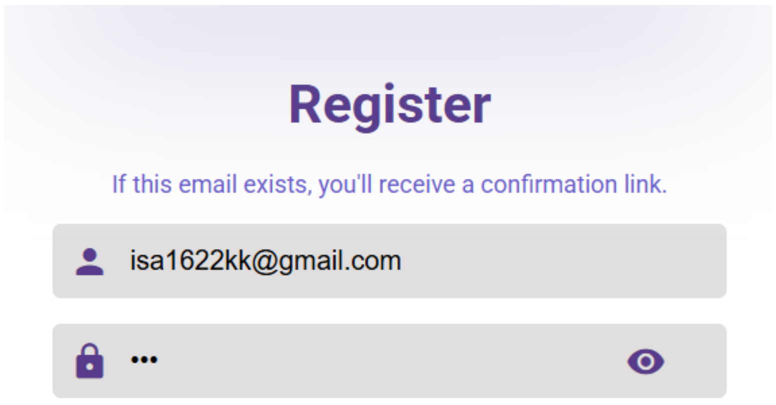


Figure 5.1.1 Registration Interface

The backend response user registration by checking if the provided email is registered. In preliminary implementation, backend will return error and show error message “Email already registered”. However, this informative message may be exploited by malicious actors to perform enumerate attack on valid user emails. To mitigate this, the implementation was updated to return a more generic success message regardless of whether the email is already registered. For Register module, it will always return “If this email exists, you’ll receive a confirmation link.” for error likes exist email or if the system encounters an error while attempting to send the verification email.


Register

If this email exists, you'll receive a confirmation link.

isa1622kk@gmail.com

•••

Figure 5.1.2 Consistent message regardless of registration status.

Unlike the registration response, which is intentionally generic to prevent email enumeration, the system’s response to excessive login attempts is more specific: “Too many login attempts. Try again in 15 minutes.” The reason for this difference lies in the function and intent of rate limiting. Generic message is given as registration forms are a common target for enumeration attacks that seek to confirm the existence of user accounts. Nonetheless, rate limiting primarily serves to mitigate brute-force login attempts. In this context, providing a clearer message about temporary blocking helps inform legitimate users who may have repeatedly entered incorrect credentials and are confused by subsequent login failures. The message does not expose whether the submitted email or password was valid; instead, it indicates that too many attempts — regardless of their correctness — were made in a short period. Importantly, it still prevents enumeration because it is consistent across both valid and invalid emails. This careful balance allows for a user-friendly experience while still upholding the system's security standards.

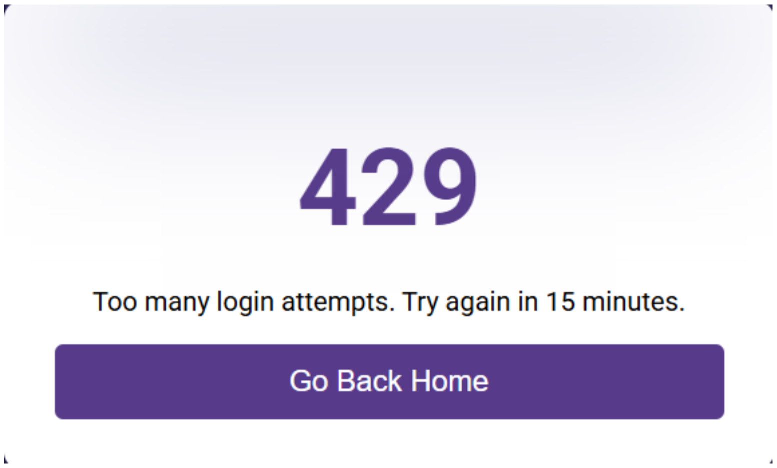
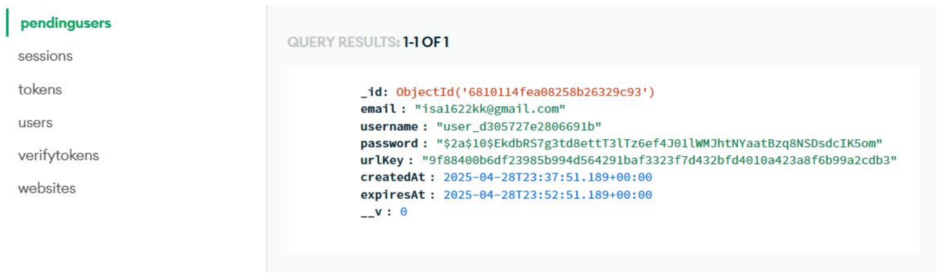


Figure 5.1.3 Error message for excessive registration request.

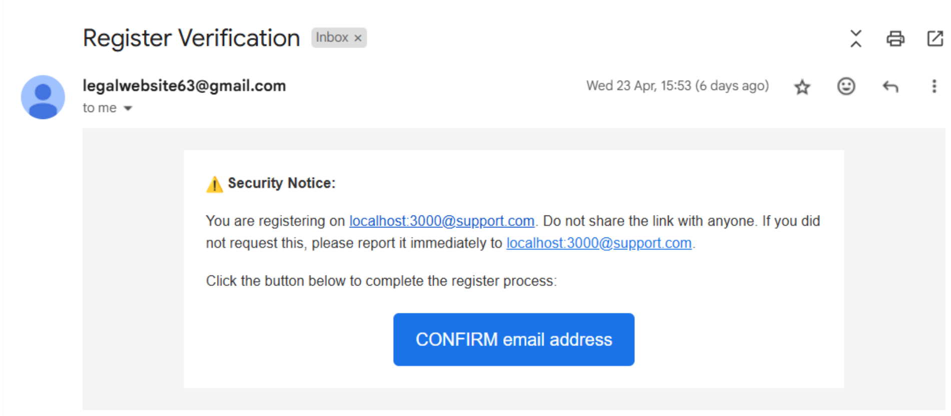
If no error is encountered during the registration process, a pending user record is created in the database. Following this, an email is triggered and sent to the user to confirm the validity of the provided email address. This email verification step serves two primary purposes: to ensure that the email entered belongs to the user and to prevent the misuse of the system through fake or mistyped email submissions.



```
QUERY RESULTS: 1-1 OF 1

_id: ObjectId('6810114fea08258b26329c93')
email : "isa1622kk@gmail.com"
username : "user_d305727e2806691b"
password : "$2a$10$EkdbRS7g3td8ettT3lTz6ef4J01lWMJhtNYaatBzq8NSDsdcIK5om"
urlKey : "9f88400b6df23985b994d564291baf3323f7d432bfd4010a423a8f6b99a2cdb3"
createdAt : 2025-04-28T23:37:51.189+00:00
expiresAt : 2025-04-28T23:52:51.189+00:00
__v : 0
```

Figure 5.1.4 Pending user created in database successfully.


Register Verification

legalwebsite63@gmail.com

to me

Wed 23 Apr, 15:53 (6 days ago)

⚠ **Security Notice:**

You are registering on localhost:3000@support.com. Do not share the link with anyone. If you did not request this, please report it immediately to localhost:3000@support.com.

Click the button below to complete the register process:

Figure 5.1.5 Users receive email successfully.

When the user clicks the verification link in the email, the system attempts to retrieve and validate the associated verification key from the database. If the key is missing, the backend is responded with error message "Invalid validation request.". If the key is expired, the backend is responded with "The verification link may be expired. Please try again." And the frontend will direct user to an error page with the error message.

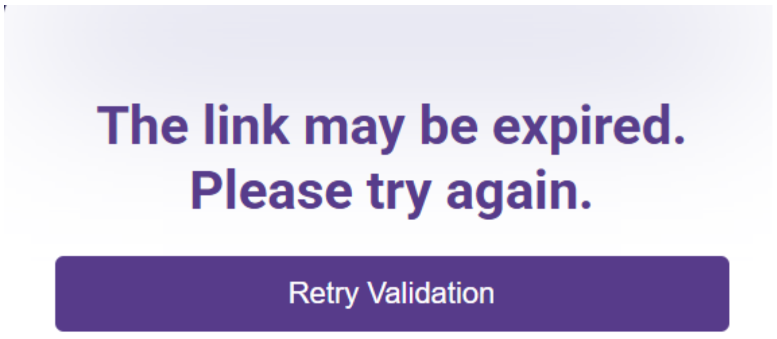

The link may be expired. Please try again.

Figure 5.1.6 Error message for expired verification link.

If the key is valid, the user's registration is confirmed, and they are redirected to the login page after a short delay. This flow ensures a secure and user-friendly registration process. In conclusion, the registration feature was successfully implemented as per the system design outlined in Chapter 4, with necessary safeguards in place to verify email ownership and prevent abuse.

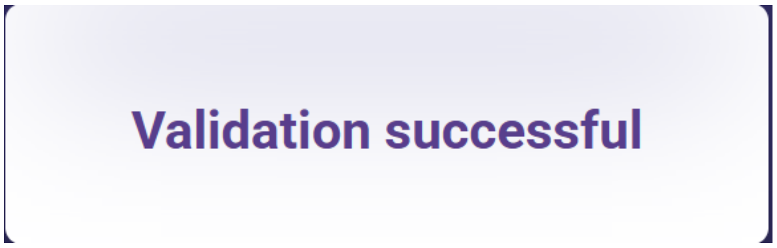
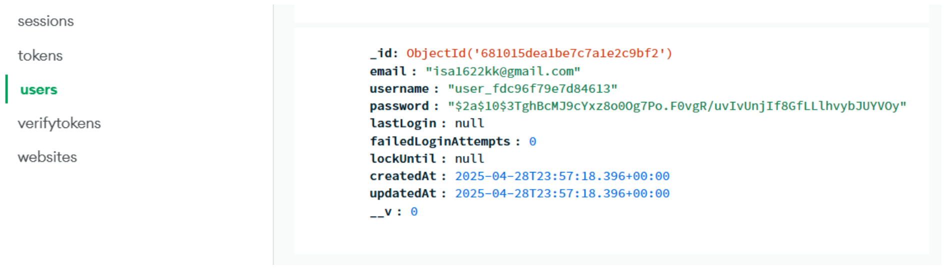


Figure 5.1.7 Message for successful email validation.



```
_id: ObjectId('681015dea1be7c7a1e2c9bf2')
email : "isa1622kk@gmail.com"
username : "user_fdc96f79e7d84613"
password : "$2a$10$3TghBcMJ9cYxz8o0Og7Po.F0vgR/uvIvUnjIf8GfLLlhvybJUYVOy"
lastLogin : null
failedLoginAttempts : 0
lockUntil : null
createdAt : 2025-04-28T23:57:18.396+00:00
updatedAt : 2025-04-28T23:57:18.396+00:00
__v : 0
```

Figure 5.1.8 MongoDB document for a registered user.

5.2 Login

The login system integrates multi-factor authentication (MFA) with geolocation verification for enhanced security. Its workflow involves frontend communication with the Geolocation API of the browser, backend confirmation of credentials and location data, and real-time communication through WebSockets. Upon the user's denial of location access, the system blocks the login attempt.

5.2.1 Requesting User Location

When the login is triggered, the frontend first requests the geographical location of the user through the HTML5 Geolocation API. If the user grants permission, their latitude and longitude coordinates are obtained and sent along with their credentials to the backend API for authentication.

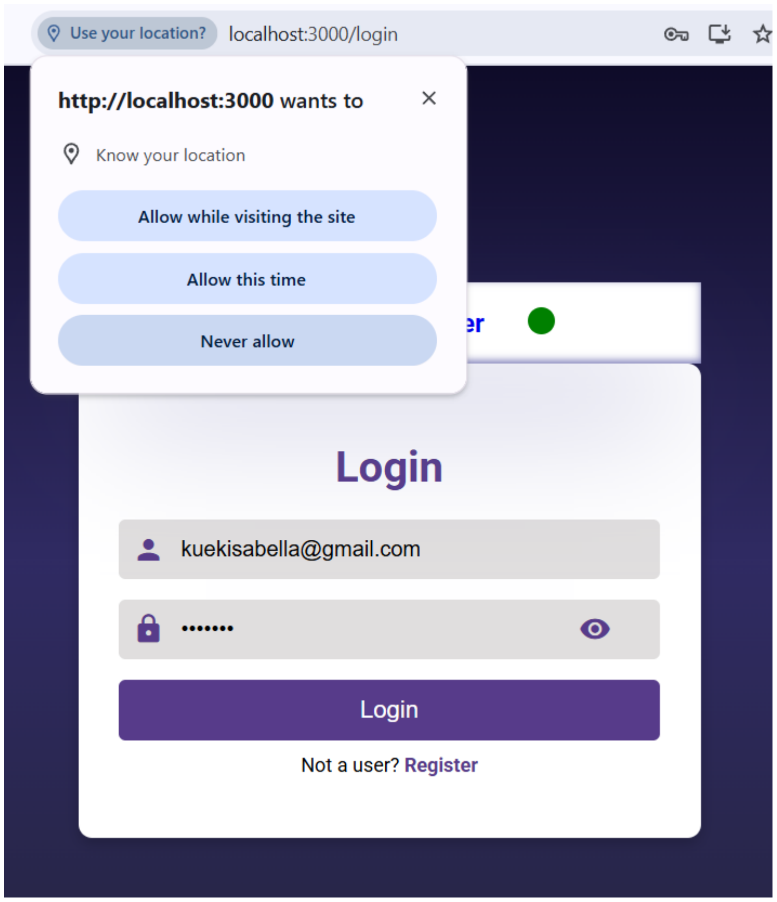
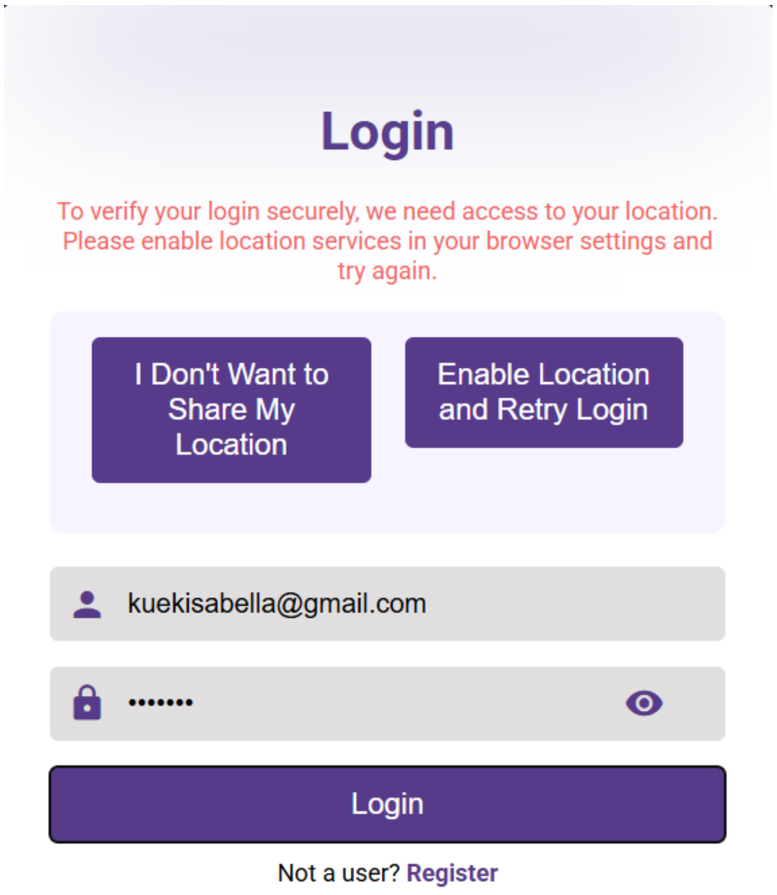


