Secure Authentication System based on Multi-Factor Authentication

PROJECT REPORT

Submitted in the partial fulfillment of the requirements for

award of the

Six Months Online Certificate Course

in

Cyber Security/

**Data Science with Python Programming/
Artificial Intelligence and Machine
Learning**

Course Duration: [22-01-2024 to 21-07-2024]

By

PASALA NAGADEVI

(HT No.2406CYS112

)

Under the Esteemed Guidance

Prof.B SATEESH KUMAR,

Professor & HOD, Department of

CSE, JNTUH University College of

Engineering , Jagityal

**DIRECTORATE OF INNOVATIVE LEARNING &
TEACHING (DILT)**

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

(Formerly SCDE_SCHOOL OF CONTINUING AND DISTANCE EDUCATION)

Kukatpally, Hyderabad, Telangana State, INDIA- 500 085

JULY 2024

ABSTRACT

In today's digital landscape, access to various online platforms has become ubiquitous, and so has the risk associated with information security. Traditional username and password-based authentication systems have become increasingly vulnerable to brute force attacks. The proposed project, "Secure Authentication System Based on Multi-Factor Authentication (MFA)," aims to address this pressing concern. MFA, which requires users to provide multiple forms of authentication such as a combination of knowledge-based factors (e.g. One-Time Passwords) and inherence-based factors (e.g. biometrics), significantly enhances the security of user accounts and digital assets. The key objectives of this project are to enhance security, reduce unauthorized access, prevent data breaches, improve user trust, mitigate credential stuffing attacks, comply with regulations, provide flexibility, enable secure remote access, and continuously improve security. To achieve these objectives, the project will leverage a combination of technologies and methods like One-Time Passwords (OTP), Biometric Authentication, Security Tokens, and Push Notifications. Each of these technologies contributes uniquely to the security of the system. For instance, OTPs provide a dynamic and time-sensitive second layer of security, while biometric authentication offers a highly personalized and difficult-to-replicate third layer. In designing and implementing this secure authentication system, the project will prioritize user experience, scalability, and compliance with security best practices and regulations. The system will be developed using Python, MySQL database, and Django framework. The aim is to design a system that is not only secure but also user-friendly and scalable, capable of adapting to growing user bases and evolving security threats. Implementing such a system is critical for enhancing the security of user accounts and sensitive data, as MFA adds an extra layer of protection by necessitating multiple forms of verification before granting access.

The proposed system consists of three main modules; the first module is the sign up module where the user registers his main information and password and email id regarding predefined conditions. The second module is the login module which is used for password comparison. The third module is responsible for generating QR code for synchronizing the generated OTP between the application server and authenticator (on a mobile device). There is an additional module for generating the TOTP.

After successful registration (sign up), the QR code is generated regarding the generated OTP in our system. The user can use an authenticator application on his mobile device, such as Google Authenticator, for scanning the QR code and obtaining the synchronized OTP code. For the authentication process, if the user enters the valid password for login, the system will forward him to the OTP verification page to enter the valid OTP code displayed on the authenticator application and after that user should enter the valid OTP received in their mail id The results demonstrate the effectiveness of our proposed system.

Keywords: password, OTP, QR code, multi-factor authentication.

TABLE OF CONTENTS

Chapter : 1 Introduction

Authentication is the core of cybersecurity, and it is used to determine who we are allowing access into our systems, applications or networks. It is the process of verifying who a user claims to be; that he or she says, in fact -is associated with what type. Authentication is one of the primary components in order to create trust and securing sensitive data keeping unwanted hands away from accessing digital resources. This process is usually also combined with authorization, which means that after we have authenticated users (know who they are), we can then decide what rights to give them once we know their identity. Organizations can enforce security policies and safely guard sensitive data in authentication systems, mitigating the risk of unauthorized access and guarding itself from possible threaten actors through rock-solid login processes. Most used method of authentication is using passwords, and we find this everywhere across all the services.

MFA is the process of combining different authentication factors to identify a system's user. These authentication factors are typically something one knows (text password), something one is (biometric ex: fingerprint), and something one has (Smart card, authentication token). The commonly known Two Factor Authentication (2FA) typically includes two factors from separate categories, while MFA can include several factors from the same category Authentication is proving that a person is whom they claim to be .This is distinctly different from identification, being "the act of asserting who a person is". These two concepts are often mistaken. The way that authentication is done is by a person providing a factor. This is common to access a system or resource. There are authentication mechanisms that use only one factor. Many will recognize the most common factor for single authentication mechanisms being a password. It is not uncommon for access to resources to be placed behind this simple authentication mechanism. However, this single point of failure has known vulnerabilities. The vulnerabilities are becoming more complex as the types of attacks do as well . Among the many challenges that exist with text-based passwords, some of the most common are related to the memorization of quality passwords. One study found that 80% of passwords were able to be cracked by the average personal computer in less than a week. This concept is relevant as computing power increases over time and is due to low complexity passwords and a lack of understanding from end users. As participants of the same study appeared "to be unconcerned about the risks associated with poor password composition". Moreover, 40% of users had never changed their passwords. This has compounded in recent years due to the increase in the number of services that require passwords, reusing passwords, and using passwords with guessable openly known content such as a spouse's name. This concept contributed to the development of adding more factors for authentication. First, there was Two Factor Authentication, then Multi-Factor Authentication to ensure a more secure authentication of a user. Authenticating users has a deep involvement in the field of cyber security because impersonation can pass through authentication mechanisms and typically gives access to protected resources. In general, this is the foundation of cyber security, and often the first line of defense. Authentication mechanisms are typically classified by how many factors they use and the categories of the factors. The types of factors can be categorized into one of the following groups. Something one knows (Knowledge factor), something one has (Possession factor), and something one is (inherence or biological factor). Examples of knowledge factors would be things like PIN numbers, text passwords, lock patterns, graphical passwords, challenge-response, etc. Examples of possession factors would be: ID cards, NFC, RFID, smart cards, hardware tokens, physical keys, or a device itself. Examples of biological factors would be: facial recognition, iris, fingerprints, or even other physical characteristics such as typing speed and speech recognition. This is the most well-known way of categorizing authentication mechanisms. Categorization also assists in defining whether a mechanism is 2FA or MFA. Commonly a 2FA mechanism has two different factors from each of the three groups, and MFA has more than two and can have several from the same group. It is not uncommon for other factors to exist and not fit into the "textbook" definition of these categories, such as location-based authentication. This is not technically a knowledge, possession, or biological factor. 2FA and MFA, however, have not solved the challenges faced with authentication. End-users have negative connotations with MFA for a variety of reasons. Adoption rates for MFA concerning websites have stagnated. A 2018 study showed that 2FA over a growing set of websites was close to 50% for over four years from 2014 to 2018. Both the end-users and services providing authentication options have impacted the adoption of MFA. MFA systems themselves are still vulnerable to attacks. Although the types of attacks are numerous, familiar and relevant examples are: Leaks from a server, phishing, theft, man-in-the-middle, social engineering, shoulder surfing, keyloggers, spyware, and other types of malware/Trojans. Despite this, it is still a safe assumption that MFA and 2FA are more secure than single-factor systems. This is because the more factors that correlate to a user, the more confidently one can assure the identity of a person.

Chapter 2: Literature survey

2.1 Introduction

The literature review centers on two main facets: the system perspective and the methodological standpoint. Regarding the system aspect, it delves into the presently utilized tools, highlighting their limitations. On the methodological front, it explores alternative approaches proposed by various researchers.

2.2 Background

Authentication is crucial when individuals access systems to confirm their identity and safeguard their private data. Consequently, various authentication techniques, incorporating elements such as knowledge, ownership, and OTPs, have been proposed. In logins, authentication is vital to allow people into systems so that they can prove who they are and help protect their data from being seen by others. Therefore different authentication methods have been introduced including using something you know, own and 1-time passwords (OTPs).

It allows a single authentication process for device access, which is a knowledge based authorization. The main method of user authentication and access control among all internet banking systems is the use of knowledge attributes, so it seems correct taking into consideration both what technology we trust and how familiar are most users with given solutions. However, knowledge-based authentication is exposed to a wide range of attack types: brute force attacks; rainbow table attacks; dictionary or wordlist based password-attacks and combinations thereof along with social engineering + phishing techniques (accordingly frameworks for man-in-the-middle MITMF have been developed); password-based assaults through session hijacking(s) as well malware. Further, it was recognised that a reliance solely on password-based authentication; User login studies have established that an incredible 86% of user selected passwords are weak. In the current security scenario that is replete with shortcomings concerning KBA, a secure solution like Ownership-Based Authentication evolved to substantially boost authentication and verification of online transactions for devices as well as applications.

Property authentication, which often needs security hardware tokens, is usually used for two factor systems. Returning an element could be used as a secondary form of validation but also the primary authentication mechanism. This role is where you should not check authentication depending solely on who owns the given partitions. This is true for even bad passwords as a weak password combined with possession based authentication still adds hurdles that potential attackers must take to gain access additionally

2.3 Related work

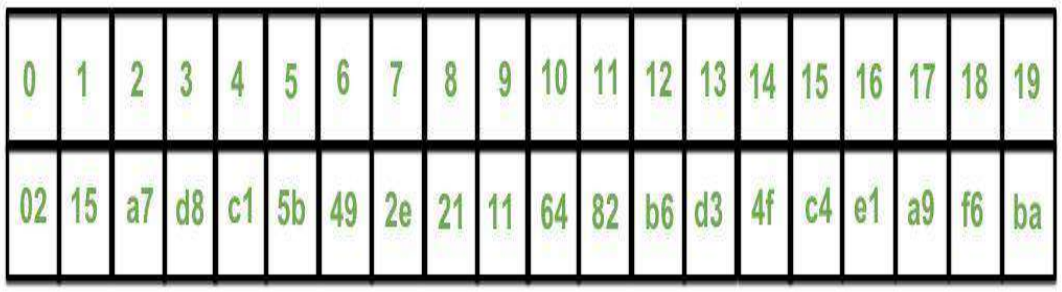
Authentication mechanisms today create a double layer gateway prior to unlocking any protected information. This double layer of security, termed as two factor authentication, creates a pathway that requires validation of credentials (username/email and password) followed by creation and validation of the One Time Password (OTP). The OTP is a numeric code that is randomly and uniquely generated during each authentication event. This adds an additional layer of security, as the password generated is a fresh set of digits each time an authentication is attempted and it offers the quality of being unpredictable for the next created session. The two main methods for delivery of the OTP is:

1. SMS Based: This is quite straightforward. It is the standard procedure for delivering the OTP via a text message after regular authentication is successful. Here, the OTP is generated on the server side and delivered to the authenticator via text message. It is the most common method of OTP delivery that is encountered across services.

2. Application Based: This method of OTP generation is done on the user side using a specific smartphone application that scans a QR code on the screen. The application is responsible for the unique OTP digits. This reduces wait time for the OTP as well as reduces security risk as compared to the SMS based delivery.

The most common way for the generation of OTP defined by The Initiative for Open Authentication (OATH) is the Time Based One Time Passwords (TOTP), which is a Time Synchronized OTP. In these OTP systems, time is the cardinal factor to generate the unique password. The password generated is created using the current time and it also factors in a secret key. An example of this OTP generation is the Time Based OTP Algorithm (TOTP) described as follows:

1. Backend server generates the secret key
2. The server shares secret key with the service generating the OTP
3. A hash based message authentication code (HMAC) is generated using the obtained secret key and time. This is done using the cryptographic SHA-1 algorithm. Since both the server and the device requesting the OTP have access to time, which is obviously dynamic, it is taken as a parameter in the algorithm. Here, the Unix timestamp is considered which is independent of time zone i.e. time is calculated in seconds starting from January First 1970. Let us consider "0215a7d8c15b492e21116482b6d34fc4e1a9f6ba" as the generated string from the HMAC-SHA1 algorithm.
4. The code generated is 20 bytes long and is thus truncated to the desired length suitable for the user to enter. Here dynamic truncation is used. For the 20-byte code "0215a7d8c15b492e21116482b6d34fc4e1a9f6ba", each character occupies 4 bits. The entire string is taken as 20 individual one byte strings.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
02	15	a7	d8	c1	5b	49	2e	21	11	64	82	b6	d3	4f	c4	e1	a9	f6	ba

We look at the last character, here a. The decimal value of which is taken to determine the offset from which to begin truncation. Starting from the offset value, 10 the next 31 bits are read to obtain the string "6482b6d3″. The last thing left to do, is to take our hexadecimal numerical value, and convert it to decimal, which gives 1686288083. All we need now are the last desired length of OTP digits of the obtained decimal string, zero-padded if necessary. This is easily accomplished by taking the decimal string, modulo 10 ^ number of digits required in OTP. We end up with "288083" as our TOTP code.