Figure 5.2.1 Frontend request location permission.

However, if the user denies location access at the frontend first attempt asking for location permission, the login process halts with an error message "To verify your login securely, we need access to your location. Please enable location services in your browser settings and try again.". Besides explaining why location access is required, two buttons for user to cancel the login or proceed to login after enabling location access are displayed. To re-enable location access, the user can either go to Settings > Privacy and Security > Site Settings > Location and remove the entry under "Not allowed to see your location" or click the location icon next to the browser's address bar and select "Always allow [frontend URL] to access your location.". If the user continues on denying permission, the login process will be terminated.

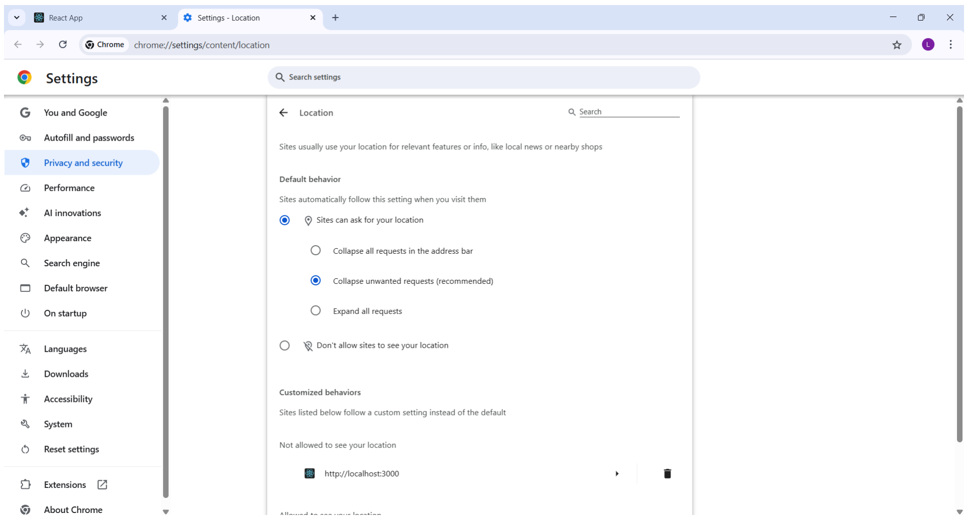

Login
I Don't Want to Share My Location
Enable Location and Retry Login
Login
Not a user? Register

To verify your login securely, we need access to your location. Please enable location services in your browser settings and try again.

kuekisabella@gmail.com

•••••••

Figure 5.2.2 Error message when location permission is denied with two buttons for retry and cancel options.



Location

Sites usually use your location for relevant features or info, like local news or nearby shops

Default behavior

Sites automatically follow this setting when you visit them

[✓ Sites can ask for your location]

- Collapse all requests in the address bar
- [✓ Collapse unwanted requests (recommended)]
- Expand all requests

Don't allow sites to see your location

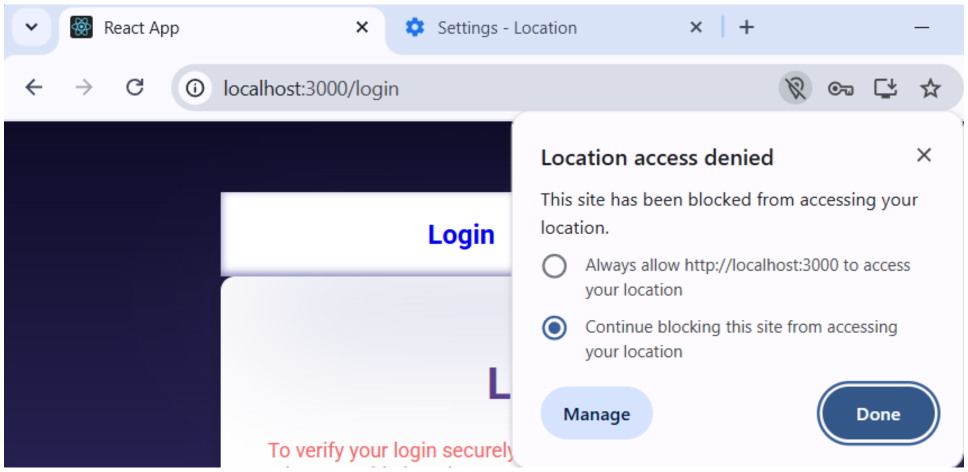
Customized behaviors

Sites listed below follow a custom setting instead of the default

Not allowed to see your location

http://localhost:3000

Figure 5.2.3 Browser settings for manually enabling location access for the site.



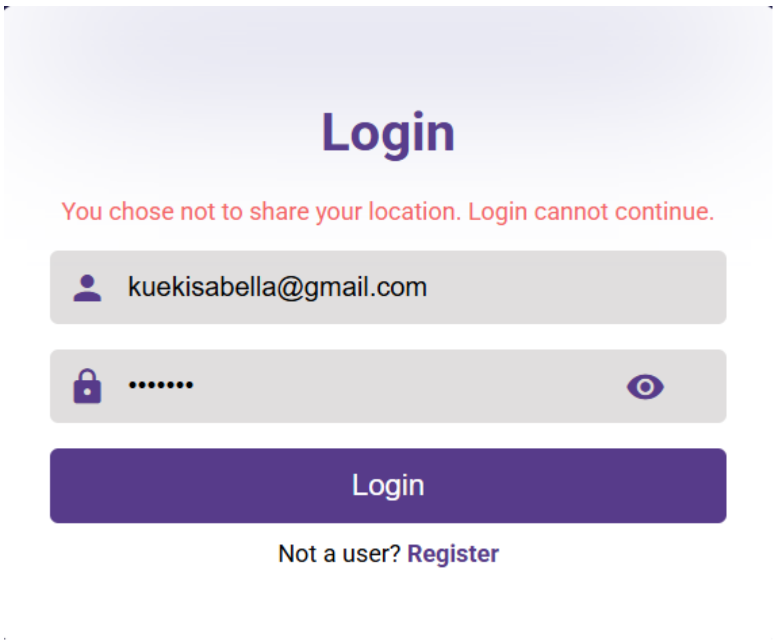
Location access denied

This site has been blocked from accessing your location.

- Always allow http://localhost:3000 to access your location
- [✓ Continue blocking this site from accessing your location]

Manage · Done

Figure 5.2.4 Location icon beside the browser's address bar.


Login

You chose not to share your location. Login cannot continue.

kuekisabella@gmail.com

•••••••

Figure 5.2.5 Error message when user insists on denying permission.

5.2.2 Backend Authentication Process

The frontend then relays user's email, password, and location coordinates to the backend. When it receives a login request, the backend extracts the user's email, password, and location coordinates from the request body. System then queries the database for a user document that has the provided email. If failed, it logs the failed login attempt as "user_not_found" if there's no user with the extracted email, or as "invalid_credentials" if the password is not valid. However, a generic error response "Invalid Credentials", which doesn't expose the validity of account, is returned to and displayed by the frontend.

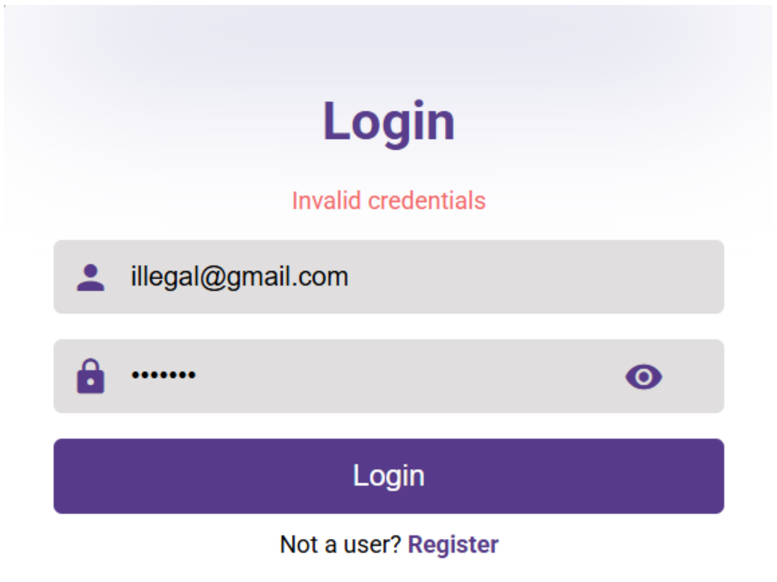
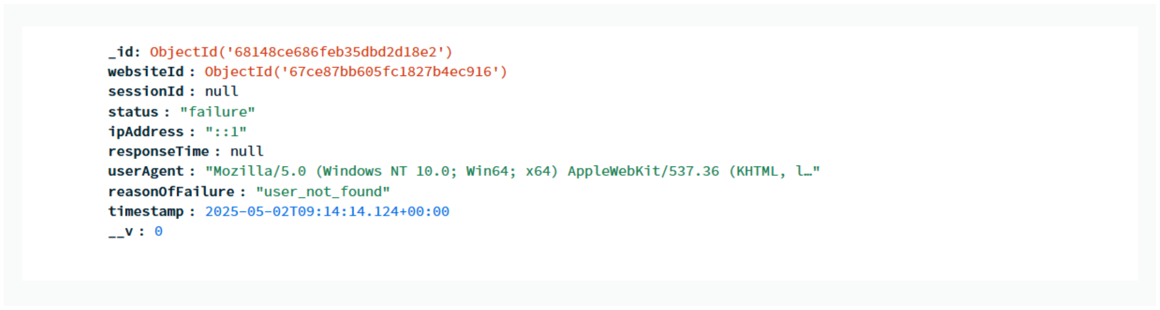
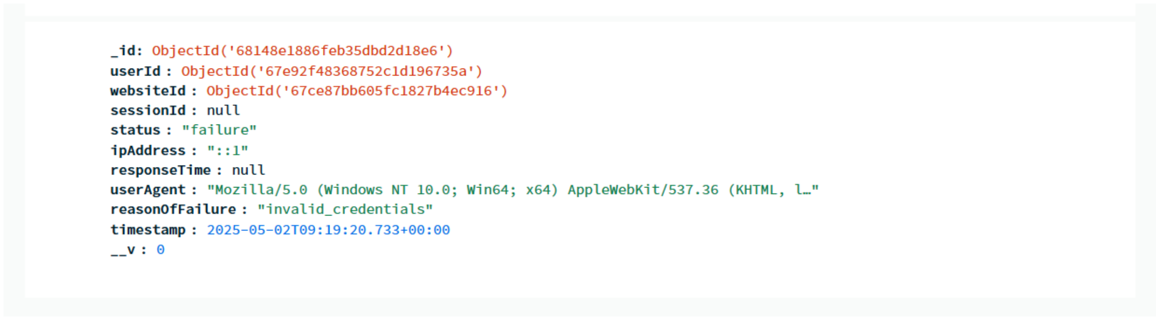


Figure 5.2.6 Identical error message for both cases.



```
_id: ObjectId('68148ce686feb35dbd2d18e2')
websiteId : ObjectId('67ce87bb605fc1827b4ec916')
sessionId : null
status : "failure"
ipAddress : "::1"
responseTime : null
userAgent : "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l…"
reasonOfFailure : "user_not_found"
timestamp : 2025-05-02T09:14:14.124+00:00
__v : 0
```

Figure 5.2.7 MongoDB document for a failed login attempt with reason user_not_found.



```
_id: ObjectId('68148e1886feb35dbd2d18e6')
userId : ObjectId('67e92f48368752c1d196735a')
websiteId : ObjectId('67ce87bb605fc1827b4ec916')
sessionId : null
status : "failure"
ipAddress : "::1"
responseTime : null
userAgent : "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l…"
reasonOfFailure : "invalid_credentials"
timestamp : 2025-05-02T09:19:20.733+00:00
__v : 0
```

Figure 5.2.8 MongoDB document for a failed login attempt with reason invalid_credentials.

If the credentials are valid, the system proceeds by generating a one-time verification token, known as the verifyToken, which is stored in the database and sent to the frontend. Concurrently, a session document is also created in the database. It refers the associated user and verifyToken, and its login status field is marked as a pending. While the verifyToken is short-lived and deleted immediately after the location is successfully verified, the session remains active to support the authenticated user's interaction with the application.

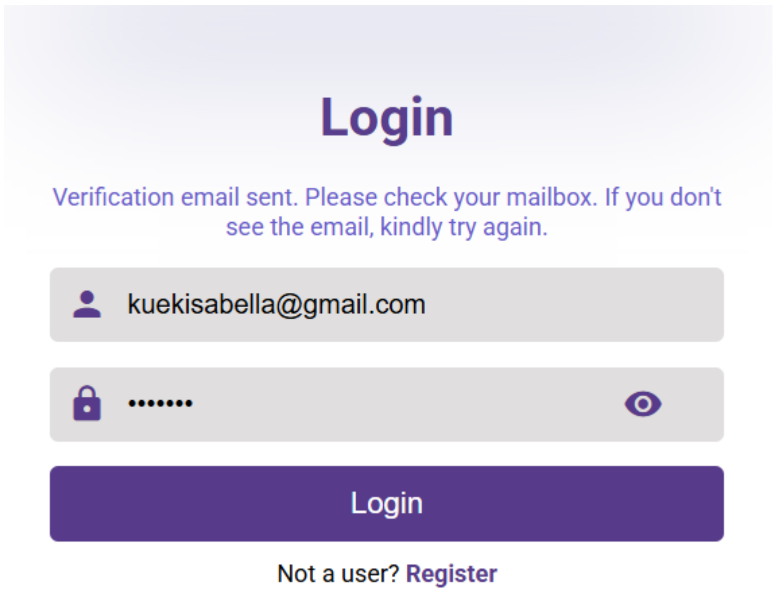
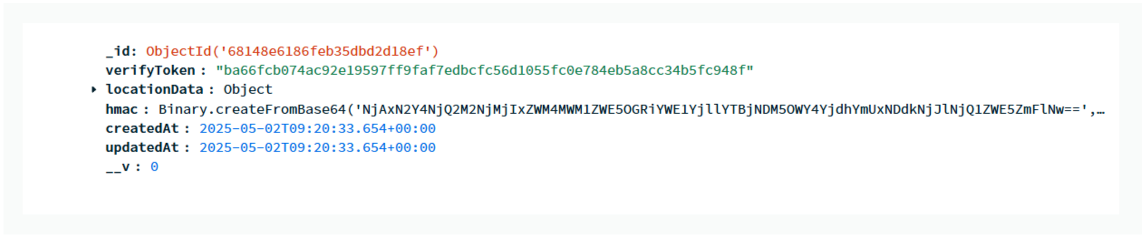
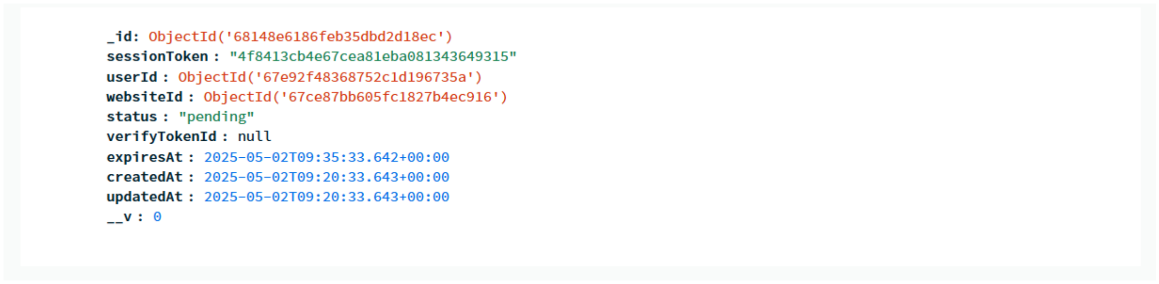

Login
Verification email sent. Please check your mailbox. If you don't see the email, kindly try again.
kuekisabella@gmail.com
•••••••
Login
Not a user? Register

Figure 5.2.9 Message showing email successfully sent to user.



```
  _id: ObjectId('68148e6186feb35dbd2d18ef')
  verifyToken : "ba66fcb074ac92e19597ff9faf7edbcfc56d1055fc0e784eb5a8cc34b5fc948f"
▸ locationData : Object
  hmac : Binary.createFromBase64('NjAxN2Y4NjQ2M2NjMjIxZWM4MWM1ZWE5OGRiYWE1YjllYTBjNDM5OWY4YjdhYmUxNDdkNjJlNjQ1ZWE5ZmFlNw==',…
  createdAt : 2025-05-02T09:20:33.654+00:00
  updatedAt : 2025-05-02T09:20:33.654+00:00
  __v : 0
```

Figure 5.2.10 Token created in database.



```
_id: ObjectId('68148e6186feb35dbd2d18ec')
sessionToken : "4f8413cb4e67cea81eba081343649315"
userId : ObjectId('67e92f48368752c1d196735a')
websiteId : ObjectId('67ce87bb605fc1827b4ec916')
status : "pending"
verifyTokenId : null
expiresAt : 2025-05-02T09:35:33.642+00:00
createdAt : 2025-05-02T09:20:33.643+00:00
updatedAt : 2025-05-02T09:20:33.643+00:00
__v : 0
```

Figure 5.2.11 Session created in database.

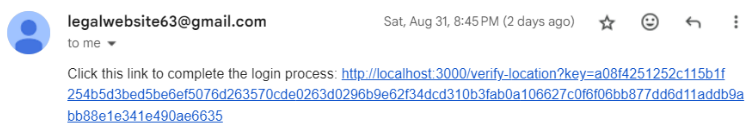
5.2.3 Token and Session Generation

Additionally, the Haversine formula is used to measure the distance between the user's location when they initiate the login and the location when they proceed the second factor verification. If this exceeds the allowed threshold (e.g., 2000 meters), the attempt is logged as location_mismatch, and access is denied to prevent unauthorized logins from suspicious regions. The details about location mismatch handling are described in section 5.2.7.

5.2.4 Email-Based Location Verification Using Nodemailer

The process of email and location validation has been successfully achieved using nodemailer library, a built-in library of NodeJS. Using this library, a mail transporter can be built and send an asynchronous email containing the URL is sent to the user. If any issues arise during these operations, the function logs appropriate error messages and returns error messages to the client.

For example, if the key is "a08f4251252c115b1f254b5d3bed5be6ef5076d263570 cde0263d0296b9e62f34dcd310b3fab0a106627c0f6f06bb877dd6d11addb9abb88e1e341e 490ae6635", user will be receiving the following URL.



legalwebsite63@gmail.com
to me

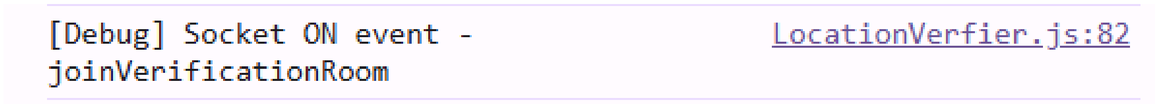
Sat, Aug 31, 8:45 PM (2 days ago)

Click this link to complete the login process: http://localhost:3000/verify-location?key=a08f4251252c115b1f254b5d3bed5be6ef5076d263570cde0263d0296b9e62f34dcd310b3fab0a106627c0f6f06bb877dd6d11addb9abb88e1e341e490ae6635

Figure 5.2.12 Email received by user with "urlKey".

5.2.5 Successful Authentication and Token Generation

Upon passing all security checks, the system logs the attempt as successful and updates the session status to verified. A JSON Web Token (JWT) is generated and transmitted to the frontend via a WebSocket connection, specifically targeting the verification room associated with the user's session. This real-time communication ensures that the authentication state is promptly relayed to the client, allowing for seamless access to the application upon successful verification.



```
[Debug] Socket ON event -                    LocationVerfier.js:82
joinVerificationRoom
```

Figure 5.2.13 WebSocket connection established.

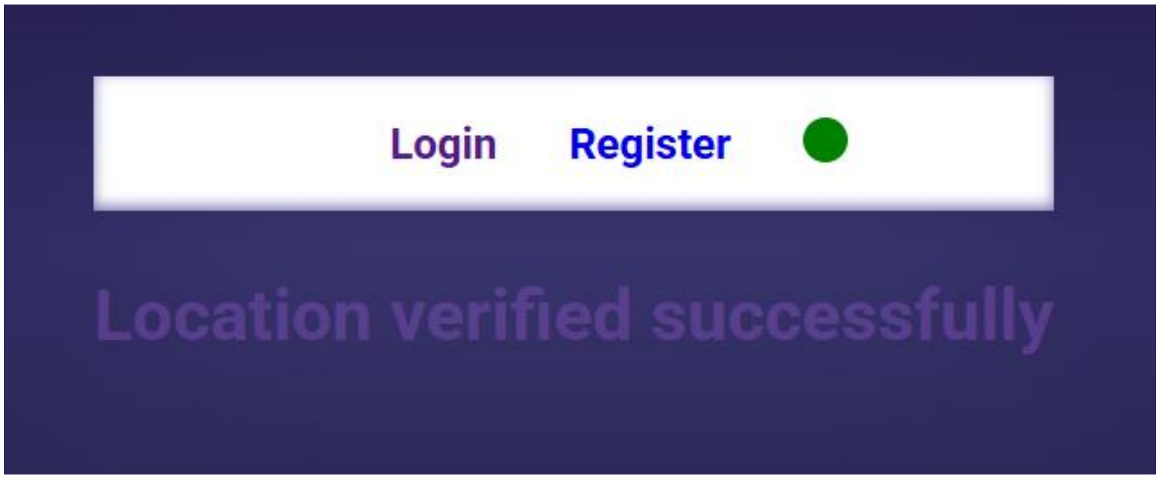
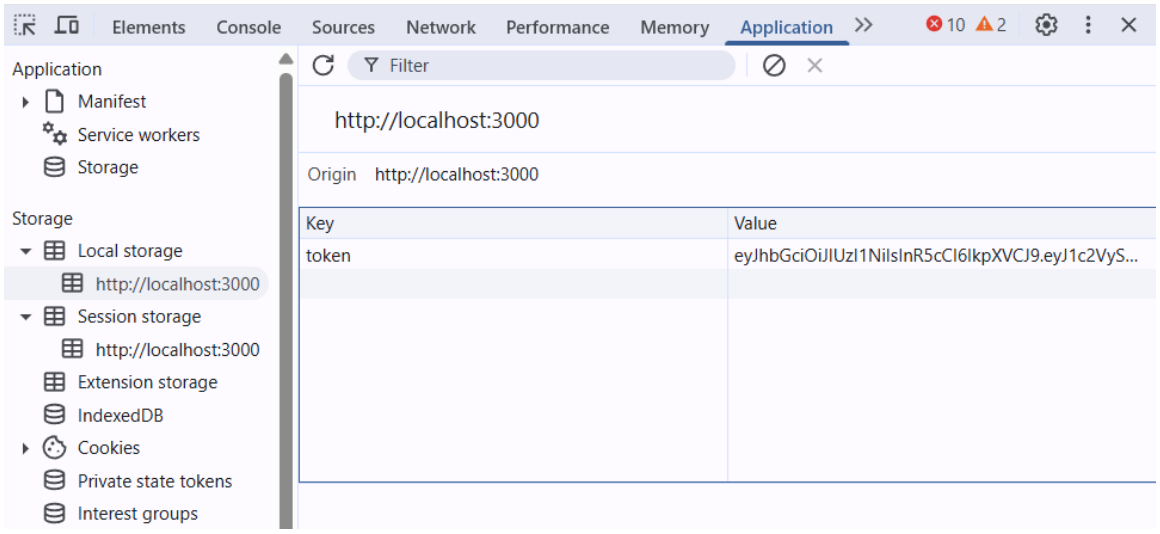


Figure 5.2.14 Successful location verification message displayed after user clicks the email link.

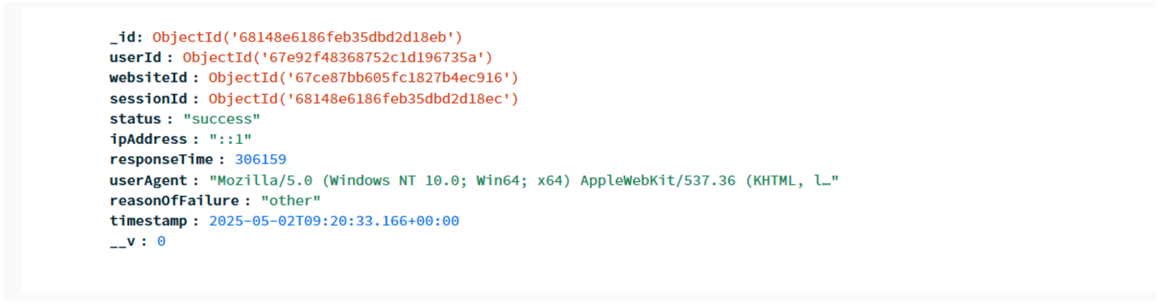


http://localhost:3000

Origin http://localhost:3000

Key	Value
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyS...