5. A counter is used to keep track of the time elapsed and generate a new code after a set interval of time
6. OTP generated is delivered to the user by the methods described above.

Apart from the time-based method described above, there also exist certain mathematical algorithms for OTP generation for example a one-way function that creates a subsequent OTP from the previously created OTP. The two factor authentication system is an effective strategy that exploits the authentication principles of "something that you know" and "something that you have". The dynamic nature of the latter principle implemented by the One Time Password Algorithm is crucial to security and offers an effective layer of protection against malicious attackers. The unpredictability of the OTP presents a hindrance in peeling off the layers that this method of cryptography has to offer.

2.4 Summary

The literature review provides a comprehensive overview of authentication methods with a focus on knowledge-based and ownership-based authentication and biometrics. It discusses the importance of authentication in confirming individuals' identities and protecting their private data.

chapter 3: Problem statement

In the current digital landscape, single-factor authentication methods, such as password-based logins, are vulnerable to various attacks, including phishing, brute force, and dictionary attacks. These security breaches can lead to unauthorized access, identity theft, and data breaches, which can significantly harm users and businesses alike. To address these security concerns, there is a critical need for a more robust authentication system that can ensure the integrity and confidentiality of user data while maintaining accessibility and ease of use for legitimate users. A significant issue solved by multi-factor authentication has to be that passwords are an insecure way of authenticating users. It is clear that passwords are incredibly easy to be stolen, guessed and cracked leaving sensitive information compromised in the process. Attackers seek passwords to compromise access with multiple different methods like phishing, keylogging or social engineering and it makes way for unauthorized good pass into an account/system.

Technical Challenges

Designing a Secure QR Code System: Implementing a QR code system that is secure against potential vulnerabilities, such as QR code hijacking or tampering.

Email OTP Security and Delivery: Ensuring that email OTPs are delivered securely and promptly, without being susceptible to interception or manipulation.

User Interface Design: Developing a user interface that is intuitive and guides users through the multi-factor authentication process without causing frustration or confusion.

Error Handling and Recovery: Implementing robust error handling mechanisms to cope with failed authentication attempts or connectivity issues while maintaining system security.

Performance Optimization: Optimizing the system to handle the additional load of multi-factor authentication without slowing down the login process or affecting the overall application performance.

Solution Scope

The solution should cover the full spectrum of user authentication, from account creation to session management, ensuring that each step is secured by multi-factor authentication. This includes:

User Registration: Incorporating MFA during the account creation process to verify the user's identity.

Login Flow: Implementing a secure and efficient MFA login flow that includes password authentication, followed by QR code and email OTP verification.

Session Management: Ensuring that user sessions are secure and that access control mechanisms are in place to prevent unauthorized access.

By addressing these challenges and objectives, the secure authentication system based on multi-factor authentication using QR codes and email OTPs in Django will provide a comprehensive solution that enhances security without compromising user experience.

Chapter 4 :Objectives

A secure authentication system, particularly one that employs multi-factor authentication (MFA) using QR codes and Email OTPs (One Time Passwords), is a robust approach to enhancing user security in applications developed using Django. The objectives of such a system can be outlined as follows:

1. Enhance Security

*prevent unauthorized access:*By focusing on these objectives, a secure authentication system based on multi-factor authentication using QR codes and email OTPs in a Django application can provide a secure, user-frieEnsure unauthorized accounts: The system reduces the chances of preventing unapproved access by forcing multiple verification procedures. Only an attacker possessing both the user password and access to QR code or email OTP could successfully attack a regular account.

Mitigate Phishing Attempts: QR codes can capture session keys or tokens for transmission and enable the use of a secure channel - all but preventing phishing. Email OTP: To validate user actions, providing more security to unauthorized modifications over user accounts

2 Compliance and Regulatory Standards

Adherence to Industry Standards: Implementing MFA helps in complying with industry standards and regulations that require strong authentication mechanisms, such as those in the financial and healthcare sectors.

Audit and Accountability: The system logs attempts to log in or access services, which can be used for auditing purposes and to monitor compliance with security policies.

3. User Experience and Usability

Convenience for Users: MFA can be designed to be user-friendly by offering multiple options (QR code and email OTP) for second-factor authentication, catering to different user preferences and accessibility.

Reduced Reliance on Passwords: Reducing the dependency on passwords alone can lead to better user experience by alleviating the burden of remembering complex passwords or managing password resets.

4.System Reliability

Redundant Authentication Methods: The presence of multiple authentication methods ensures that if one factor fails or is unavailable, another can be used, maintaining system integrity and availability.

Scalability: The system is designed to handle increased user loads without compromising security or performance, making it suitable for large-scale applications.

5. Integration and Compatibility

Django Framework Integration: Leveraging Django's rich ecosystem and security features ensures compatibility with the existing Django project and can seamlessly integrate with Django's user management and authentication modules.

Third-Party Service Compatibility: The system can be designed to integrate with third-party authentication services, such as Google Authenticator for QR codes or email services for sending OTPs, providing flexibility and robustness in the authentication process.

6. Cost-Effectiveness

Cost Reduction: Implementing MFA with existing mechanisms like email can reduce the cost of additional hardware tokens required for some MFA methods, making it a cost-effective solution for enhancing security.

Prevention of Fraud and Theft: By significantly reducing the risk of account breaches, the system can prevent financial losses and damage to reputation, which can outweigh the initial investment in setting up a MFA system.

By focusing on these objectives, a secure authentication system based on multi-factor authentication using QR codes and email OTPs in a Django application can provide a secure, user-frieEnsure unauthorized accounts: The system reduces the chances of preventing unapproved access by forcing multiple verification procedures. Only an attacker possessing both the user password and access to QR code or email OTP could successfully attack a regular account.

- Prevent Unauthorized Access.

- Mitigate Password-related Risks.
- Safeguard Sensitive Transactions.
- Comply with Regulatory and Compliance Standards.
- Improve User Experience.

Chapter 5 :Methodology

System development encompasses various phases, including planning, development, and maintenance, which are crucial for project success. Here is an extended version of the previous text that includes the maintenance phase:

System development can be approached through various methods, some of which offer more advantages than others. Common system development models include the Waterfall model, Spiral model, and Incremental development. For our application, we have opted for the agile model, a contemporary software development approach that offers several benefits over the traditional Waterfall model. The Agile model is characterized by iterative cycles, as illustrated in Figure 5-1. This model enables developers to break down the software application's development into discrete components. Within each cycle, a specific application component is worked on. Given the project's scope, each agile model cycle comprises five essential steps:

- **Planning:** In the initial planning phase, we define the project scope, objectives, and requirements. This is a crucial step to ensure that the project is well-defined and aligns with the intended goals.
- **Analysis:** During this phase, we gather information and analyze components by identifying the actors and their respective functions.
- **Design:** In this stage, we create the design for the component that is to be developed
- I**mplementation:** This step involves the actual implementation of the component under development.
- **Evaluation Testing:** Here, we conduct thorough testing of the component being developed.

Following the completion of a cycle, we proceed to the maintenance phase, where we ensure that the system operates smoothly and remains up to date with changing requirements and technology. The maintenance phase involves routine updates, bug fixes, and improvements to enhance the system's performance and longevity. It also includes user support and addressing any issues that may arise after deployment.

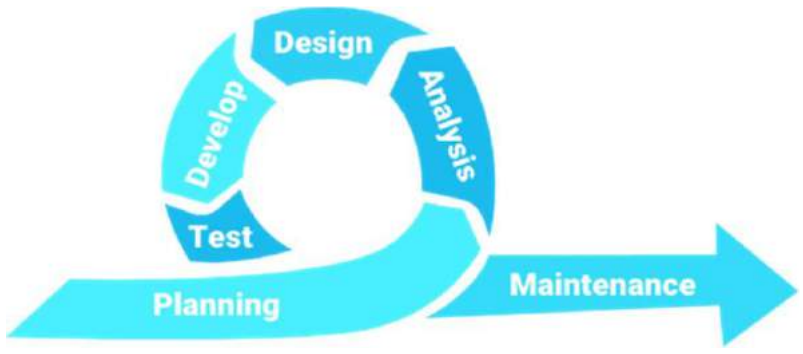


Figure 5-1

chapter 6: System analysis

6.1 Introduction

In this chapter, we conduct a thorough examination of the proposed system, covering a detailed assessment of its feasibility, an exploration of its functional and nonfunctional prerequisites, an overview of the overarching system structure, and an outline of the development approach to be employed for project completion.

6.2 Requirements

This section provides an overview of the project's deliverables' requirements, which can be classified into two distinct categories based on their nature: functional requirements and non- functional requirements.

6.2.1 Functional Requirements

- ***User Registration:*** Users should be able to create new accounts by providing essential information such as username, password, and email address.
- ***Login:*** Users should be able to log in using their registered credentials (e.g., username and password)
- ***Password Policies:*** Enforce password policies, including password complexity requirements (e.g., minimum length, special characters)
- ***One-Time Password (OTP) Generation:*** Implement the generation of OTPs, which are temporary codes, typically numeric, that users can use for authentication.
- ***Token Delivery Methods:*** Provide token delivery using authenticator application and QR code associated with the OTP generator for maintaining the synchronization.
- ***Validate:*** Ensure OTPs are valid fora limited time window to enhance

security.

● ***Email OTP Delivery Methods:*** provide OTP delivered to the registered mail Id.

● ***Validate:*** Ensure OTPs are valid fora limited time window to enhance security.

.

6.2.2 Non-Functional Requirements

● ***Usability:*** The user interfaces for registration, login, and account management should be intuitive and user-friendly.

● ***Resource Efficiency:*** Optimize resource usage (CPU, memory, storage) to minimize costs and reduce the system's environmental impact.

● ***Resource Efficiency:*** Optimize resource usage (CPU, memory, storage) to minimize costs and reduce the system's environmental impact.

● ***Maintainability:*** Design the system in a modular and maintainable way, allowing for updates and enhancements with minimal disruption.

● ***Load Testing:*** Conduct load testing to identify performance bottlenecks and optimize system performance.

6.3 System Analysis

This section focuses on the analysis phase of the proposed system. It describes the process modeling and system user interaction. The following tables, explains the use cases in detail, each includes actors, description, precondition, and event flow.

6.3.1 Flow Chart

Figure 6-1 presents a flow chart depicting the structure of the proposed application program. This visual representation is highly valuable in the process of designing the application program.

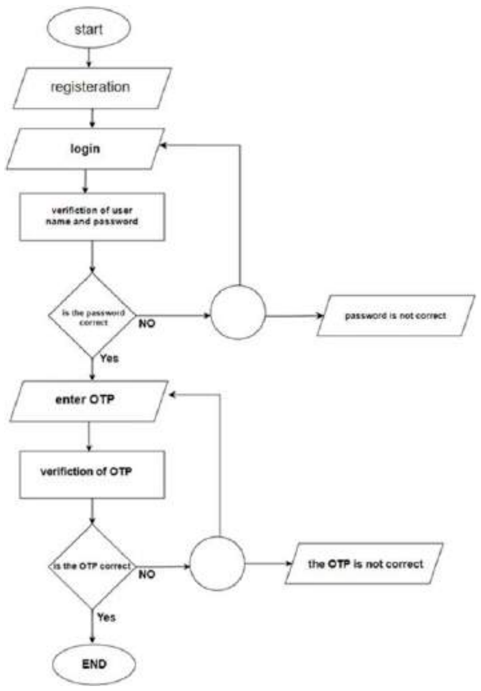


Figure 6-1

The following tables, explains the use cases in detail, each includes actors, description, precondition, and event flow.

Table 6-1 Register

Use case number	UC: 1
Use case name	register
Actors	User
Description	This use case involves the user creating a new account by Providing the necessary information.
Pre-condition	The user must have access to the registration page or Interface.
Scenario flows	1. User accesses the registration page or interface. 2. System presents a form for the user to enter the required details (username, password, email, etc.) 3. User enters the required information. 4. System validates the entered information. 5. If the validation is successful, the system creates a new user account and stores the account details. 6. User account is registered successfully. 7. QR code is now displayed to the user. 8. Scan QR code By using Google Authenticator
Post condition	The user account is successfully registered in the system.

Table 6-2 Generate QR code.

Use case number	UC: 2
Use case name	Generate QR code
Actors	system
Description	Generate QR code containing OTP to user after successful Sign up
Pre-condition	The user must Sign up to display the QR code
Scenario flows	1. The user should scan the generated QR code by using Google authenticator. 2. Now user have a complete synchronize.
Post condition	The user scanned the generated QR code and no need to QR anymore.