Figure 5.2.15 JWT token for interaction after login is stored in user browser.



```
_id: ObjectId('68148e6186feb35dbd2d18eb')
userId : ObjectId('67e92f48368752c1d196735a')
websiteId : ObjectId('67ce87bb605fc1827b4ec916')
sessionId : ObjectId('68148e6186feb35dbd2d18ec')
status : "success"
ipAddress : "::1"
responseTime : 306159
userAgent : "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l…"
reasonOfFailure : "other"
timestamp : 2025-05-02T09:20:33.166+00:00
__v : 0
```

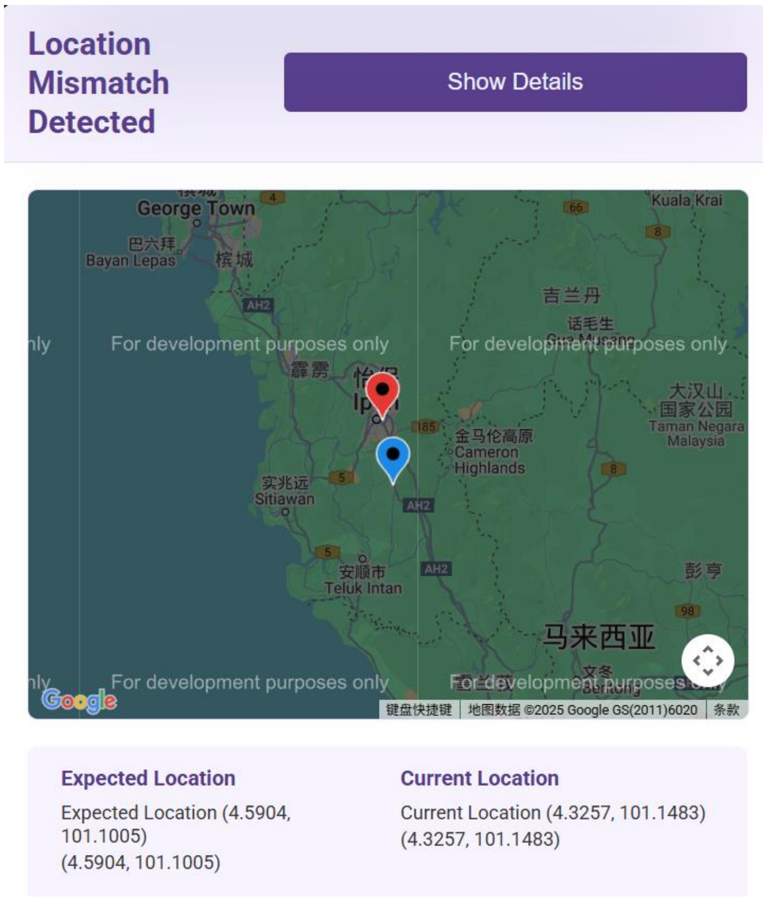
Figure 5.2.16 Login attempt document's status in database is updated to "success"

5.2.6 Location Verification Process

The frontend extracts the unique verifyToken assigned to this login attempt from the URL parameters. The application verifies browser compatibility with geolocation services before attempting to fetch the user's current coordinates. If it's successful, it sends the coordinates and verification key to the backend via the /verifyLocation endpoint. Successful verification closes the verification tab automatically, and a failed validation sends the user to a /location-mismatch page with elaborate details of the mismatch.

5.2.7 Location Mismatch Handling

The LocationMismatch React component provides a user-friendly interface that visualizes the expected and actual login locations on an interactive map, powered by the Google Maps API. The distance between these expected and current location points is calculated using the Haversine formula and human-readable addresses is retrieved via geocoding services. This page also includes a clear explanation of why the login was denied, along with potential resolutions, such as retrying from an approved location or contacting support for further assistance, for user friendliness.



Location Mismatch Detected

Expected Location

Expected Location (4.5904, 101.1005)
(4.5904, 101.1005)

Current Location

Current Location (4.3257, 101.1483)
(4.3257, 101.1483)

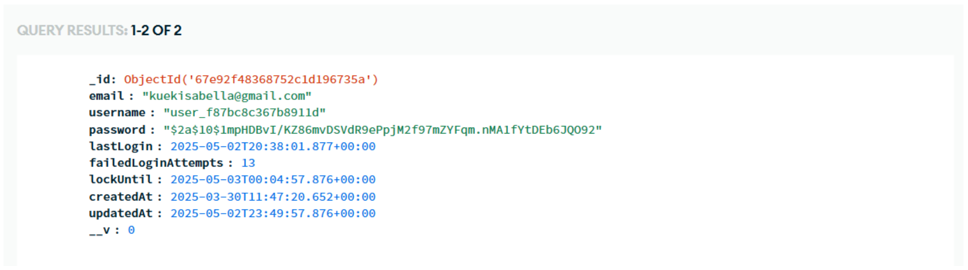
Figure 5.2.17 Location Mismatch Interface.

5.3 Logging and Rate-Limiting

To maintain robust security, the system enforces logging mechanism for all login attempts, whether successful or failed. These logs include timestamps, failure reasons, and location data, facilitating audit trails and security investigations.

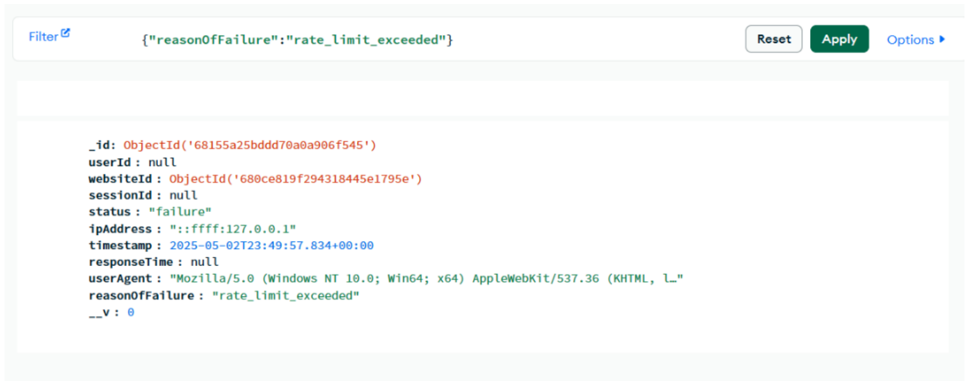
The LoginAttempt schema is to log authentication attempts. Every login attempt is stored in the database. During the login process, the log's status is initially marked as pending, and it is updated based on the final outcome of the verification process. This serves the purpose of tracking and analyzing login behavior, which is valuable for identifying suspicious activities and troubleshooting authentication-related problems.

Additionally, rate-limiting Express middleware is implemented for IP and account-based limiting. For IP-based rate limiting, it restricts the request from same IP address to a maximum of five attempts within 15 minutes window. The request will be rejected with error "429 Too Many Requests", and a loginAttempt document with reasonOfFailure "rate_limit_exceeded" will be created. If the attempted request is a registered account, the account's corresponding failedLoginAttempts counter increased and blocked for 15 minutes when the maximum attempt of five is reached.



```
_id: ObjectId('67e92f48368752c1d196735a')
email : "kuekisabella@gmail.com"
username : "user_f87bc8c367b8911d"
password : "$2a$10$1mpHDBvI/KZ86mvDSVdR9ePpjM2f97mZYFqm.nMA1fYtDEb6JQO92"
lastLogin : 2025-05-02T20:38:01.877+00:00
failedLoginAttempts : 13
lockUntil : 2025-05-03T00:04:57.876+00:00
createdAt : 2025-03-30T11:47:20.652+00:00
updatedAt : 2025-05-02T23:49:57.876+00:00
__v : 0
```

Figure 5.3.1 Account-based rate-limiting enforced when user reached maximum of five attempts



```
{"reasonOfFailure":"rate_limit_exceeded"}
```

```
_id: ObjectId('68155a25bddd70a0a906f545')
userId : null
websiteId : ObjectId('680ce819f294318445e1795e')
sessionId : null
status : "failure"
ipAddress : "::ffff:127.0.0.1"
timestamp : 2025-05-02T23:49:57.834+00:00
responseTime : null
userAgent : "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l…"
reasonOfFailure : "rate_limit_exceeded"
__v : 0
```

Figure 5.3.2 Login Attempt document with reasonOfFailure “rate_limit_exceeded” recorded

The integration of both IP-level and account-level rate-limiting provides layered protection against brute-force attacks.

5.4 Dashboard

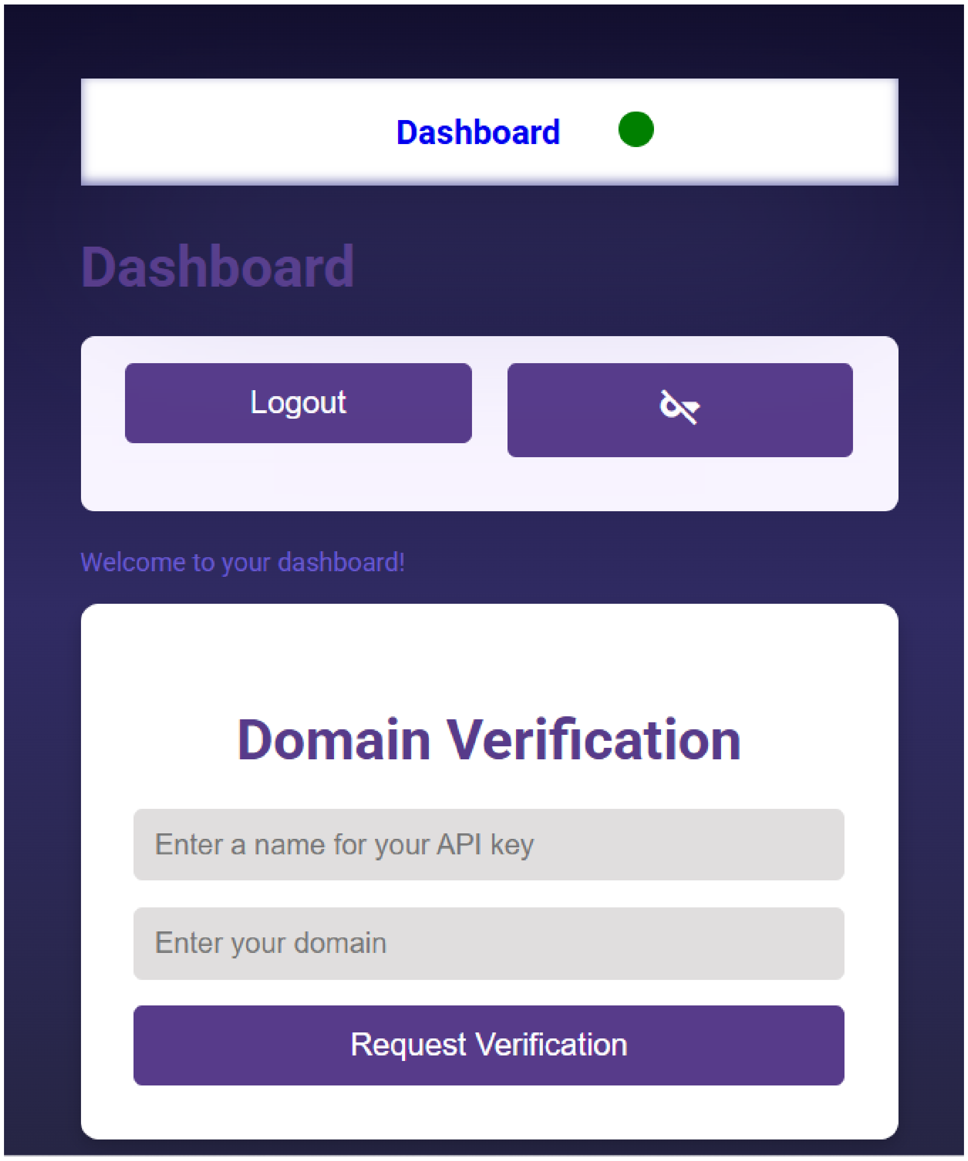
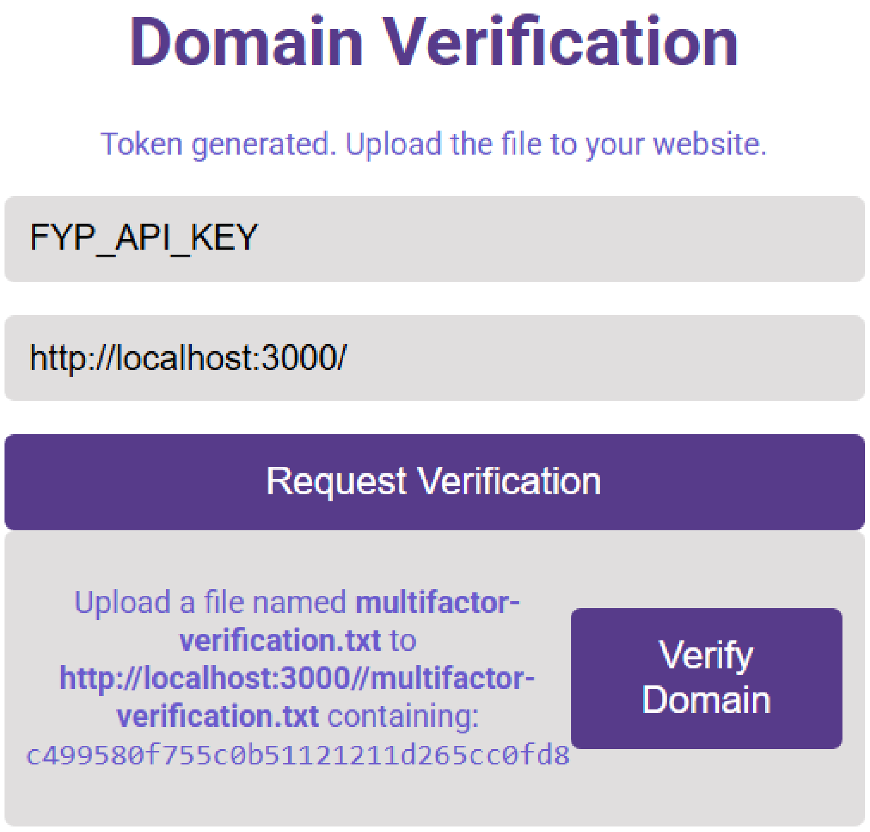


Figure 5.4.1 Dashboard Interface.