Table 6-3Generate OTP

Use case number	UC: 3
Use case name	Generate OTP
Actors	User, System
Description	This use case involves the user generating a one-time Password (OTP) for authentication or verification purposes.
Pre-condition	The user requests to generate an OTP.
Scenario flows	1. Users should sign up to generate OTP. 2. System generates a unique one-time password and associates it with the user's account. 3. System sends the generated token to the user. 4. System sends the generated OTP to the user.
Post condition	The system generates and sends an OTP to the user for Further authentication or verification.

Table 6-4 Login

Use case number	UC: 4
Use case name	Login
Actors	User, system
Description	This use case involves the user logging into their account using their credentials.
Pre-condition	The user must have a registered account.
Scenario flows	1. User accesses the login page or interface. 2. System presents a form for the user to enter their credentials (username, password, etc.).

Table 6-5 Verify Username and password

Use case number	UC: 5
Use case name	Verify Username and Password
Actors	User, System
Description	This use case involves the user verifying their entered Username and password combination.
Pre-condition	The user must provide their username and password.
Scenario flows	1. Users provide their username and password. 2 System validates the provided username and password against the stored user account information. 3 .System confirms the correctness of the username and password combination. 4.The system returns a verification result indicating the Validity of the username and password.
Post condition	The system verifies the correctness of the provided username and password combination.

Table 6-6 Display login error

Use case number	UC: 6
Use case name	Display login error
Actors	User, System
Description	This use case involves displaying an error message when the user fails to log in.
Pre-condition	The user's login attempt fails due to incorrect credentials.
Scenario flows	1. System detects that the login attempt was unsuccessful. 2. System displays an appropriate error message indicating the reason for the failure.
Post condition	An error message is displayed to the user explaining the Login failure.

Table 6-7 Access Using token

Use case number	UC: 7
Use case name	Access Using OTP
Actors	User, System
Description	This use case involves accessing a secure feature or action using the generated token.
Pre-condition	The user has a valid token.
Scenario flows	1. User enters the token showing in google authenticator. 2. User submits the token for verification. 3. System validates the entered token. 4. System grants access to the requested secure feature or action if the token is valid.
Post condition	The user gains access to the secure feature or action.

Table 6-8 Verify token

Use case number	UC: 8
Use case name	Verify token
Actors	System
Description	This use case involves verifying the entered token for authentication purposes.
Pre-condition	The user has entered a token.
Scenario flows	1. System compares the entered token with the generated token. 2. System determines whether the entered token is valid or not.
Post condition	The entered token is successfully verified.

Table 6-9 Display incorrect token

Use case number	UC: 9
Use case name	Display Incorrect token
Actors	User, System
Description	This use case involves displaying an error message when the entered token is incorrect.
Pre-condition	The user's entered OTP does not match the generated token.
Scenario flows	1. System detects that the entered token is incorrect. 2. System displays an appropriate error message indicating the incorrect token.
Post condition	An error message is displayed to the user indicating the Incorrect token.

Table 6-10 Access Using OTP

Use case number	UC: 7
Use case name	Access Using OTP
Actors	User, System
Description	This use case involves accessing a secure feature or action using the generated OTP.
Pre-condition	The user has a valid OTP.
Scenario flows	5. User enters the OTP showing in google authenticator. 6. User submits the OTP for verification. 7. System validates the entered OTP. 8. System grants access to the requested secure feature or action if the OTP is valid.
Post condition	The user gains access to the secure feature or action.

Table 6-11 Verify OTP

Use case number	UC: 8
Use case name	Verify OTP
Actors	System
Description	This use case involves verifying the entered OTP for authentication purposes.
Pre-condition	The user has entered an OTP.
Scenario flows	3. System compares the entered OTP with the generated OTP. 4. System determines whether the entered OTP is valid or not.
Post condition	The entered OTP is successfully verified.

Table 6-12 Display incorrect OTP

Use case number	UC: 9
Use case name	Display Incorrect OTP
Actors	User, System
Description	This use case involves displaying an error message when the entered OTP is incorrect.
Pre-condition	The user's entered OTP does not match the generated OTP.
Scenario flows	3. System detects that the entered OTP is incorrect. 4. System displays an appropriate error message indicating the incorrect OTP.
Post condition	An error message is displayed to the user indicating the Incorrect OTP.

Chapter 7 :Algorithm

In the proposed method, we have utilized TOTP as a starting algorithm to produce needed one time passwords.TOTP is dependent on HTOP; However, HTOP is used counterwhereas TOTP is a time-based algorithm. TOTP is going to generate an innovative worth after a determined period. Thisparticular occasion is known as the time step. TOTP supportsHMAC-SHA2 and HMAC-SHA1 hash functions . The Proposed system has two phases, namely: registration stage, an authentication phase. A comprehensive explanation of each phase is provided below. Before making use of this service, the user should register the information of theirs during a procedure known as the registration phase. Verification of that information may just be achieved by a procedure known as an authentication phase. Each of the suggested materials and strategies are completed in the system during both registration process as well as the login procedure.

Registration Phase

After the registration is done, the client app creates a six digit one time password (OTP) that may be utilized for the authentication aim. the registration process of the proposed system is working as follows.

Step 1: The user input his credential information on the server.

Step 2: The server stores the user's information.

Step 3:The server generates the QR code and verifies the token

Step 4:The user verifies the QR code using the user's mobile device and enters the token .

Step 5: The registration process was successful.

step 6: user’s login after entering the credentials the server generates a token and after email otp.

authentication phase:

Step 1: The user input his credential information on the server.

Step 2: The server determines the user’s information.

Step 3: The server generates the QR code.

Step 4: On the user side, the user will open the application.

Step 5: the user will get TOTP number after decode the QR code

Step 7: The user enter the token to verify

Step 8: the user will input the TOTP number in the server side, if matched,

Step 9: Authentication successful

7.1 Architecture design

The system architecture of our project (multi-factor authentication) involves components such as the user interface, backend server, user management, TOTP generation, authenticator app, time synchronization, TOTP verification, multi-factor integration, and security measures. The user interface allows users to initiate the authentication process, while the backend server handles requests and coordinates TOTP generation and verification. User management stores account information, and TOTP generation generates passwords based on shared secret keys. The authenticator app generates TOTPs on the user's device. Time synchronization ensures accurate TOTP generation, and TOTP verification compares user-provided TOTPs with expected values

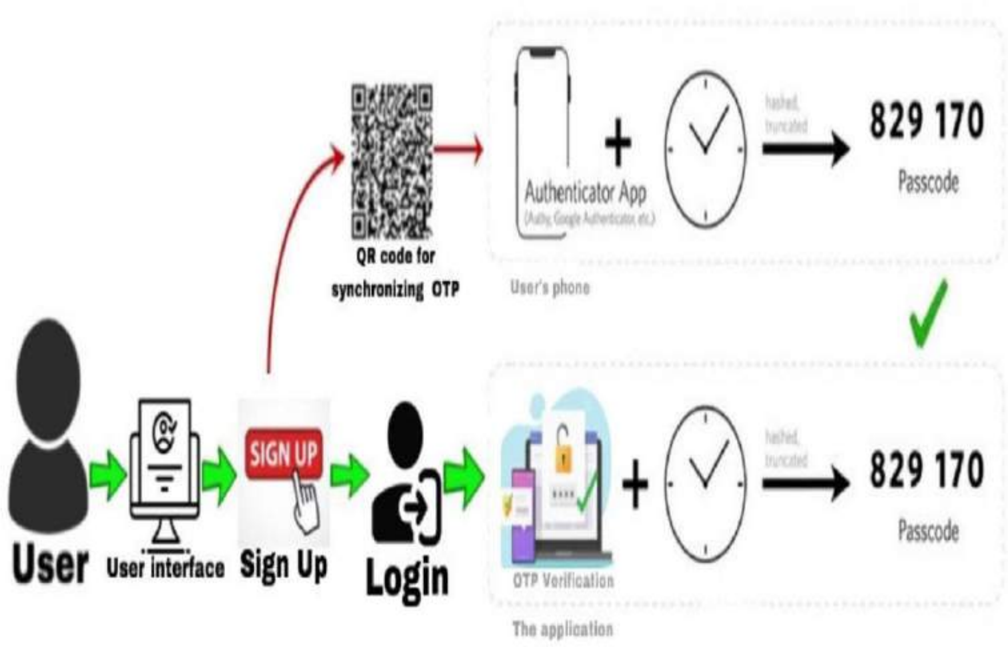


Figure 7-1 Architectural design

TOTP algorithm (RFC 6238) implies that an OTP is a product of two parameters encrypted together. These are a common value, which is a shared secret key, or seed; and a variable, in this case – the running time. These parameters are encrypted with a hash function.

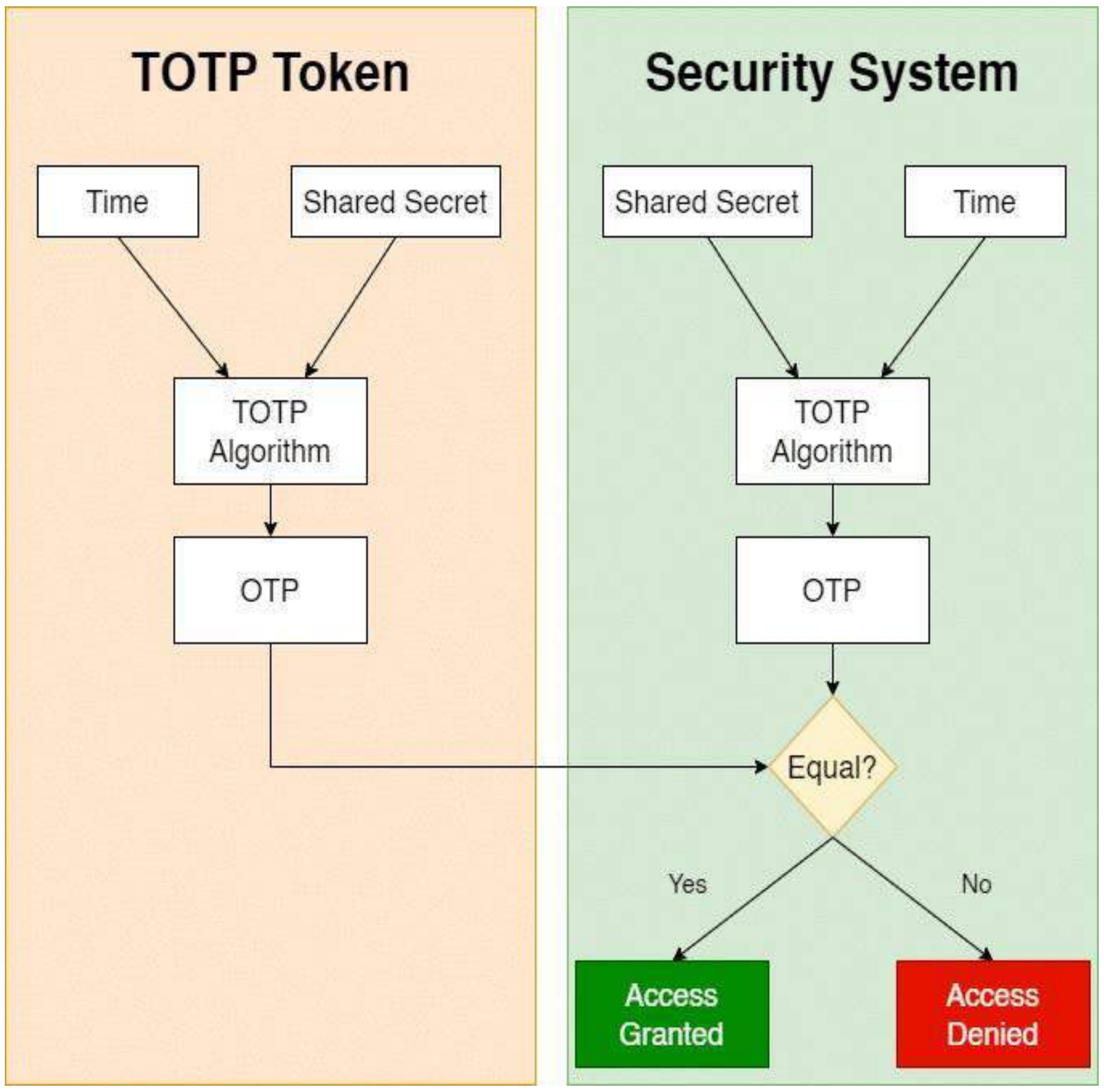


Figure 7-2 TOTP algorithm

Here's a TOTP algorithm example to illustrate:

A user wants to log into a TOTP MFA protected application or website. For the OTP authentication to run, the user and the TOTP server need to initially share a static parameter (a secret key).When the client logs into the protected website, they have to confirm they possess the secret key. So their TOTP token merges the seed and the current time step and generates a HASH value by running a predetermined HASH function. This value essentially is the OTP code the user sees on the token.