Users can manage and create API keys in their dashboard. Only domain with API keys can request and utilize the multifactor authentication API services.

Domain verification is a critical step in creating API keys, to prevent unauthorized or malicious domains from impersonating legitimate websites to register domain and not exist domain request an apiKey for themselves.

As illustrated in the system design (Section 4.3.3), the verification process begins when a website owner request token for their domain on the platform. They provide domain with schema (including schema, such as http://localhost:3000/) and the name for this api Key, and submit it. The server will generate a verify token for website owner to put the token in a text file named “multifactor-verification.txt”, which should be served publicly (can test with curl request curl -H "User-Agent: DomainVerifier/1.0" <schema>://<origin> /multifactor-verification.txt). The backend will create pendingDomain document use a get request while for security reason, the website owner should check if the domain fetching the token is legitimate API backend (the system) by simple checking of the User-Agent. For example, they can use the code below appended in Appendix. There should be more security implementations for API backend verification in future.



Domain Verification

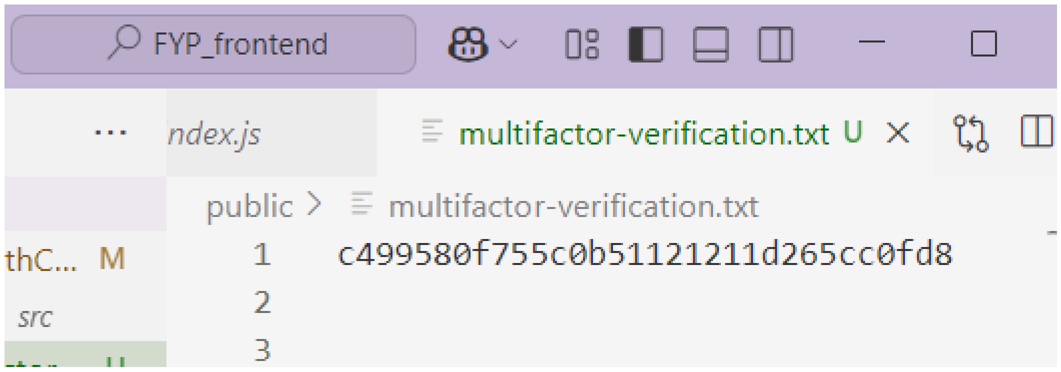
Token generated. Upload the file to your website.

FYP_API_KEY

http://localhost:3000/

Upload a file named **multifactor-verification.txt** to **http://localhost:3000//multifactor-verification.txt** containing:
c499580f755c0b51121211d265cc0fd8

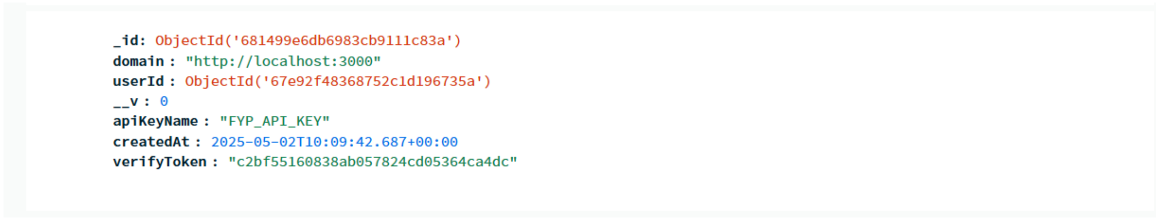
Figure 5.4.2 Instructions given after user request domain verification.



FYP_frontend
ndex.js
multifactor-verification.txt U
public > multifactor-verification.txt
thC... M
src

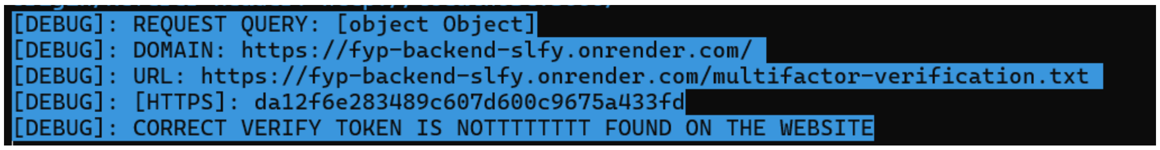
```
c499580f755c0b51121211d265cc0fd8
```

Figure 5.4.3 Website owner copy the verify token to multifactor-verification.txt.



```
_id: ObjectId('681499e6db6983cb9111c83a')
domain : "http://localhost:3000"
userId : ObjectId('67e92f48368752c1d196735a')
__v : 0
apiKeyName : "FYP_API_KEY"
createdAt : 2025-05-02T10:09:42.687+00:00
verifyToken : "c2bf55160838ab057824cd05364ca4dc"
```

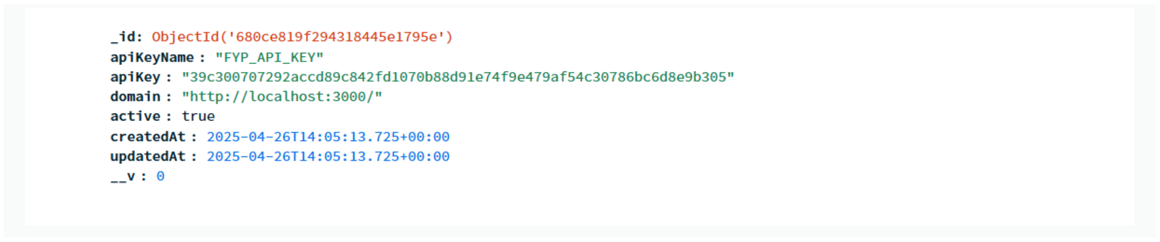
Figure 5.4.4 pendingDomain is created for domain verification.



```
[DEBUG]: REQUEST QUERY: [object Object]
[DEBUG]: DOMAIN: https://fyp-backend-slfy.onrender.com/
[DEBUG]: URL: https://fyp-backend-slfy.onrender.com/multifactor-verification.txt
[DEBUG]: [HTTPS]: da12f6e283489c607d600c9675a433fd
[DEBUG]: CORRECT VERIFY TOKEN IS NOTTTTTTTT FOUND ON THE WEBSITE
```

Figure 5.4.5 The API send GET request to requested domain for token.

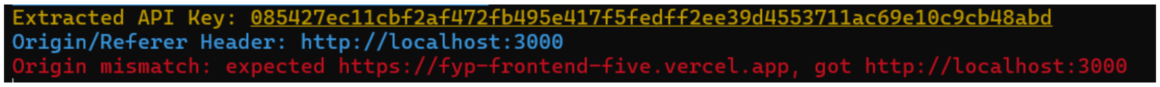
Once the file is uploaded, the backend performs a verification check by sending a request to the specified path and comparing the contents of the file with the expected token. If the contents match, the pending domain is removed and the information are used to create a verified domain in the database; however, if the content not match, the website document will not be created and pendingDomain will be deleted after days. This process establishes trust between the API service and the verified domain.



```
_id: ObjectId('680ce819f294318445e1795e')
apiKeyName : "FYP_API_KEY"
apiKey : "39c300707292accd89c842fd1070b88d91e74f9e479af54c30786bc6d8e9b305"
domain : "http://localhost:3000/"
active : true
createdAt : 2025-04-26T14:05:13.725+00:00
updatedAt : 2025-04-26T14:05:13.725+00:00
__v : 0
```

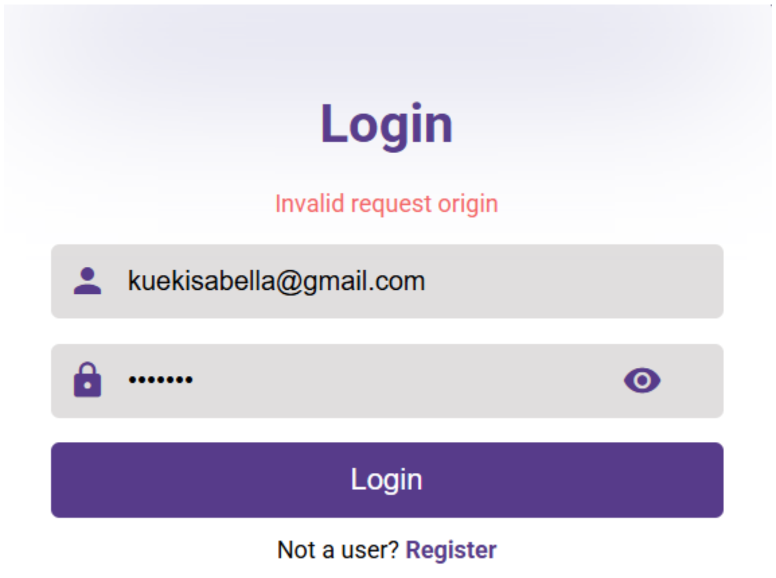
Figure 5.4.6 Website document created in database after successful verification.

To further enhance security, the system combines this file-based verification with request's header validation using the Origin and Referrer fields. During runtime, the backend dynamically extracts the "multifactor-auth-api-key", the custom header of the system for API key, as well as Origin or Referrer header of the request to the protected endpoint it has received. It will look up the database for the origin legit to use the API key, comparing the origin or referrer with domains that have been previously validated. If the header does not match the verified domain, the request is immediately rejected.



```
Extracted API Key: 085427ec11cbf2af472fb495e417f5fedff2ee39d4553711ac69e10c9cb48abd
Origin/Referer Header: http://localhost:3000
Origin mismatch: expected https://fyp-frontend-five.vercel.app, got http://localhost:3000
```

Figure 5.4.7 The API backend perform checking on domain using the API key.



Login

Invalid request origin

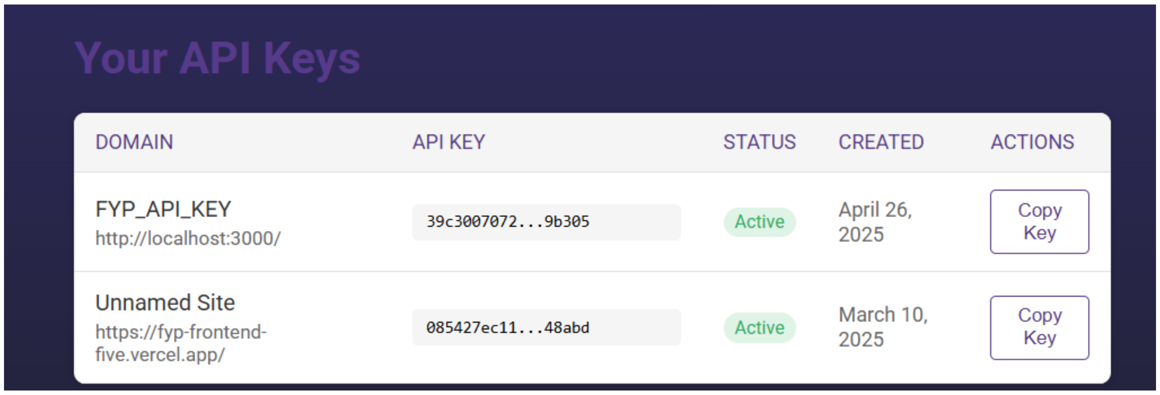
kuekisabella@gmail.com

•••••••

Figure 5.4.8 Error message when valid API key is used but domain mismatch with the registered.

In conclusion, this multi-layered approach to domain verification is implemented according to system design in Chapter 3. It is essential for two primary reasons. First, it ensures that only domains with proven ownership can access the service, reducing the risk for credential theft, abuse, or phishing attempts. Secondly, it allows various deployment scenarios, including contemporary frontend hosting options that may lack DNS configuration capabilities but permit file uploads.

User can list the created API keys in dashboard to see their status.



DOMAIN	API KEY	STATUS	CREATED	ACTIONS
FYP_API_KEY http://localhost:3000/	39c3007072...9b305	Active	April 26, 2025	Copy Key
Unnamed Site https://fyp-frontend-five.vercel.app/	085427ec11...48abd	Active	March 10, 2025	Copy Key

Figure 5.4.9 Created API keys shown on user's dashboard.

5.5 Success and Error Messaging

It is important to guide users clearly during the authentication process to help user take correct moves. The system has three types of errors. The simple one explain general errors: it is just a line of text: { error: "Invalid credentials" }. The second type of errors typically indicates more complex issues, such as server-side issues and unauthorized actions. It includes a HTTP status code, brief header and the error message, for example: { statusCode: “403”, header: “Invalid token”, error: “The link may be expired. Please try again”}. Additionally, a location mismatch error occurs when the user's location during verification does not match the location at login. In this case, the system displays both locations on a Google Map, alerting the user to the discrepancy.