Since the secret key, the HASH function, and the timestep are the same for both parties, the server makes the same computation as the user's OTP generator.

The user enters the OTP and if it is identical to the server's value, the access is granted. If the results of the calculations aren't identical, the access is, naturally, denied.To explain the above example a bit let's note here that the mentioned seed is a string of random characters, usually 16–32 characters long. "Sharing" the key usually implies scanning a QR code that shows the seed generated by the server with the client's TOTP app. alternatively, the key is already programmed in their TOTP device. The time step is calculated using UNIX time, which starts on January 1, 1970, UTC. The timesteps are to be 30 or 60 seconds, so the time value used for TOTP is the number of seconds run since 00:00 January 1, 1970, divided by 30, or 60. Finally, the mentioned HASH function is a cryptographic mathematical function that simply changes one value into another and usually shortens the result to 6-8 symbols. This result is what we called a HASH value above.

7.2 Object Oriented Design

7.2.1 Class Diagram

In the class diagram, we've organized and grouped the various elements of the proposed system, such as (users, registration, login, token generation, and OTP) based on their specific functions and characteristics. We've used Python classes to represent each entity and define their functionalities. Figure 7-2 below provides a visual representation of the essential classes that have been identified and are scheduled for development in this project.

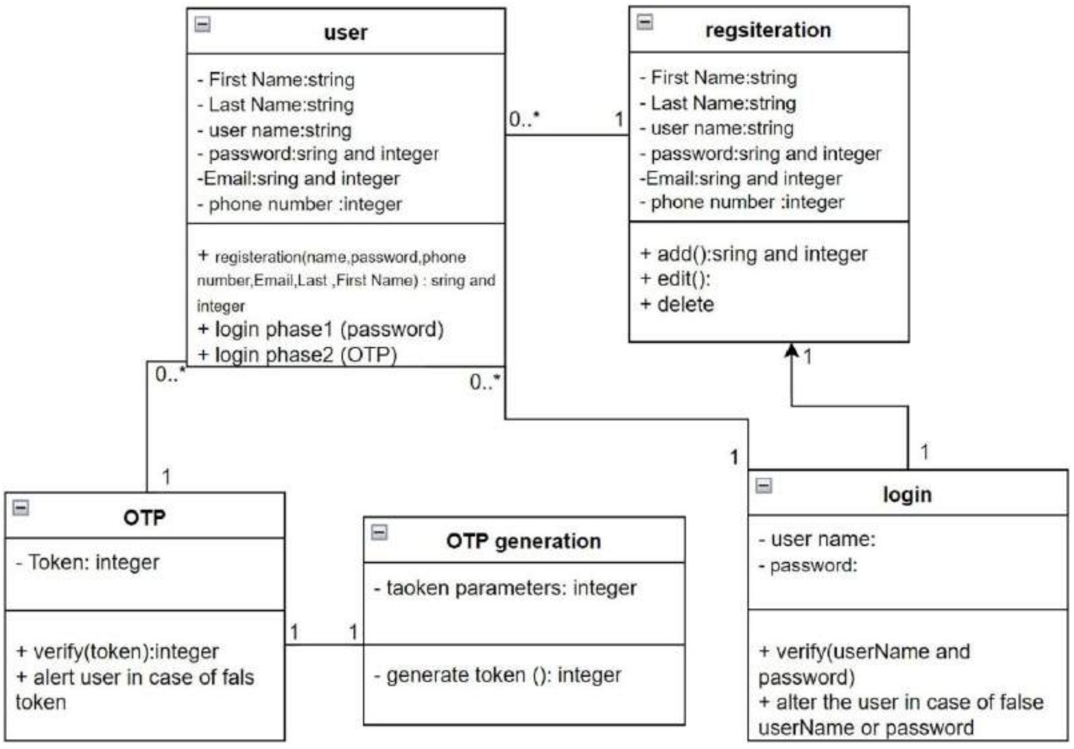

user
- First Name:string
- Last Name:string
- user name:string
- password:sring and integer
-Email:sring and integer
- phone number :integer
+ registeration(name,password,phone number,Email,Last ,First Name) : sring and integer
+ login phase1 (password)
+ login phase2 (OTP)
regsiteration
- First Name:string
- Last Name:string
- user name:string
- password:sring and integer
-Email:sring and integer
- phone number :integer
+ add():sring and integer
+ edit():
+ delete
0..*
1
0..*
0..*
1
1
1
1
1
1
OTP
- Token: integer
+ verify(token):integer
+ alert user in case of fals token
OTP generation
- taoken parameters: integer
- generate token (): integer
login
- user name:
- password:
+ verify(userName and password)
+ alter the user in case of false userName or password

Figure 7-3 Class Diagram

7.2.2 Sequence Diagrams

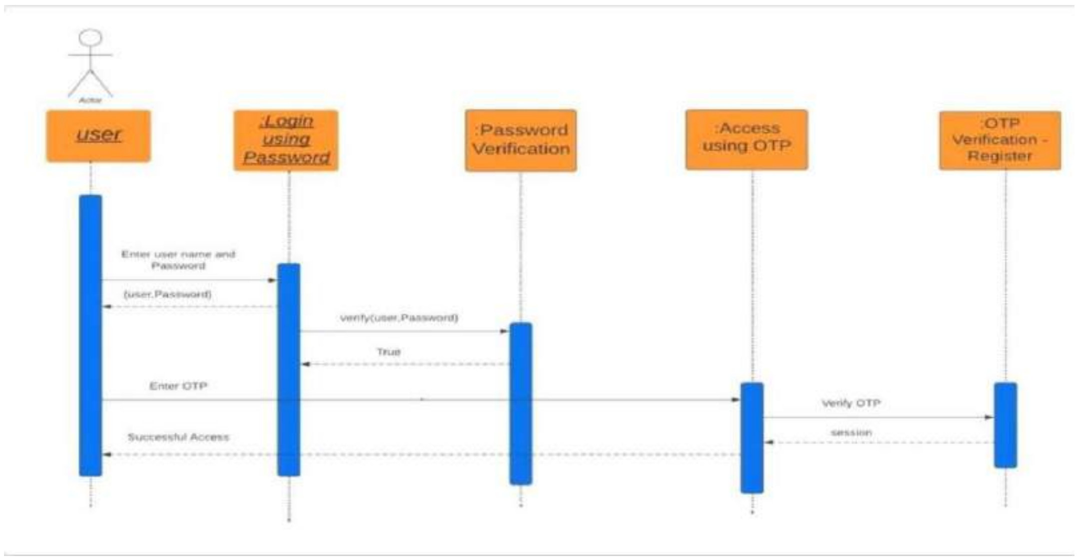
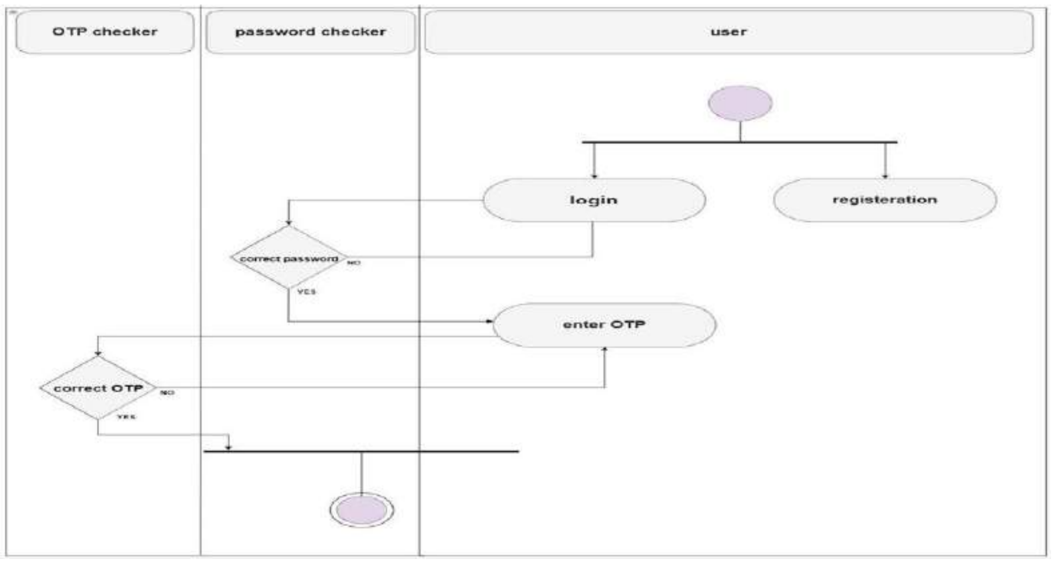


Figure 7-4 sequence diagram

There is a type of interaction diagram that provides a comprehensive depiction of the step-by- step execution of operations. These diagrams depict the interactions between objects within a collaborative context. Figure 7-4 presents the sequence diagrams for the proposed system. These diagrams provide a time-focused representation of the application flow, showcasing the sequence of interaction they occur.

7.2.3 Activity Diagram

Figure 7-5 illustrates the activity diagram of the activity diagram for the proposed application.



Chapter 8: Implementation

8.1 Introduction

During this stage, the system specifications are transformed into a functional and dependable solution. This is when the actual coding of the system takes place. The duration and effort required for this phase are significantly influenced by the preceding phases, as both the analysis and design stages serve as the foundation for implementation. In this same phase, initial testing, such as unit testing, is conducted by the developer to confirm if the system's requirements at the unit level align with the development. Typically, it's challenging to accurately predict the duration of this phase because practical challenges may arise when translating the conceptual design into an executable one.

8.2 Tools and Languages

This section presents tools and language used during the system implementation.

Table 8-1 Technology summary

Technology construct	**Usage**
python	To develop authentication system
Vs code	An integrated development environment (IDE) for python
MySQL	To establish a connection to a MySQL database from python
Django	To develop the project

8.2.1 External Libraries

- mysqlclient-2.2.4.dist-info
- pyotp-2.9.0.dist-info
- pip-24.1.1

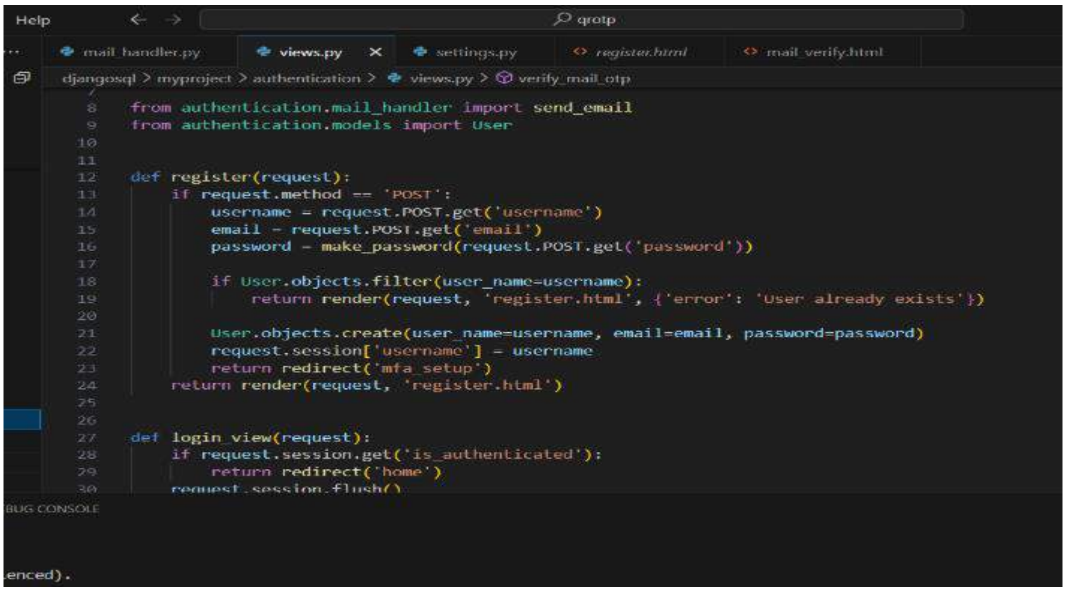
8.2.2 Required packages list

- Django==5.0.6
- django-otp==1.5.0
- mysqlclient==2.2.4
- pyotp==2.9.0
- sqlparse==0.5.0
- tzdata==2024.1