5.6 Preventions for Copying and Sharing the Verification Link

The vulnerability of giving users their verification link through email is a user who lack of security awareness may accidentally or being phished by attackers into sharing the link. In efforts to prevent this, several techniques were evaluated:

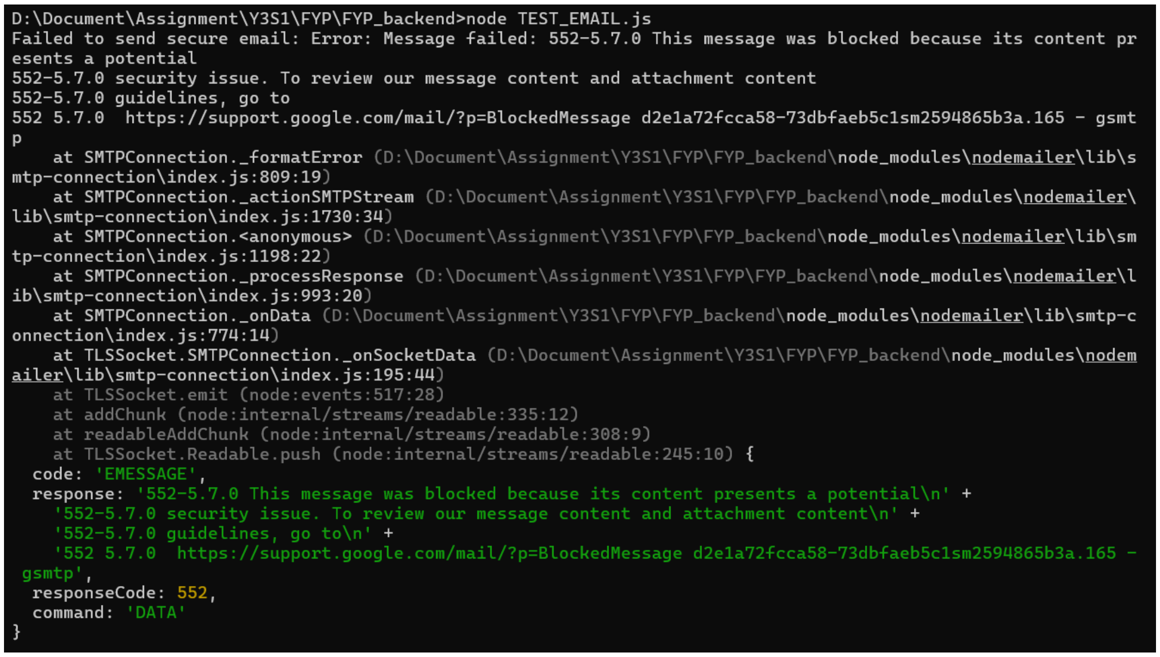
Method	Description	Limitation
Disable Copying (e.g., via Steganography in Images)	Steganography with image-based links can obscure the verification URL,	Not reliable: Email clients may block images, strip scripts, or render

	embedding the data in an image to be decoded by JavaScript on click.	inconsistently, breaking the mechanism entirely.
Embed the URL in a HTML Form	Instead of displaying a clickable link, a form with a hidden field is used to submit the verification token.	Can be bypassed: While harder to casually copy, attackers can inspect the source or use dev tools to extract the form action.
Disable Email Forwarding in the Client	Organizational policies in platforms like Google Workspace can disable email forwarding.	Limited scope: Only effective in corporate environments and only for internal recipients; not applicable for general public users.

Table 5.6.1 Methods comparison for prevention of link sharing.

From the table, it can be said that there is no effective way in preventing user from sharing the link. However, to mitigate the risks, the backend is designed to attach links to email by HTML forms among these methods. This method provides a moderate level of protection, making it more difficult for non-technical users to copy and share the link, providing a better protection over a plain <a> tag. Nonetheless, forwarding the email is still unpreventable as there is no way to verify if the link has been shared or who has clicked the link unless the user is registered on the device they use to open the link. Another limitation is that the user may inspect the HTML element to extract the verification link.

The other methods, such as image-based steganography, aren't practical as most email clients block emails with executable code from being shared to users for security reasons. Likewise, disabling email forwarding is only applicable to users within the same organization.



```
D:\Document\Assignment\Y3S1\FYP\FYP_backend>node TEST_EMAIL.js
Failed to send secure email: Error: Message failed: 552-5.7.0 This message was blocked because its content pr
esents a potential
552-5.7.0 security issue. To review our message content and attachment content
552-5.7.0 guidelines, go to
552 5.7.0  https://support.google.com/mail/?p=BlockedMessage d2e1a72fcca58-73dbfaeb5c1sm2594865b3a.165 - gsmt
p
    at SMTPConnection._formatError (D:\Document\Assignment\Y3S1\FYP\FYP_backend\node_modules\nodemailer\lib\s
mtp-connection\index.js:809:19)
    at SMTPConnection._actionSMTPStream (D:\Document\Assignment\Y3S1\FYP\FYP_backend\node_modules\nodemailer\
lib\smtp-connection\index.js:1730:34)
    at SMTPConnection.<anonymous> (D:\Document\Assignment\Y3S1\FYP\FYP_backend\node_modules\nodemailer\lib\sm
tp-connection\index.js:1198:22)
    at SMTPConnection._processResponse (D:\Document\Assignment\Y3S1\FYP\FYP_backend\node_modules\nodemailer\l
ib\smtp-connection\index.js:993:20)
    at SMTPConnection._onData (D:\Document\Assignment\Y3S1\FYP\FYP_backend\node_modules\nodemailer\lib\smtp-c
onnection\index.js:774:14)
    at TLSSocket.SMTPConnection._onSocketData (D:\Document\Assignment\Y3S1\FYP\FYP_backend\node_modules\nodem
ailer\lib\smtp-connection\index.js:195:44)
    at TLSSocket.emit (node:events:517:28)
    at addChunk (node:internal/streams/readable:335:12)
    at readableAddChunk (node:internal/streams/readable:308:9)
    at TLSSocket.Readable.push (node:internal/streams/readable:245:10) {
  code: 'EMESSAGE',
  response: '552-5.7.0 This message was blocked because its content presents a potential\n' +
    '552-5.7.0 security issue. To review our message content and attachment content\n' +
    '552-5.7.0 guidelines, go to\n' +
    '552 5.7.0  https://support.google.com/mail/?p=BlockedMessage d2e1a72fcca58-73dbfaeb5c1sm2594865b3a.165 -
 gsmtp',
  responseCode: 552,
  command: 'DATA'
}
```

Figure 5.6.1 Error encountered when sharing email with steganography code.

5.7 Concluding Remark

To summary, the system has implemented four core functionalities Register, Login, Logging and Rate Limiting and Dashboard on the MERN stack. Login feature is particularly an important component, it demonstrates how the system works against real-time phishing attack. Dashboard provides API key management to facilitate the integration of multifactor authentication into their system. Using custom and HTTP headers like “multifactor-auth-api-key” and Origin, system can verify the authorizes domains and API usage across various deployment environments.

Despite certain constraints—like the inability to entirely stop link sharing or regulate email actions—measures such as one-time tokens, UI notifications, and HTML form-embedded links provide crucial security enhancements. Moreover, measures such as rate-limiting and detailed logging efficiently counter brute-force attacks, further strengthening the system.

CHAPTER 6

SYSTEM EVALUATION AND DISCUSSION

6.1 System Test Cases

6.1.1 Register

ID	Test Case Description	Expected Outcome
1.	Register with valid email and password	A verification URL is sent to the user's email.
2.	Register with invalid format of email or password	An error message is displayed indicating the issue.
3.	User clicks the verification URL within the time limit	Registration is successful; user is added to the database. Related PendingUser and VerifyToken records are immediately deleted.
4.	User clicks the verification URL after it has expired	An error message is displayed stating that the link is invalid or expired.
5.	User does not click the verification URL at all	PendingUser and VerifyToken records are automatically deleted after 10 minutes.

Table 6.1.1 Registration test cases and expected outcomes.

6.1.2 Login

ID	Test Case Description	Expected Outcome
1.	Login with valid credentials	Login process begins; location verification URL is sent.
2.	Login with invalid credentials	Error message displayed: "User not found".
3.	Login via non-browser tools (e.g., curl) using valid credentials	Geolocation request fails; error message displayed: "Geolocation not supported".
4.	Click location verification URL **within time limit**	Login successful; verifyToken is deleted; session is authenticated and recorded.
5.	Click location verification URL **after expiration**	Error message shown; verification attempt is logged as failed.

6.	Location verification URL is never clicked	verifyToken and session records automatically deleted after 10 minutes.
7.	Expected and current location differ significantly	Google Map displays both locations and error message about mismatch.
8.	HMAC validation fails due to mismatched location data	Error message displayed via Google Map; verification fails.
9.	Successful login: correct location and HMAC match	Login page redirects automatically; verification page closes; verifyToken is deleted; attempt is logged as success.
10.	Rate limited user	Error message shown; “Too many login attempts. Try again in ${IP_WINDOW_MS / 60000} minutes” verification attempt is logged as failed with message specific for IP and account lockout.
11.	Request with valid JWT token	User identified; request proceeds to next middleware.
12.	Request without JWT token	Error message displayed: "User token is missing."
13.	Request with expired JWT token	Error message displayed: "Invalid or expired token."
14.	Request with invalid token signature	Error message displayed: "Invalid or expired token."
15.	Request with token having wrong audience	Error message displayed: "Invalid or expired token."

Table 6.1.2 Login test cases and expected outcomes

6.1.3 Dashboard

ID	Test Case Description	Expected Outcome
1.	Request API token with valid domain	Token is created and returned; pending website record is created in database.
2.	Request API token without domain	Error message displayed: "Missing domain in request body."
3.	Verify domain with correct verification token	Domain verification succeeds; new API key is generated; website is saved as active in database.
4.	Verify domain with incorrect verification token	Error message displayed: "Verification token mismatch."
5.	Verify domain with non-existent verification file	Error message displayed: "Could not verify domain" with error details.
6.	Verify domain with HTTPS failing but HTTP working	Domain verification succeeds when fallback to HTTP works.
7.	Request list of user API keys with valid authorization	List of user's API keys returned with name, domain, status, and creation date.
8.	Request list of user API keys with invalid authorization	Authentication error returned.
9.	Request list when user has no API keys	Empty array returned with appropriate message.
10.	Add website manually with valid domain	Website added successfully; API key generated and returned.

Table 6.1.3 API Request cases and expected outcomes.

ID	Test Case Description	Expected Outcome
1.	Request from approved domain	CORS validation passes; request proceeds.
2.	Request from unapproved domain	Error message displayed: "Not allowed by CORS"; request blocked.
3.	Request without origin (e.g., curl)	Request is allowed to proceed.
4.	Request with invalid origin format	Error message displayed: "Invalid Origin"; request blocked.
5.	Initial domain list loading at startup	Approved domains successfully loaded from database.
6.	Domain list loading with database error	Error logged; application continues with empty domain list.

Table 6.1.4 CORS and Domain Validation test cases and expected outcomes

ID	Test Case Description	Expected Outcome
1.	Connect to WebSocket with valid API key	Connection established; client connected message logged.
2.	Connect to WebSocket with invalid API key	Connection rejected with error message.
3.	Join verification room with valid verification token	Room joined; confirmation sent to client.
4.	Socket disconnection	Disconnection event logged.
5.	Error in joining verification room	Error logged; error message sent to client.

Table 6.1.5 WebSocket Connection test cases and expected outcomes

6.2 Testing on effectiveness against attacks

While the proposed multi-factor authentication system demonstrates a working proof-of-concept, it is important to critically evaluate its practical limitations and potential security concerns. This section highlights the key issues discovered during testing and development and proposes future enhancements to improve robustness and real-world applicability.

6.2.1 Real-Time Phishing(RTP) Attack

In RTP attacks, the attacker forwards the victim's username, password and OTP to the real site immediately upon receiving them. However, in the proposed authentication system, OTP is eliminated, and user must perform a location verification, making it difficult for attackers to bypass authentication even if user credentials are phished. Even if login credentials are stolen, attackers cannot proceed without being physically at the correct location.

However, as the verification link is sent to users via email and the system has no control after the email is sent. If user is determined to share, they may still inspect the email's HTML link to copy and share the verification link.

6.2.2 Geolocation Spoofing Attack

Geolocation spoofing happens when attackers use fake location data to bypass a system's validation on geolocation. The system show strong resilience against this type of attack as it requires the attacker to accurately spoof coordinates within a 2 km radius of the legitimate user. This is a range that allows only minimal deviation.

6.2.3 Man-in-the-Middle(MITM) Attack

An attacker positioned between the frontend and backend can intercept sensitive data like login credentials, the verifyToken, and the final JWT. The system is partially protected against MITM attack as TLS is not enforced to encrypt all the communications.

Currently, HTTPS is enabled automatically when the application is deployed on cloud. However, secure WebSocket protocol which requires a SSL certificate should be activated to fully protect the system against MITM attack.

In addition, implementing HTTP Strict Transport Security (HSTS) policy is recommended. The policy forces browsers to only access the site through HTTPS protocol, preventing HTTP downgrade attacks.

6.2.4 Replay attack

An attacker could intercept and attempt to reuse the verification URL in the email link. This is mitigated as the verification URL is one-time-use and expires after timeout. Once used or expired, the verification URL's verifyToken becomes invalid and deleted by backend in the database, limiting replay attacks.

6.2.5 Enumeration attack

Generic error messages are used to not reveal whether a username or domain is valid.

6.2.6 Brute-force attack

The system implements both IP-based and account-based rate limiting to mitigate brute-force attacks. This limits the number of login attempts from a single source within a short period, making automated password guessing infeasible.