8.3 Main and Most Important Codes

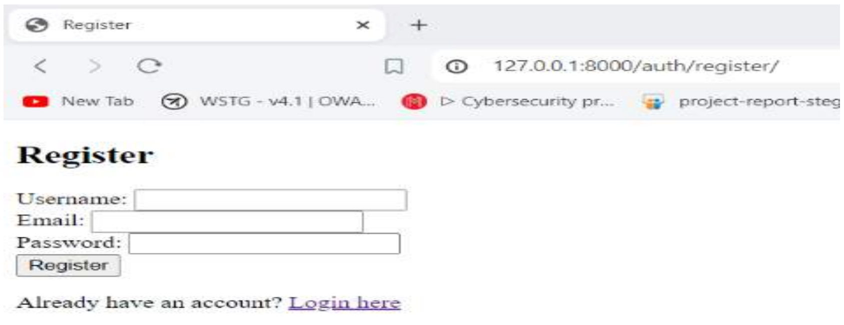
8.3.1 Code for register

This code snippet shows the user interactions that handle login and user registration in a GUI application in Figure 8.1, 8.1.1, 8.2, 8.2.1 It creates a new login window when a button is clicked and registers the user by storing their information in a database in figure 8-3.



```python
from authentication.mail_handler import send_email
from authentication.models import User


def register(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = make_password(request.POST.get('password'))

        if User.objects.filter(user_name=username):
            return render(request, 'register.html', {'error': 'User already exists'})

        User.objects.create(user_name=username, email=email, password=password)
        request.session['username'] = username
        return redirect('mfa_setup')
    return render(request, 'register.html')


def login_view(request):
    if request.session.get('is_authenticated'):
        return redirect('home')
    request.session.flush()
```

Figure 8-1 code for register



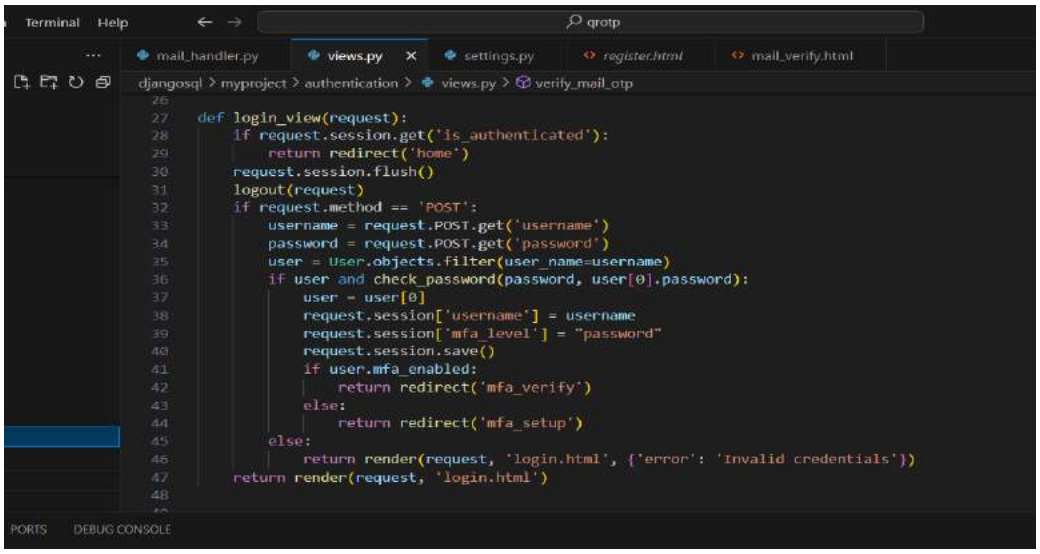
Register

Username:
Email:
Password:
Register

Already have an account? Login here

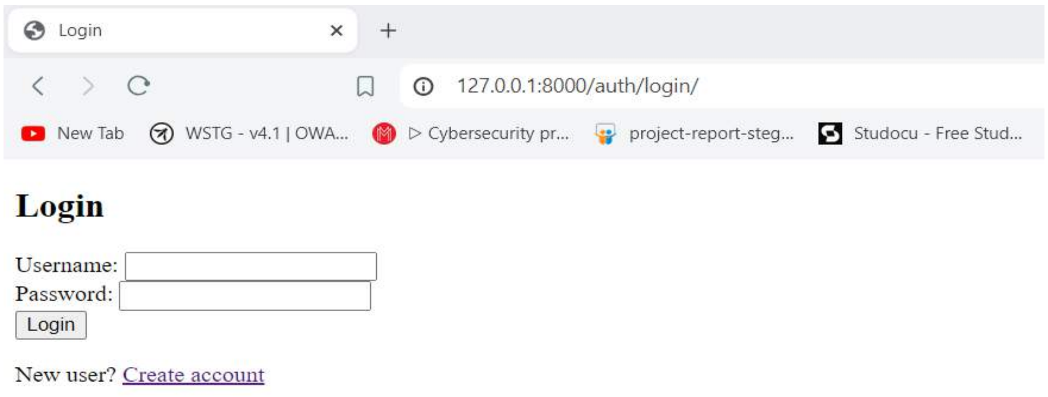
Figure 8-1-1 Register page

8.3.2 code for login



```python
def login_view(request):
    if request.session.get('is_authenticated'):
        return redirect('home')
    request.session.flush()
    logout(request)
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = User.objects.filter(user_name=username)
        if user and check_password(password, user[0].password):
            user = user[0]
            request.session['username'] = username
            request.session['mfa_level'] = "password"
            request.session.save()
            if user.mfa_enabled:
                return redirect('mfa_verify')
            else:
                return redirect('mfa_setup')
        else:
            return render(request, 'login.html', {'error': 'Invalid credentials'})
    return render(request, 'login.html')
```

Figure 8-2 Code for login page



127.0.0.1:8000/auth/login/

Login

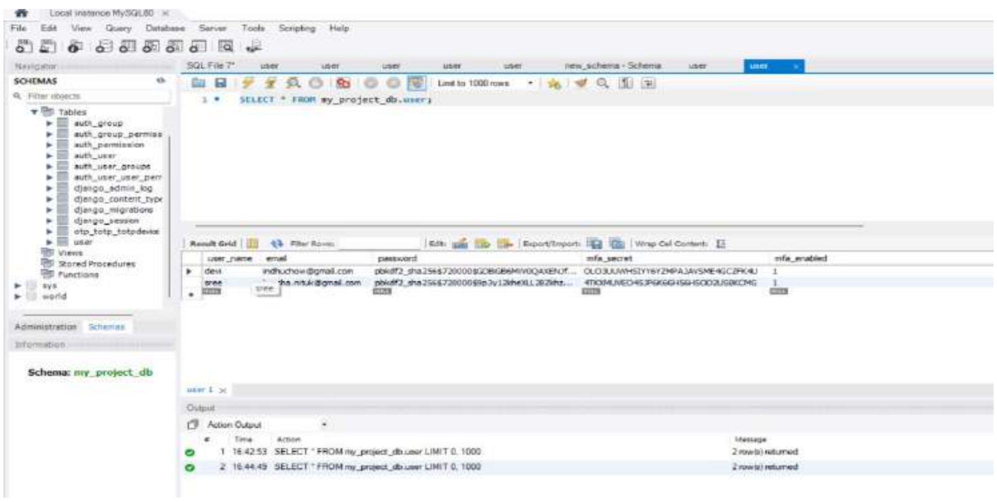
Username:

Password:

Login

New user? Create account

Figure 8-2-1 login page



```
SELECT * FROM my_project_db.user;
```

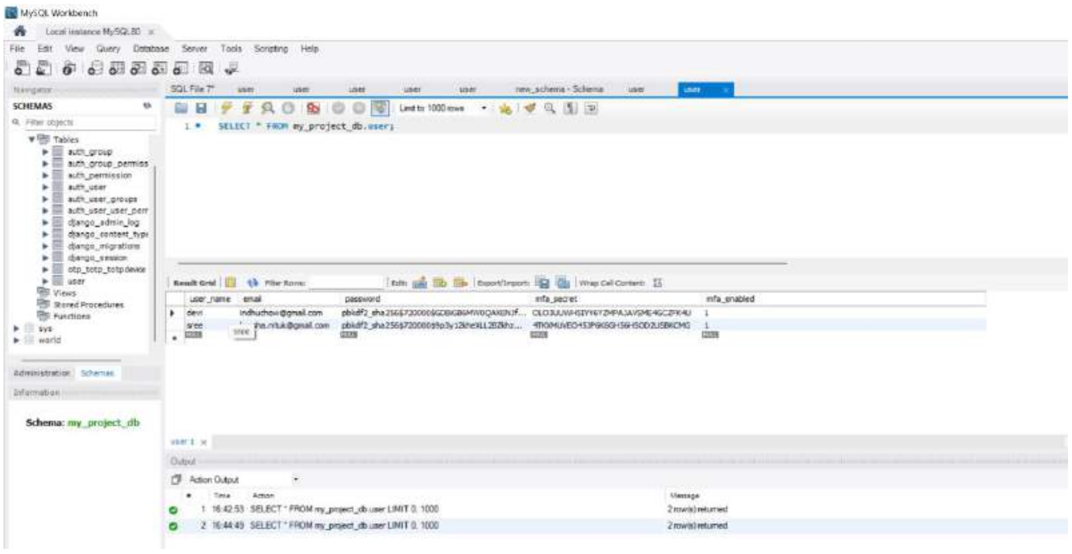
	user_name	email	password	mfa_secret	mfa_enabled
▶	devi	indhuchow@gmail.com	pbkdf2_sha256$720000$GDBGB6MIV0QAXENJf...	OLO3UUWHSIYY6YZMPAJAVSME4GCZPK4U	1
	sree	[illegible]sha.nituk@gmail.com	pbkdf2_sha256$720000$8p3y12bheXLL2B2bhz...	4TIOXMUVEO4SJP6K6GHSGHSODZUSBKCMG	1

#	Time	Action	Message
1	16:42:53	SELECT * FROM my_project_db.user LIMIT 0, 1000	2 row(s) returned
2	16:44:49	SELECT * FROM my_project_db.user LIMIT 0, 1000	2 row(s) returned

Figure 8-3 database registry

8.3.3 Database

First, we created a database in MySQL because it is an open-source database, meaning it is freely available for use and modification Figure 5-3. This reduces software costs and allows for customization to meet specific needs. And is known for its simplicity and ease of use and can handle large amounts of data and supports scalability by allowing users to efficiently manage and organize databases as they grow. It's suitable for both small-scale projects and large enterprise applications. MySQL includes robust security features to protect data integrity and privacy. It supports user authentication, access controls, and encryption, helping developers implement secure database applications. Integration Capabilities MySQL integrates well with various programming languages and development frameworks. It provides connectors and APIs for languages such as Java, Python, PHP, and more, facilitating seamless integration with different applications. Overall, MySQL is a versatile and powerful database management system that is widely used in web development, enterprise applications, and various other domains due to its reliability, performance, and extensive feature set.



```
1  SELECT * FROM my_project_db.user;
```

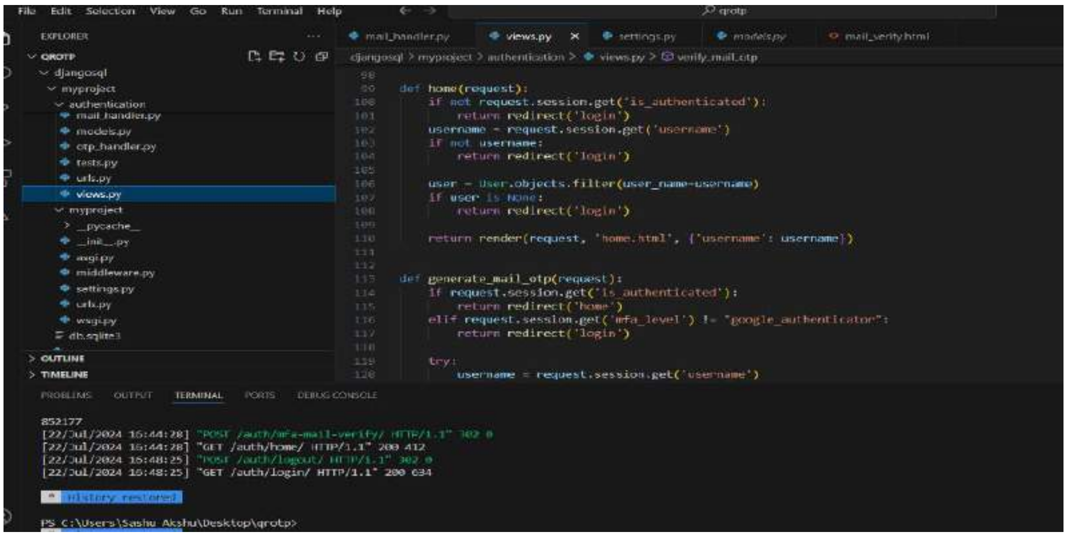
user_name	email	password	mfa_secret	mfa_enabled
devi	indhuchow@gmail.com	pbkdf2_sha256$720000$GDBGB6MW0QAKIN3f...	CLO3UUWHSIYY6YZMPAJAVSME4GCZFK4U	1
sree	sha.rituk@gmail.com	pbkdf2_sha256$720000$9p3y12kheXLL2BZkhz...	4TIOXMUVEO453P6KGGH56HSODZUSBKCMG	1
NULL		NULL	NULL	NULL

```
#  Time      Action                                          Message
1  16:42:53  SELECT * FROM my_project_db.user LIMIT 0, 1000  2 row(s) returned
2  16:44:49  SELECT * FROM my_project_db.user LIMIT 0, 1000  2 row(s) returned
```

Figure 8-4 sql database

8.4 Code for OTP generation

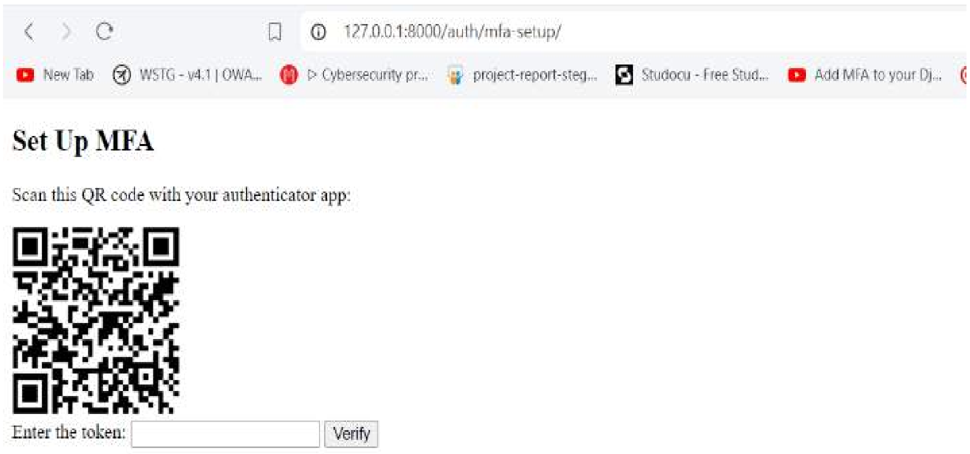
The code generates an OTP token by calculating a counter value based on a timestamp, generating a hash using the HMAC-SHA1 algorithm with a shared secret key, extracting part of the hash, and converting it into a digital OTP token. The code also includes methods to generate an OTP authentication URL, generate a QR code image for the URL, and display the QR code and OTP code in the graphical user interface (GUI).



```python
def home(request):
    if not request.session.get('is_authenticated'):
        return redirect('login')
    username = request.session.get('username')
    if not username:
        return redirect('login')

    user = User.objects.filter(user_name=username)
    if user is None:
        return redirect('login')

    return render(request, 'home.html', {'username': username})


def generate_mail_otp(request):
    if request.session.get('is_authenticated'):
        return redirect('home')
    elif request.session.get('mfa_level') != "google_authenticator":
        return redirect('login')

    try:
        username = request.session.get('username')
```

```
852177
[22/Jul/2024 16:44:28] "POST /auth/mfa-mail-verify/ HTTP/1.1" 302 0
[22/Jul/2024 16:44:28] "GET /auth/home/ HTTP/1.1" 200 412
[22/Jul/2024 16:48:25] "POST /auth/logout/ HTTP/1.1" 302 0
[22/Jul/2024 16:48:25] "GET /auth/login/ HTTP/1.1" 200 634

 History restored

PS C:\Users\Sashu Akshu\Desktop\qrotp>
```

Figure 8-4-1 code for OTP



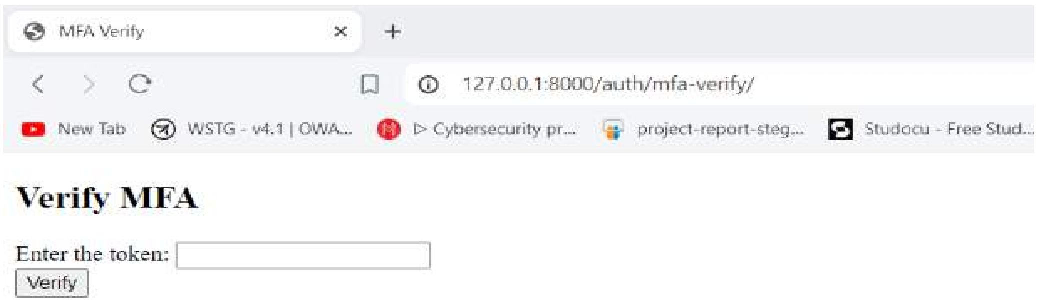
127.0.0.1:8000/auth/mfa-setup/

Set Up MFA

Scan this QR code with your authenticator app:

Enter the token: [ ] Verify

Figure 8-4-2 MFA setup


MFA Verify

127.0.0.1:8000/auth/mfa-verify/

New Tab WSTG - v4.1 | OWA... ▷ Cybersecurity pr... project-report-steg... Studocu - Free Stud...

Verify MFA

Enter the token:

Verify

Figure 8-4-3 MFA verify

8.5 Evaluation Testing

8.5.1 Introduction

The testing process is part of the software development cycle. It involves gathering information about the quality of a system's operation and the implementation of its various features. Evaluation testing is a process that aims to evaluate the quality of a system's operation. It involves gathering information about its requirements and characteristics. For testing of our project, I followed three levels of testing: unit testing, integration testing, and system testing.

8.5.2 Testing Strategy

The following are all the types of testing which are going to be illustrated more in the following sections:

- Unit Testing
- Integration Testing
- Performance Testing

8.5.3 User Procedures

There are 9 main components that were mentioned previously. The combinations are as follows.

- Graphical User Interface (GUI)
- Database
- Registration
- QR generation token
- Login
- Token entry

- Token verification
- Email OTP entry
- OTP verification

8.5.4 Operator Procedures

The requirements which should be available in order for the testing to take a place that leads to testing the model in an accurate way are as the following.

- Computer Device
- Django to Run Project
- google Authenticator to scan the Generated QR Code
- My SQL database

8.5.5 Testing Approach

This section describes the approaches for testing which are going to be illustrated in sufficient detail to explain the major tasks, and types of testing to be performed. Also, explaining the methods, and the purpose of using these methods.

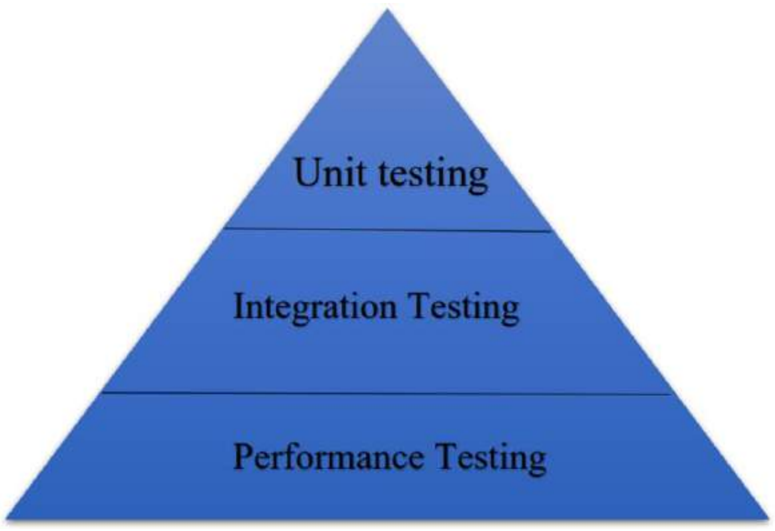


Figure 8-5 testing strategy

8.5.5.1 Unit Testing

Individual units or components of software are tested in unit testing, which is a sort of software testing. The goal is to ensure that each unit of software code works as intended. Developers perform unit testing throughout the development (coding) phase of an application. Unit tests are used to isolate a part of code and ensure that it is correct. A singular function, method, procedure, module, or object might be considered a unit. The following shows the unit testing which has been performed on the project.

8.5.5.2 Run Program

Table 8-2

step	action	UI unit response
1	Run Program	Register page will displayed

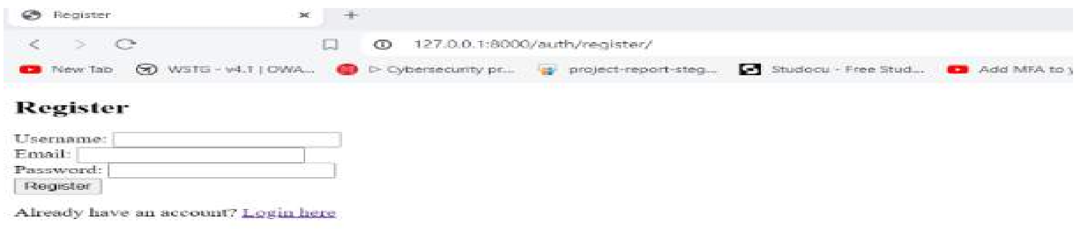
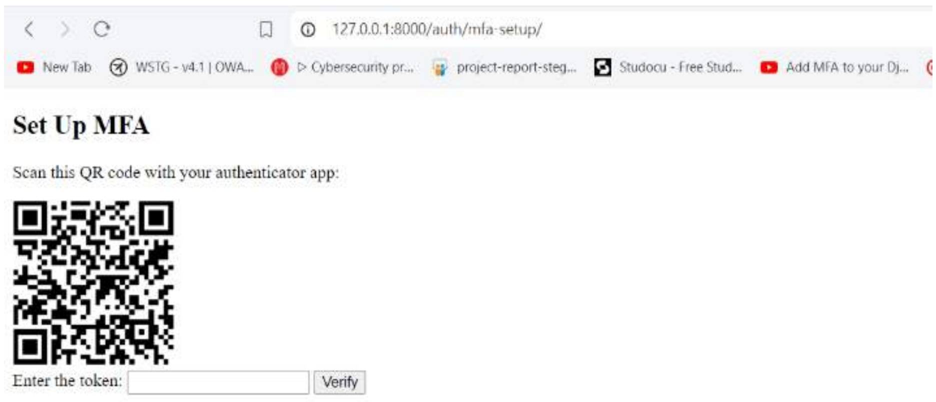


Figure 8-6 register page

User enter the required fields QR code will be displayed that shown in figure and user enters the token then the login page will be displayed.


127.0.0.1:8000/auth/mfa-setup/

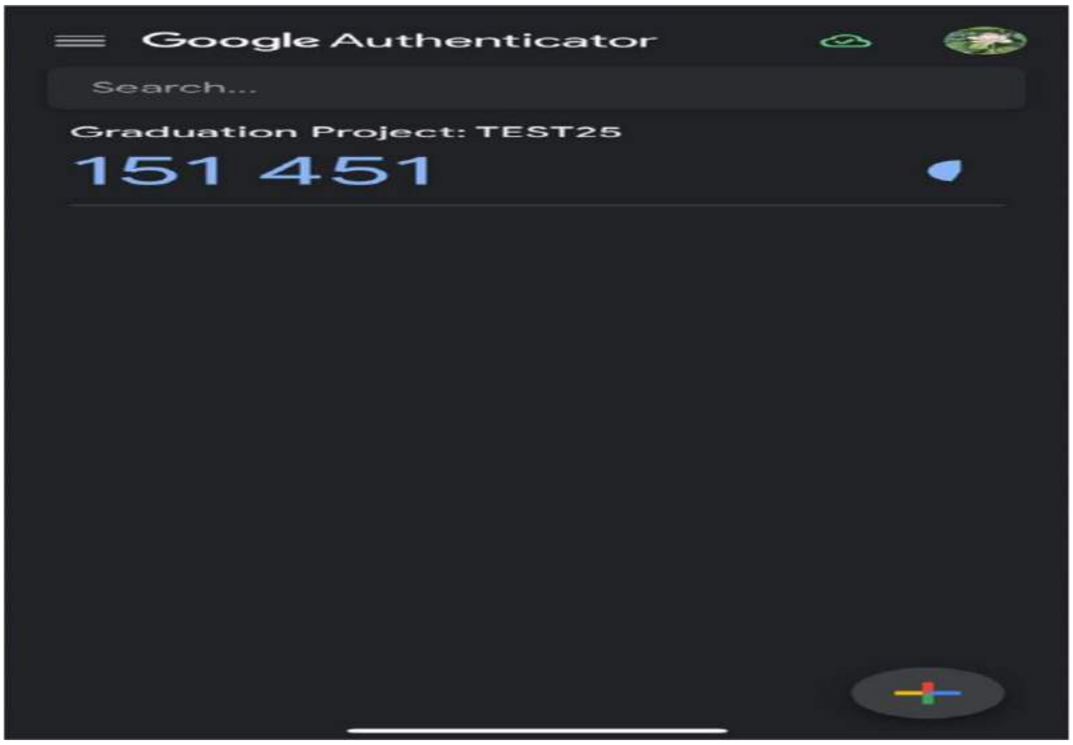
New Tab | WSTG - v4.1 | OWA... | ▷ Cybersecurity pr... | project-report-steg... | Studocu - Free Stud... | Add MFA to your Dj...