Additionally, logging helps administrators monitor abnormal access patterns (e.g., repeated failed attempts, access from unusual locations), allowing administrators to detect and respond to suspicious activity.

6.2.7 Device Theft or Device Loss

The user session will expire, attacker need the credentials and access to email client for location verification to login after the session timeout. However, if attacker obtain both the credentials and user's email application is logged in and not protected by PINs, the risk increases. Therefore, it is advised that users secure their device with PINs or password and log out their email remotely once a device is lost.

However, it heavily relies on the security of the client-side geolocation API and the assumption that the second location check is performed from a non-compromised device.

6.3 Challenges Encountered

During the development of the multifactor authentication system, several technical and implementation challenges were encountered, particularly due to the system's distributed and security-sensitive nature.

6.3.1 Preventing Link Copying and Sharing

To protect the security model of the system, one of the criteria is ensuring verification links are used only by the intended user, preventing user from forwarding or sharing it. Several strategies are studied and tested to reduce the risk. However, due to the nature of how email works, no current technologies provide absolute prevention on copying. Therefore, warnings against link sharing and HTML form with properties against copying is implemented to reduce the risk, the nature of email still makes complete prevention impossible.

6.3.2 Cross-Origin Resource Sharing (CORS) policy

Since the backend API is a public service and used by websites hosted on different domains, strict CORS policies were required to prevent misuse and other risks. Therefore, the API key mechanism is implemented for the backend to dynamically allow requests only from domains that had completed the verification process.

6.3.3 Secure Domain Verification

Implementing domain verification posed both security and compatibility challenges. It had to be designed in a way that supported a wide range of hosting platforms, including GitLab Pages, Netlify, and Vercel. As discussed in section 4.1.5.3, different approaches are looked into and based on several factors: security, ease of use, and compatibility with cloud-hosted services, the system applies the combination of file upload verification and checking Referrer/Origin headers.

6.3.4 Geolocation Permission and Accuracy

The system initially relied on browser-based HTML5 geolocation to obtain accurate user location. However, challenges arose when users denied location access or when the geolocation API failed. Implementing a fallback using IP-based geolocation was considered, but this approach offered significantly lower accuracy and was not reliable. Communicating these limitations to users and handling such scenarios gracefully within the UI and backend logic added complexity to the implementation.

6.4 Objectives Evaluation

As in Chapter 1.3 Project Objectives, there are a total of 5 objectives defined on this project. This section is to measure the achievement of this project. The evaluation discussion for the 5 objectives is listed below

1. To develop a location-based MFA system that mitigates the risk of RTP attack.

This objective has been partially achieved. The system enforces location-based verification as a second factor during login, making it difficult for attackers to bypass authentication even if user credentials are phished. However, the room for improvements exist.

Although HTML form can increase the difficulty for non-tech-savvy users to share the verification link, it doesn't stop a more tech-savvy victim from forwarding the link. Therefore, although this approach adds a barrier against RTP attacks, its vulnerabilities prevent it from being a fully robust solution.

2. To design a user-friendly interface and minimize the steps of authentication, increasing the system's adaptability and ensuring a smooth transition.

The frontend design emphasizes usability by integrating clear feedback, minimal user inputs, and visual cues. Websocket is used for realtime communication between frontend and backend, minimize the time waiting for verification to proceed. Besides, this authentication can be used for any browser-compatible device.

3. To build a backend API that facilitates the integration of location-based multi-factor authentication to a system.

This objective is achieved as demonstrated in Chapter 6. The implemented API is provides feedback in a clear, flexible way as described in Chapter 5.5. Access to protected API endpoints is protected with multiple security measures such as domain verification, Origin or Referer checking. These has made the API integrate with any system easily in a secure way.

4. To incorporate extra features to counter brute-force attempts.

This objective is fully achieved as the system prevents brute-force attacks through multiple security measures: rate-limiting, account lockout mechanism, login attempt and monitoring of login attempts.

5. To evaluate the system's security and usability.

An evaluation was performed against common attack such as phishing, GPS spoofing, man-in-the-middle, and replay attacks in Chapter 6. To summarise, the system shows strong resilience in replay, brute-force and enumeration attack. However, for RTP, MITM, GPS spoofing attacks, there are identified vulnerabilities. Usability was evaluated based on interface design and the reduction of steps. The system provides an effective and user-centric experience. Overall, the system demonstrates a balance between security and usability.

6.5 Concluding Remark

In section 6.1, different test cases were tested, and it is confirmed that the functionalities match expected outcomes. Besides, security's resilience against multiple attacks is analysed in Section 6.2. The system increases barrier and multi-factor authentication's user-experience by step minimization for RTP attack. However, due to the nature of email, vulnerability regarding user-controlled link sharing exists. Effective mechanism against replay, enumeration and brute-force attacks is implemented. The future enhancements for mitigating MITM attacks and device theft which the system falls short will be discussed in detail in Future Work section.

the "Challenges Encountered" section provides valuable insights into the practical difficulties of implementing a secure and user-friendly system, particularly regarding email link security and geolocation accuracy. Lastly, the objective evaluation

CHAPTER 7

CONCLUSION

7.1 Conclusion

Phishing attack, especially RTP attack is threatening the users' online security This project develops a location-based multi-factor authentication system that enhances security while maintaining user-friendliness. By integrating traditional username/password authentication with location-based verification, the project has demonstrated the potential to improve the security of modern web applications using user location.

While building the project on the MERN stack, Agile iterative development principles were applied. As a result, the system's modularity, security and maintainability is ensured. The key security features like secure token generation, WebSocket-based one-time location verification, and CORS enforcement were implemented to protect against common attacks. The frontend is deployed on cloud via platforms Vercel and backend via Render.

Chapters 4 and 5 highlight the functional design and actual implementation of core features including registration with email verification, login with location validation, dashboard for API key management, and logging and rate limiting. Some considerations about domain verification methods also discussed in section 4.1.5.3. The database schema and the description of each schema's field is also discussed in the last section of Chapter 4.

Finally, system testing is conducted and reported in Chapter 6. It is confirmed that the key functionalities work as intended and effectively mitigate various attack scenarios, especially brute-force, GPS spoofing and RTP phishing attempts.

However, the project also encountered several challenges and limitations. Collecting and using location data may raise privacy concern and need to be improved by cryptographically prove user's location without revealing it to backend. This future work may help mitigate GPS spoofing attack as well. Additionally, although some methods are used to prevent verification URL is being shared by non-technical users, it is impossible to stop the sensitive link from being shared completely, more mechanism such as device fingerprinting may need to be investigated in near future.

Despite these challenges, the project makes a valuable contribution to the field of cybersecurity. It demonstrates how the combination of location-based authentication can mitigate RTP attack to create a more secure and user-friendly authentication system.

7.2 Future Work

To further enhance the security and robustness of the system, there are several improvements can be explored in future work. First, while HTTPS is currently enforced on the cloud platform, it is crucial to ensure Transport Layer Security (TLS) is applied to all communication channels. For example, the secure protocol (WSS) should be fully implemented on WebSocket communication with SSL certificates.

Furthermore, to improve user trust on the system and align with privacy policy, privacy-preserving location verification techniques can be studied. Potential works include zero-knowledge proofs which use cryptographic method to prove location presence without revealing user's exact coordinates.

As Artificial Intelligence(AI) technologies advance, machine learning-based treat detection and monitoring system can be implemented as an extension of system's logging and monitoring feature. For example, it can be used to detect the use of VPNs and proxies to secure the system against GPS spoofing attack.

Device fingerprinting could be introduced as an additional authentication layer by generating trust scores based on the device's past authentication behaviour. This would allow the system to recognize familiar devices and flag potentially compromised or new devices for further verification.

In summary, enforcing TLS in all communications, researching privacy-preserving methods for location, integrating machine learning in anomaly detection, enhancing GPS spoofing detection and introducing device fingerprinting can be significantly strengthen the security model of the location-based authentication system

REFERENCES

[1] R. Brown, “South Carolina Offers Details of Data Theft and Warns It Could Happen Elsewhere,” *The New York Times*, Nov. 20, 2012. Available: https://www.nytimes.com/2012/11/21/us/more-details-of-south-carolina-hacking-episode.html. [Accessed: Apr. 18, 2024]

[2] N. Kshetri, “Why the IRS was just hacked – again – and what the feds can do about it,” *The Conversation*, Feb. 16, 2016. Available: https://theconversation.com/why-the-irs-was-just-hacked-again-and-what-the-feds-can-do-about-it-54524. [Accessed: Apr. 18, 2024]

[3] S. Keller, “Identity Fraud Hits Record High with 15.4 Million U.S. Victims in 2016, Up 16 Percent According to New Javelin Strategy & Research Study,” *Javelin*, Feb. 01, 2017. Available: https://www.javelinstrategy.com/press-release/identity-fraud-hits-record-high-154-million-us-victims-2016-16-percent-according-new. [Accessed: Apr. 18, 2024]

[4] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan, “Phishing Attacks: A Recent Comprehensive Study and a New Anatomy,” *Frontiers*, vol. 3, Mar. 2021, doi: https://doi.org/10.3389/fcomp.2021.563060. Available: https://www.frontiersin.org/articles/10.3389/fcomp.2021.563060/full. [Accessed: Apr. 18, 2024]

[5] M. Jakobsson, “Two-factor inauthentication – the rise in SMS phishing attacks,” *Computer Fraud & Security*, vol. 2018, no. 6, pp. 6–8, Jun. 2018, doi: https://doi.org/10.1016/s1361-3723(18)30052-6

[6] U. A. Abdurrahman, M. Kaiiali, and J. Muhammad, “A new mobile-based multi-factor authentication scheme using pre-shared number, GPS location and time stamp,” in *2013 International Conference on Electronics, Computer and Computation (ICECCO)*, IEEE Xplore, Jan. 2014, pp. 293–296. Available: https://ieeexplore.ieee.org/document/6718286. [Accessed: Apr. 21, 2024]

[7] M. N. Aman, M. H. Basheer, and B. Sikdar, “Two-Factor Authentication for IoT With Location Information,” *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3335–3351, Apr. 2019, doi: https://doi.org/10.1109/jiot.2018.2882610

[8] M. Bartlomiejczyk, I. E. Fray, M. Kurkowski, S. Szymoniak, and O. Siedlecka-Lamch, “User Authentication Protocol Based on the Location Factor for a Mobile Environment,” *IEEE Access*, vol. 10, pp. 16439–16455, 2022, doi:

https://doi.org/10.1109/access.2022.3148537

[9] N. Chhetri, "A Comparative Analysis of Node.js (Server-Side JavaScript)," *Culminating Projects in Computer Science and Information Technology*, Mar. 2016, Available: https://repository.stcloudstate.edu/csit_etds/5/. [Accessed: Apr. 18, 2024]

[10] K. Lei, Y. Ma, and Z. Tan, "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js," *IEEE Xplore*, pp. 661–668, Jan. 2015, doi: https://doi.org/10.1109/CSE.2014.142. Available: https://ieeexplore.ieee.org/abstract/document/7023652. [Accessed: Apr. 18, 2024]

[11] OpenJS Foundation, "Express - Node.js web application framework," *Expressjs.com*, 2017. Available: https://expressjs.com/. [Accessed: Apr. 18, 2024]

[12] "Firebase Realtime Database | Store and sync data in real time," *Firebase*. Available: https://firebase.google.com/products/realtime-database#:~:text=The%20Firebase%20Realtime%20Database%20is. [Accessed: Apr. 18, 2024]

[13] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," *JSON Web Token (JWT)*, May 2015, doi: https://doi.org/10.17487/rfc7519

[14] K. Shingala, "JSON Web Token (JWT) based client authentication in Message Queuing Telemetry Transport （MQTT)," *ResearchGate*, Mar. 2019. Available: https://www.researchgate.net/publication/331587509_JSON_Web_Token_JWT_based_client_authentication_in_Message_Queuing_Telemetry_Transport_MQTT. [Accessed: Apr. 18, 2024]

[15] Collections, Mansfield Library Archives and Special. "Research Guides: Maps Research Guide: How to Use Maps." *Libguides.lib.umt.edu*, libguides.lib.umt.edu/c.php?g=275290&p=6870295. Accessed 5 May 2025.

[16] Luse, Andy, and Jim Burkman. "Gophish: Implementing a Real-World Phishing Exercise to Teach Social Engineering." *Research and Practice Journal of Cybersecurity Education, Research and Practice*, vol. 2020, no. 2, 2021, digitalcommons.kennesaw.edu/cgi/viewcontent.cgi?article=1072&context=jcerp. Accessed 5 May 2025.

[17] “Password Requirements.” *Kaspersky.com*, 2024, support.kaspersky.com/KPC/1.0/en-US/183862.htm.

[18] Sun, Yuanyi, et al. “Let Your Camera See for You: A Novel Two-Factor Authentication Method against Real-Time Phishing Attacks.” *ResearchGate*, 31 Aug. 2021, www.researchgate.net/publication/354310380_Let_Your_Camera_See_for_You_A_Novel_Two-Factor_Authentication_Method_against_Real-Time_Phishing_Attacks. Accessed 5 May 2025.