Set Up MFA

Scan this QR code with your authenticator app:

Enter the token: [ ] Verify

Figure 8-7


Google Authenticator

Search...

Graduation Project: TEST25

151 451

Figure 8-8 google authenticator

Users should scan the QR code using a mobile device and enter the token from google authenticator app the registration of the user will be successful. The google authenticator shown in Figure 8-8

8.5.5.3 Login

Table 8-3

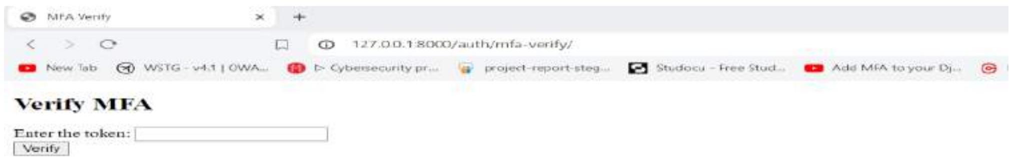
Step	Action	UI Unit Response
2	Enter Username and Password	IF username and password correct UI will move to OTP entry page

Figure 8-9 login

8.5.5.4 Token verification

Table 8-4

Step	Action	UI Unit Response
3	Enter The verification code displayed in Google Authenticator	If the token is correct, UI will move to the email verification page



Verify MFA

Enter the token:

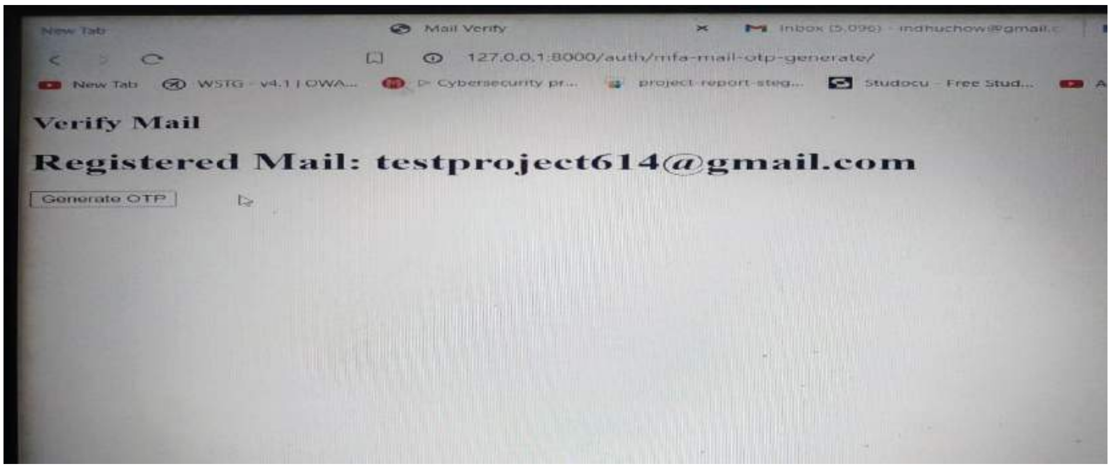
Verify

Figure 8-10Token verification

8.5.5.5 email otp verification

Table 8-5

Step	Action	UI Unit Response
3	Enter The verification code received to the registered mail id.	If the OTP is correct, UI will move to the welcome page



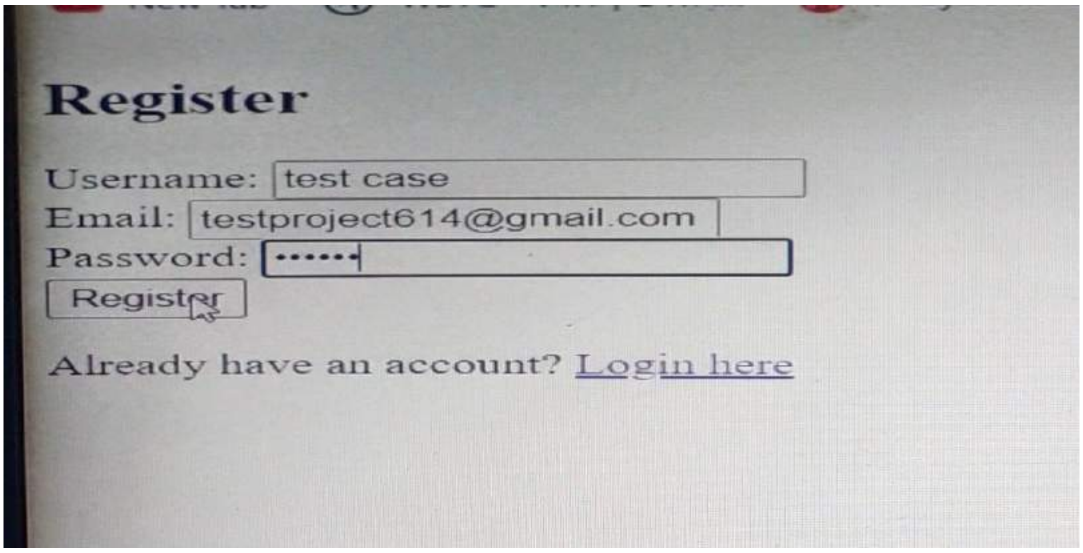
Verify Mail

Registered Mail: testproject614@gmail.com

Figure 8-11 email verification

Chapter 9: Results & Analysis

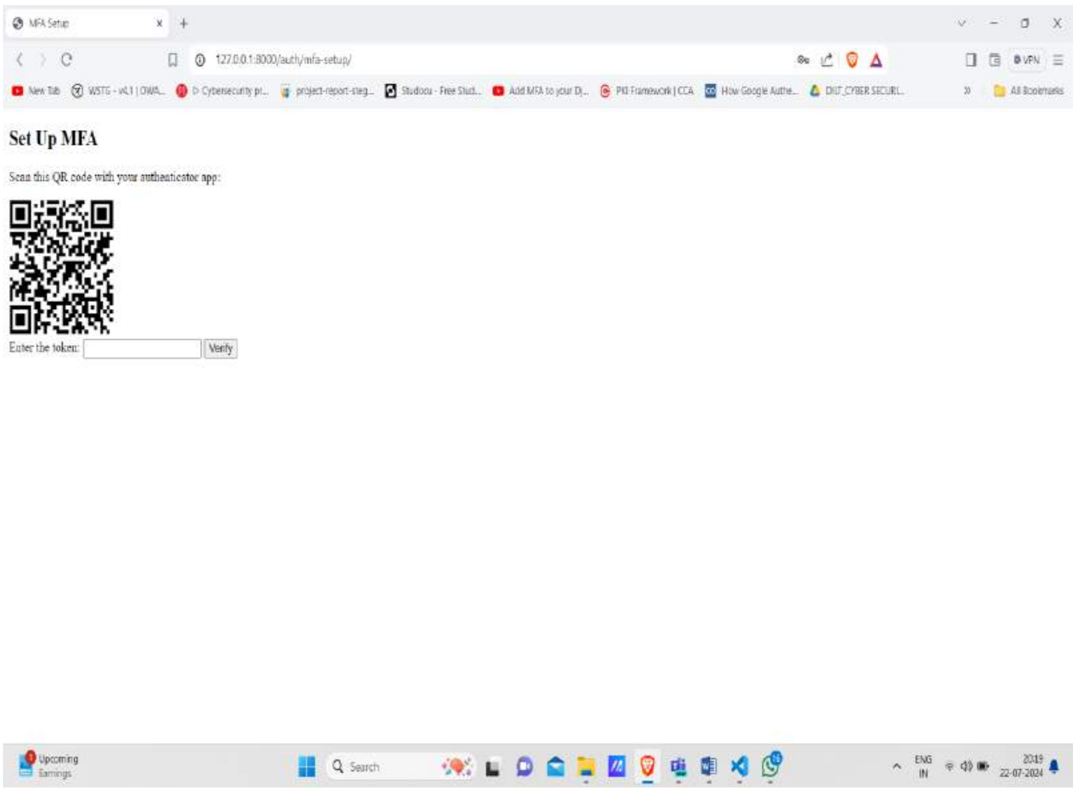
The expectation is that the response time of the detection action is going to be acceptable. Also, the response time of the model that handles the matching with the database is going to be reasonable. The below figure shows the process of the project.


Register

Username: test case
Email: testproject614@gmail.com
Password: ••••••
Register

Already have an account? Login here

Figure 9-1 registration page


MFA Setup

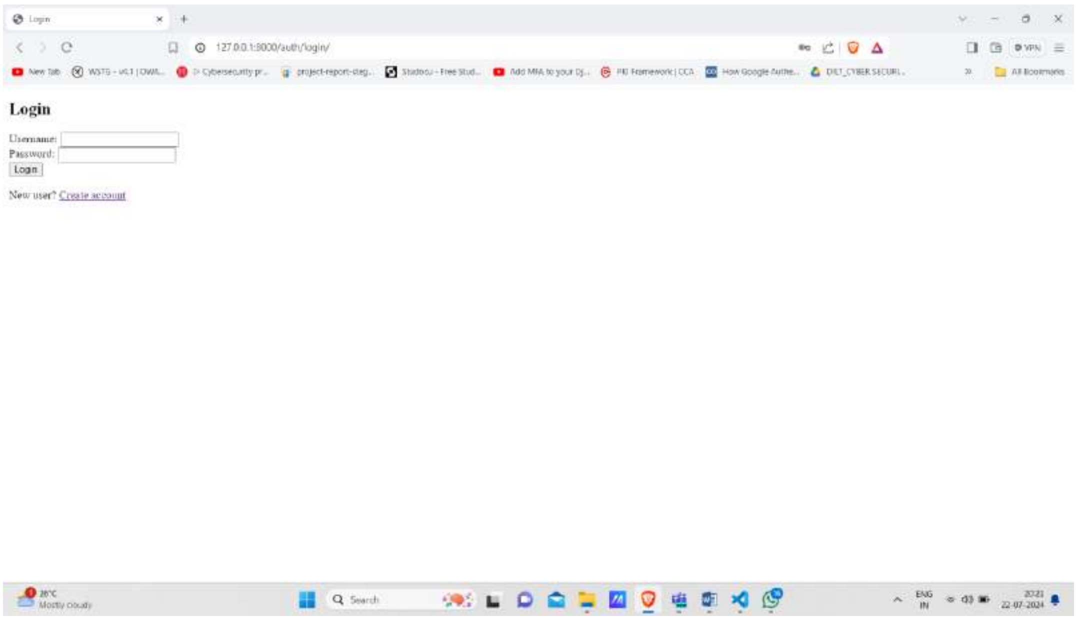
127.0.0.1:8000/auth/mfa-setup/

Set Up MFA

Scan this QR code with your authenticator app:

Enter the token: Verify

Figure 9-2 QR code generator for registration process



127.0.0.1:8000/auth/login/

Login

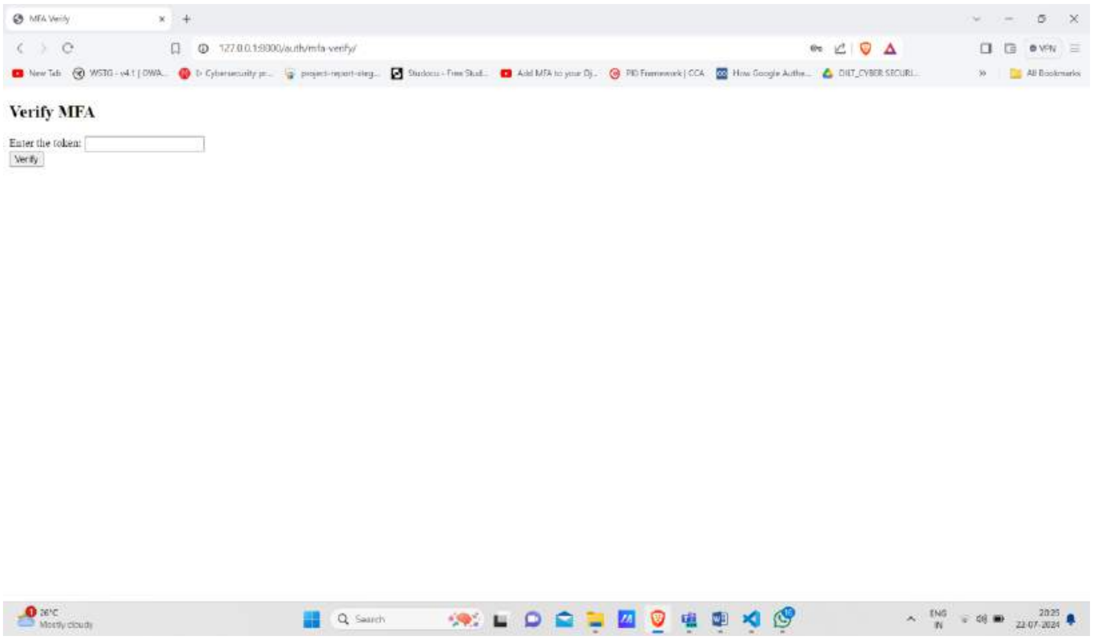
Username:

Password:

Login

New user? Create account

Figure 9-3 login page



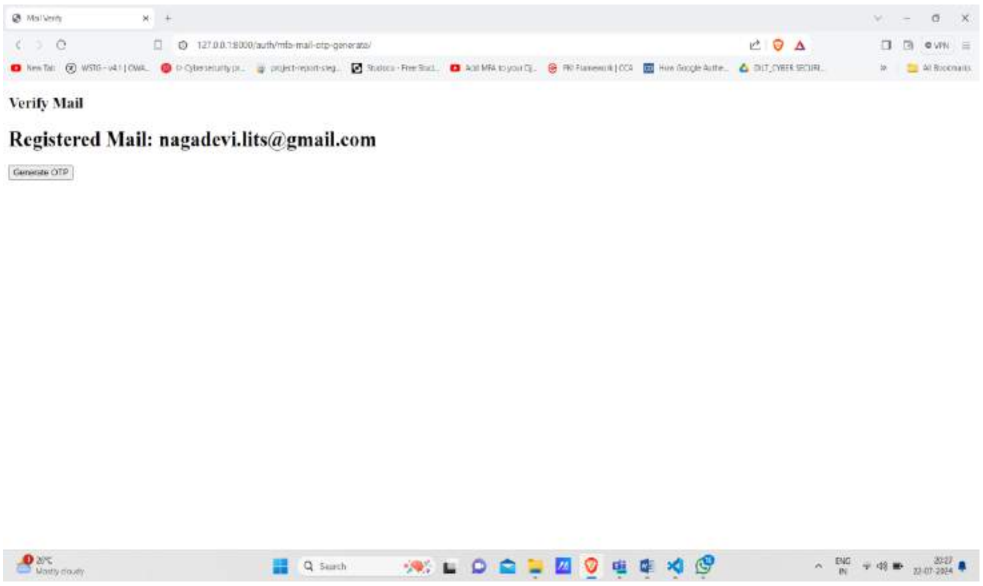
127.0.0.1:8000/auth/mfa-verify/

Verify MFA

Enter the token:

Verify

Figure 9-4 Verifying token

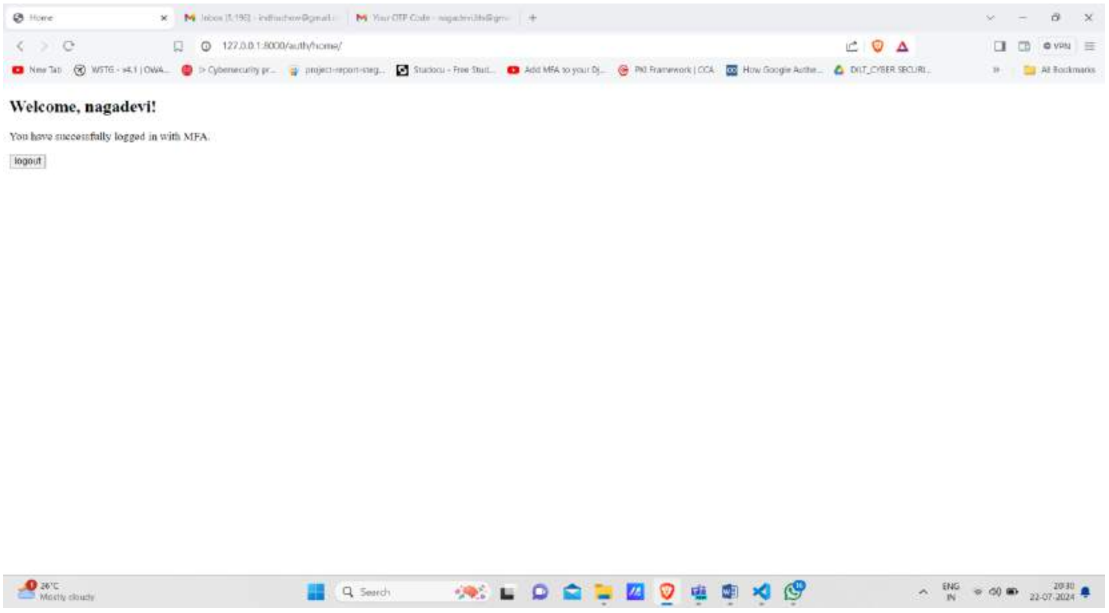


Verify Mail

Registered Mail: nagadevi.lits@gmail.com

Generate OTP

Figure 9-5 generating email OTP

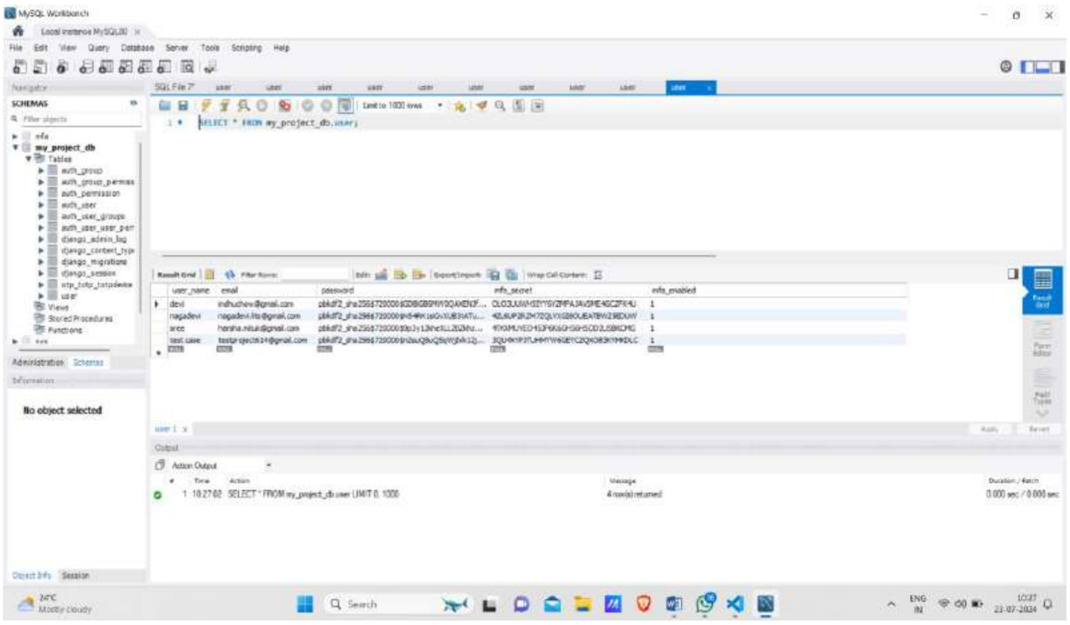


Welcome, nagadevi!

You have successfully logged in with MFA.

logout

Figure 9-6 welcome page



```
SELECT * FROM my_project_db.user;
```

user_name	email	password	mfa_secret	mfa_enabled
devi	indhuchew@gmail.com	pbkdf2_sha256$720000$[illegible]...	[illegible]	1
nagadevi	nagadevi.lits@gmail.com	pbkdf2_sha256$720000$[illegible]...	[illegible]	1
sree	harsha.nituk@gmail.com	pbkdf2_sha256$720000$[illegible]...	[illegible]	1
test case	testproject0614@gmail.com	pbkdf2_sha256$720000$[illegible]...	[illegible]	1

```
#  Time      Action                                        Message             Duration / Fetch
1  10:27:02  SELECT * FROM my_project_db.user LIMIT 0, 1000  4 row(s) returned  0.000 sec / 0.000 sec
```

Figure 9-7 database page

9.1 Summary

In this chapter, the system is tested through three approaches of testing; unit testing, integration testing, and performance testing. According to the testing results, the model performed as expected.

Chapter 10: Conclusion and Future scope

10.1 conclusion

For manipulating the vulnerabilities associated with using the password as a single-factor authentication method, our designed and implemented system considered two- factor authentication using the password comparison and time-based one-time password (TOTP) which is considered the most secure method to thwart unauthorized access, where the code is changed every 60 seconds.

The developed system consists of three main modules, sign up module, login, and TOTP generation module, in addition to auxiliary module for generating QR code for synchronizing the generated OTP between the application server and authenticator (on a mobile device). After successful registration, the QR code is generated regarding the generated OTP and the user can scan it using an authenticator application, such as Google Authenticator, for obtaining the synchronized OTP code. If the user enters the valid password for login, the system will forward him to the token verification page to enter the valid token code displayed on the authenticator application and after the email verification page displays Results of the conducted evaluation testing demonstrating the effectiveness of the developed system.

10.2 Future scope

In future, we may consider additional features to the system, by considering a third factor of authentication, biometric features, such as fingerprint, face recognition, hand geometry to reinforce the authentication mechanism.

References

[1](n.d.). Django MFA — django-mfa 1.0 documentation. Retrieved July 26, 2024, from https://django-mfa.readthedocs.io/en/latest/

[2]cloud-with-django (Cloud With Django) · GitHub. (n.d.). GitHub. Retrieved July 26, 2024, from https://github.com/cloud-with-django

[3]dasguptha d, r. a. (n.d.). A fuzzy decision support system for multifactor authentication. A fuzzy decision support system for multifactor authentication. https://link.springer.com/article/10.1007/s00500-017-2607-6

[4]The Importance of Testing Multi-Factor Authentication · CODA. (n.d.). CODA. Retrieved July 26, 2024, from https://www.codasecurity.co.uk/articles/mfa-testing/

[5]Jain, S. (2024, July 3). Agile Development Models - Software Engineering. GeeksforGeeks. Retrieved July 26, 2024, from https://www.geeksforgeeks.org/software-engineering-agile-development-models/

[6] Joshi, P. (2018). Enhanced Security in Authentication Using QR Code. International Journal for Research in Applied Science and Engineering Technology. https://doi.org/10.22214/IJRASET.2018.4455.

[7]Kamrul, M., H. S. Z. (2020). An Improved Time-Based One Time Password Authentication Framework for Electronic Payments. https://doi.org/10.14569/ijacsa.2020.0111146.

[8]Kumar, a. R. (2012). A Comprehensive Study on Multifactor Authentication. A Comprehensive Study on Multifactor Authentication, 561-568. https://doi.org/10.1007/978-3-642-31552-7_57.

[9]Pretorius, A. (2022, November 14). Multi-factor authentication (MFA) for your Django admin page. Cloud With Django. Retrieved July 26, 2024, from https://www.cloudwithdjango.com/multi-factor-authentication-mfa-for-your-django-admin-page/

[10]Sonawane, S, T. D. (2014). Risk Based Multilevel and Multifactor Authentication using Device Registration and Dynamic QR code based OTP Generation. International Journal of Advanced Research in Computer and Communication Engineering. https://doi.org/10.17148/IJARCCE.2014.31053.

Annexure – I

LIST of tables

Annexure –II

LIST of Figures