[19] “Cross-Origin Resource Sharing (CORS) - HTTP | MDN,” *MDN Web Docs*, Mar. 14, 2025. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS

Appendix A

CODE SNIPPET FOR IMPLEMENTATION

A1 Location Retrieval

```
const GeolocationComponent = () => {
  const [location, setLocation] = useState({ latitude: null, longitude: null });
  const [error, setError] = useState(null);

  const handleGetLocation = () => {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(
        (position) => {
          setLocation({
            latitude: position.coords.latitude,
            longitude: position.coords.longitude,
          });
          setError(null);
        },
        (err) => {
          setError('Unable to retrieve location.');
          console.error(err);
        }
      );
    } else {
      setError('Geolocation is not supported by this browser.');
    }
  };

  return (
    <div>
      <h2>Geolocation</h2>
      <button onClick={handleGetLocation}>Get Location</button>
      {location.latitude && location.longitude ? (
        <div>
          <p>Latitude: {location.latitude}</p>
          <p>Longitude: {location.longitude}</p>
      ) : (
        <p>No location data available.</p>
      )}
      {error && <p style={{ color: 'red' }}>{error}</p>}
    </div>
  );
};
```

A2 Register

A2.1 Register Function

```
const register = async (req, res) => {
  try {
    const { email, password } = req.body;

    // Input validation
    if (!email || !password) {
      return res.status(400).json({
        message: 'Email and password are required.',
      });
    }

    // Check for existing user
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(200).json({
        message: 'If this email exists, you\'ll receive a confirmation link.',
      });
    }

    // Generate hashed password and unique username
    const hashedPassword = await bcrypt.hash(password, 10);
    const username = generateUsername(email);
    const urlKey = generateToken(32);

    // Save to PendingUser
    const pendingUser = new PendingUser({
      email,
      username,
      password: hashedPassword,
urlKey,
    });

    await pendingUser.save();

    const domain = req.website?.domain;
    const verificationURL = `${domain}/verify-email?key=${urlKey}`;

    // Send verification email
    try {
      await EmailService.sendEmailVerification(email, domain, verificationURL);
    } catch (emailError) {
      errConsoleLog('Email send error:', emailError);
      return res.status(200).json({
        message: 'If this email exists, you\'ll receive a confirmation link.',
      });
    }

    return res.status(201).json({
      message: 'If this email exists, you\'ll receive a confirmation link.',
    });

  } catch (error) {
    errConsoleLog('Registration error:', error);
    return res.status(500).json({
      code: 'SERVER_ERROR',
      message: 'An error occurred during registration.',
    });
  }
};
```

A3 Login

A3.1 Login function

```
const login = async (req, res) => {
    const start = Date.now();
    const { email, password, latitude, longitude } = req.body;

    let user = null;
    let session = null;
    let attempt = null;

    try {
        user = await User.findOne({ email });

        if (!user) {
            attempt = new LoginAttempts({
                websiteId: req.website._id,
                status: 'failure',
                ipAddress: req.ip,
                start: start,
                userAgent: req.headers['user-agent'],
                reasonOfFailure: 'user_not_found',
            });

            await setGetSchema.createLoginAttempts(attempt);
            return res.status(400).json({ message: 'Invalid credentials' });
        }
        attempt = new LoginAttempts({
            userId: user._id,
            websiteId: req.website._id,
            status: 'failure',
            ipAddress: req.ip,
start: start,
            userAgent: req.headers['user-agent'],
            reasonOfFailure: 'other',
        });

        if (!(await bcrypt.compare(password, user.password))) {
            attempt.reasonOfFailure = 'invalid_credentials';
            await setGetSchema.createLoginAttempts(attempt);
            await handleFailedLogin(user);
            return res.status(400).json({ message: 'Invalid credentials' });
        }

        if (!latitude || !longitude) {
            attempt.reasonOfFailure = 'invalid_location';
            await setGetSchema.createLoginAttempts(attempt);
            await handleFailedLogin(user);
            return res.status(400).json({ message: 'Invalid location data' });
        }
        const verifyToken = generateToken(32);
        session = await setGetSchema.createSession(generateToken(16), user._id, req.website._id);
        const token = await setGetSchema.createVerifyToken(verifyToken, latitude, longitude,
session.sessionToken);

        const verificationUrl = `${req.website.domain}verify-location`;

        session.verifyTokenId = token._id;
        attempt.sessionId = session._id;
        attempt.status = "pending";
        await setGetSchema.createLoginAttempts(attempt);

        EmailService.sendLocationVerification(user.email, req.website?.domain, verificationUrl, verifyToken,
attempt);
```

```
await resetFailedAttempts(user);


        return res.status(200).json({
            message: 'Verification email sent. Please check your mailbox. If you don\'t see the email, kindly try again.',
            verifyToken
        });
    } catch (error) {
        console.error("Login error:", error);


        if (attempt) {
            attempt.reasonOfFailure = 'other';
            await setGetSchema.createLoginAttempts(attempt);
        }


        return res.status(500).json({ message: 'An error occurred during login. Please try again later.' });
    }
};
```

A3.2 Verify Location Function

```
const verifyLocation = async (req, res, io) => {
    const start = new Date();
    let attempt;
    let session;
    let token;
    let user;
    try {
        const lat2 = req.body.latitude;
        const lon2 = req.body.longitude;
        const verifyToken = req.body.key;

        if (!lat2 || !lon2 || !verifyToken) {
            return res.status(400).json({
                code: 'MISSING_FIELDS',
                message: 'Missing required fields: latitude, longitude, or key',
            });
        }

        token = await VerifyToken.findOneAndDelete({ verifyToken });
        session = await setGetSchema.getSession(null, verifyToken, "pending");

        if (!token || !session) {
            attempt = new LoginAttempts({
                websiteId: req.website._id,
                status: 'failure',
                ipAddress: req.ip,
                start: start,
                userAgent: req.headers['user-agent'],
                reasonOfFailure: 'invalid_session'
```

A3.3 Generate User's JWT Token Function

```
function generateJwtToken(userId, domain) {
  const jwtToken = jwt.sign(
    { userId },
    process.env.JWT_SECRET,
    {
      expiresIn: '1h',  // Short expiration for security
      algorithm: 'HS256',  // Secure hashing algorithm (use RS256 for asymmetric encryption)
      issuer: 'multifactor.com',  // Set issuer
      audience: domain  // Ensure token is only used for the correct website
    }
  );
  return jwtToken;
}
```

A4 Logging and Rate Limiting

A4.1 Create LoginAttempt in Database Function

```
async function createLoginAttempts(attemptOrUserId, websiteId = null, sessionId = null, status = null,
ipAddress = null, start = null,
    userAgent = null, reasonOfFailure = null) {
    try {
        let attempt = null;
        let responseTime = null;
        let session = null;

        // If the first parameter is an object, use it directly
        if (attemptOrUserId instanceof LoginAttempts && attemptOrUserId !== null) {
            attempt = attemptOrUserId;
            websiteId = attempt.websiteId;
            sessionId = attempt.sessionId;
            status = attempt.status;
            ipAddress = attempt.ipAddress;
            start = attempt.start;
            userAgent = attempt.userAgent;
            reasonOfFailure = attempt.reasonOfFailure;
        } else {
            // If sessionToken is provided, fetch session
            if (sessionId) {
                session = await UserSession.findById(sessionId);
            }
            if (session) {
                attempt = await LoginAttempts.findOne({ websiteId, sessionId: session._id }).lean(false);
            }
            // Calculate responseTime
            responseTime = attempt ? (attempt.timestamp ? Date.now() - attempt.timestamp : null) : (start ?
Date.now() - start : null);


            if (attempt) {
                attempt.status = status ?? attempt.status;
                attempt.responseTime = responseTime ?? attempt.responseTime;
                attempt.reasonOfFailure = status === 'failure' && reasonOfFailure ? reasonOfFailure :
attempt.reasonOfFailure;
            } else {
                attempt = new LoginAttempts({
                    userId: attemptOrUserId,
                    websiteId,
                    sessionId: session ? session._id : null,
                    status,
                    ipAddress,
                    timestamp: new Date(),
                    responseTime,
                    userAgent,
                    reasonOfFailure: status === 'failure' ? reasonOfFailure : null,
                });
            }
        }
        if (!(attempt instanceof LoginAttempts)) {
        throw new Error("Login attempt is not a valid Mongoose document.");
        }
        await attempt.save();
        return attempt;
    } catch (error) {
        errConsoleLog(error.message);
        throw new Error(error.message || "Unknown error");
    }
}
```

A4.2 IP-Based and Account-Based Rate-Limiting Function

```
const rateLimiter = rateLimit({
    windowMs: IP_WINDOW_MS,
    max: IP_MAX_ATTEMPTS,
    message: { message: `Too many login attempts from this IP address. Try again in ${IP_WINDOW_MS /
60000} minutes.` },
    headers: true,
    keyGenerator: (req) => req.ip,
    handler: async (req, res) => {
        console.warn(`❌ Rate limit exceeded for IP: ${req.ip}`);


        // Log the rate limit violation
        await LoginAttempts.create({
            userId: null,
            status: 'failure',
            ipAddress: req.ip,
            timestamp: new Date(),
            sessionId: null,
            websiteId: req.website?._id || null,
            userAgent: req.headers['user-agent'],
            reasonOfFailure: 'rate_limit_exceeded'
        });
        if (req.body.email) {
            const user = await User.findOne({ email: req.body.email });
            if (user) {
                user.failedLoginAttempts += 1;
                if (user.failedLoginAttempts >= 5) {
                    user.lockUntil = Date.now() + 15 * 60 * 1000;
                }
                await user.save();
            }
```

A4.3 Reset Locked Account

```
const resetFailedAttempts = async (user) => {
  if (!user) return;


  user.failedLoginAttempts = 0;
  user.lockUntil = null;
  user.lastLogin = new Date();
  await user.save();
};
```

A.5 Dashboard and API Management

```
const verifyApiKey = async (apiKey, origin) => {
  try{
    if (!apiKey) {
      errConsoleLog("Request missing API key");
      throw new Error ("Missing API key");
    }
    const website = await Website.findOne({ apiKey });
    if (!website) {
      errConsoleLog("API key not found in DB");
      throw new Error ("Invalid API key");
    }
    if (!website.active) {
      errConsoleLog("API key found but inactive");
      throw new Error ("Inactive API");
    }
    if (origin) {
      const originDomain = new URL(origin).origin;
      const trustedDomain = new URL(website.domain).origin;

      if (originDomain !== trustedDomain) {
        errConsoleLog(`Origin mismatch: expected ${trustedDomain}, got ${originDomain}`);
        throw new Error("Invalid request origin");
      }
    } else {
      errConsoleLog("Missing Origin/Referer header");
      throw new Error("Missing request origin");
    }
    return website;
  }catch(error) {
```

POSTER

Phishing-Resistant multi factor authentication

INTRODUCTION

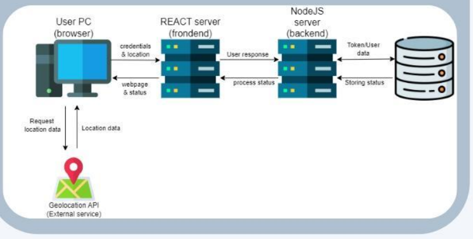
Traditional 2FA combines authentication factors to overlay a security layer on user account, however, it is still susceptible to attacks realtime phishing attacks and website mirroring attacks. Location-based multi authentication (MFA) methods have been proposed to enhance current authentication system

OBJECTIVES

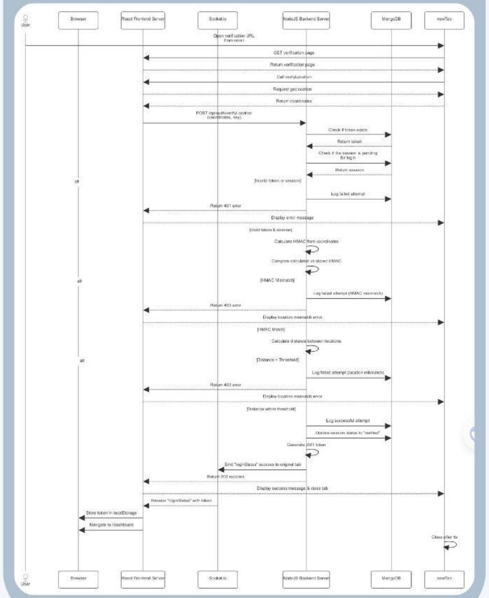
SUMMARY

1. To develop a location-based MFA system that mitigates the risk of phishing attack.
2. To design a user-friendly interface for the system.
3. To build a backend API for ease of integration.
4. To integrate additional features against brute-force attempts
5. To evaluate the system's security and usability

COMPONENTS



LOCATION VERIFICATION


User
Browser
React Frontend Server
Socket.io
NodeJS Backend Server
MongoDB
newTab
Open verification URL from email
GET verification page
Return verification page
Call verifyLocation
Request geolocation
Return coordinates
POST /api/auth/verifyLocation (coordinates, key)
Check if token exists
Return token
Check if the session is pending for login
Return session
alt
[Invalid token or session]
Log failed attempt
Return 401 error
Display error message
[Valid token & session]
Calculate HMAC from coordinates
Compare calculated vs stored HMAC
alt
[HMAC Mismatch]
Log failed attempt (HMAC mismatch)
Return 403 error
Display location mismatch error
[HMAC Match]
Calculate distance between locations
alt
[Distance > Threshold]
Log failed attempt (location mismatch)
Return 403 error
Display location mismatch error
[Distance within threshold]
Log successful attempt
Update session status to "verified"
Generate JWT token
Emit "loginStatus" success to original tab
Return 200 success
Display success message & close tab
Receive "loginStatus" with token
Store token in localStorage
Navigate to /dashboard
Close after 3s

TECH & TOOLS

1. MERN stack (MongoDB, ExpressJS, React, NodeJS)
2. HTML 5 Geolocation API
3. Vercel & Render
4. Postman
5. Socket

Kuek En Yee
Guided by Dr Gan Ming Lee
Bachelor of Information Technology (Hons) Communications and Networking