OTP WITH CLIENT CERTFICATE BASED AUTHENTICATION FOR SECURE WEB TRANSACTION

## A PROJECT REPORT

***Submitted by***

# BHAVAN KUMAR S B (Reg.No.1920106013) KANNAN S (Reg.No.1920106033) GOBIKA S (Reg.No.1920106703)

***in partial fulfillment for the award of the degree of***

# BACHELOR OF TECHNOLOGY

## IN INFORMATION TECHNOLOGY

**SONA COLLEGE OF TECHNOLOGY**

# ANNA UNIVERSITY: CHENNAI 600 025

**NOVEMBER 2023**

# ANNA UNIVERSITY: CHENNAI 600 025

## BONAFIDE CERTIFICATE

Certified that this project report **“OTP WITH CLIENT CERTFICATE BASED AUTHENTICATION FOR SECURE WEB TRANSACTION ”** is

the bonafide work of **“BHAVAN KUMAR.S.B(1920106013), KANNAN.S(1920106033), GOBIKA.S(1920106703)”** who carried out the

project work under my supervision.

|  |  |
| --- | --- |
| **SIGNATURE** | **SIGNATURE** |
| (Dr.J.Akilandeswari) | (A.Naveen Kumar) |
| Professor | Assistant Professor |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| Department of Information Technology | Department of Information Technology |
| Sona College of Technology | Sona College of Technology |
| Salem-636 005 | Salem-636 005 |

Submitted for Mini Project viva voce examination held on

……………………………..

|  |  |
| --- | --- |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

# ABSTRACT

This study proposes a robust security protocol for online banking transactions, leveraging a two-factor authentication system comprising a one- time password (OTP) and a password-protected non-shareable certificate. Upon logging into the internet banking platform, a unique certificate is dynamically generated, enhancing security by preventing certificate sharing. The certificate is further fortified with a user-defined password, adding an additional layer of protection.

During transaction initiation, both the OTP and certificate are transmitted securely to the bank's server. Successful validation of both elements ensures the transaction proceeds, while any discrepancy results in an automatic transaction failure, safeguarding against unauthorized access attempts. This dual authentication process provides an effective defense against fraudulent activities, bolstering the integrity of online banking services.

By implementing this security measure, financial institutions can significantly reduce the risk of unauthorized access and subsequent fraudulent transactions. This system represents a vital step towards fortifying the security infrastructure of internet banking, ensuring a safer and more reliable experience for users.

## ACKNOWLEDGEMENT

First and foremost, we thank to **power of almighty** for showing us inner peace and for all blessings. Special gratitude to our parents, for showing their support and love always.

We like to acknowledge the constant support provided by **Sri.C.Valliappa,**

Chairman, for his consistent motivation in pursuing my project.

Our gratitude thanks to our Vice Chairmen **Sri. Chocko Valliappa** and **Sri. Thyagu Valliappa** who leads us in a narrow path towards success in all the way.

We are immensely grateful to our principal **Dr.S.R.R.Senthil Kumar,** who has been our constant source of inspiration.

We express our sincere thanks to the Head of Information Technology, **Dr.J.Akilandeswari,M.E,Ph.D,** for providing adequate laboratory facilities to complete this thesis.

We feel elated to keep on record our heartfelt thanks and gratitude to our project guide **A**.**Naveen Kumar, Assistant Professor** our steadfast inspiration, for his valuable guidance, untiring patience and diligent encouragement during the entire span of this project.

We feel proud in sharing this success with my staff members who helped directly or indirectly in completing this project successfully.

## TABLE OF CONTENTS

**CHAPTER NO. TITLE PAGE NO.**

[ABSTRACT iii](#_TOC_250019)

[LIST OF FIGURES vii](#_TOC_250018)

LIST OF SYMBOLS AND ABBREVIATIONS viii

1. INTRODUCTION 1
   1. [Client Certificate 1](#_TOC_250017)
      1. [What is Client Certificate 1](#_TOC_250016)
      2. [Characteristics of Certificate 2](#_TOC_250015)
      3. Application 5
      4. [Layered Architecture 6](#_TOC_250014)
      5. [Components of Transcation Platform 7](#_TOC_250013)
   2. [Properties 9](#_TOC_250012)
      1. OTP and certificate-based security in 10

internet banking

1. LITERATURE SURVEY 12
2. HARDWARE AND SOFTWARE 14

REQUIREMENTS

1. PROJECT DESCRIPTION 15
   1. [Aim 15](#_TOC_250011)
   2. [Implementation Process 15](#_TOC_250010)

[4.2. 1 OTP Based Authentication 16](#_TOC_250009)

[4.2.2 OTP Flowchart 16](#_TOC_250008)

4.2.2 OTP Generation 16

* 1. Step to Generate the Certificate 17
     1. [Flowchart to Generate Certificate 17](#_TOC_250007)
     2. Generate the SSL Certificate 18
     3. Flowchart to Setup 20
  2. [Scope of the Project 20](#_TOC_250006)
     1. Integration of OTP and Certificate 21

1. METHODOLOGY 25
   1. [openSSL 25](#_TOC_250005)
      1. [Cryptography Functions 25](#_TOC_250004)

[5.2 SHA-256](#_TOC_250003)

[5.2.1 Key Characteristics of SHA-256 27](#_TOC_250002)

1. CONCLUSIONS AND FUTURE WORK 28
   1. [Conclusions 28](#_TOC_250001)
   2. Future work 29
   3. APPENDICES 30
      1. Source Code 30
      2. Screenshots 49
   4. [References 52](#_TOC_250000)

## LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PG NO.** |
| 1.1 | Layered architecture | 6 |
| 1.2 | Overall architecture | 7 |
| 4.1 | OTP generation | 17 |
| 4.2 | Certificate Generation | 17 |
| 4.3 | Flow graph-Certificate Generation | 20 |
| 4.4 | Flow graph –OTP and Certificate | 24 |
| 6.3 | Screenshots | 49 |

**LIST OF ABBERVIATIONS**

OTP - One Time Password

CA - Certificate Authority

SSL - Secure Sockets Layer TLS - Transport Layer Security

HTTPS - Hypertext Transfer Protocol Secure CSR - Certificate Signing Request

# CHAPTER 1

## INTRODUCTION

This project report addresses the critical challenge of enhancing security and user-friendliness in online transactions. Existing methods frequently exhibit vulnerabilities due to inadequate authentication measures. To rectify this, our solution integrates one-time passwords (OTPs) and client certificates, fortifying security and preserving a seamless user experience to effectively counter the threats of unauthorized access and fraud.

## CLIENT CERTIFICATE

A client certificate is a digital certificate that serves as a secure means of authenticating the identity of a user or client device in online interactions. It contains a public key and additional information, and is issued by a trusted Certificate Authority (CA). During communication with a server, the client presents this certificate to verify its identity. The server uses the public key from the certificate to establish a secure connection, enhancing authentication and protecting against unauthorized access. Client certificates are commonly used in secure online transactions, VPNs, and other contexts where robust user authentication is crucial.

## WHAT IS CLIENT CERTIFICATE

Client certificate authentication using SSL (Secure Sockets Layer) certificates is a robust security measure employed in bank transactions to ensure the integrity and confidentiality of sensitive financial information. In this process, a client, such as an online banking user, is issued a digital certificate containing a public key and other identifying information. This certificate is generated by a Certificate Authority (CA) following a stringent identity verification process.

The key security feature lies in the secure ownership of the associated private key by the certificate holder.

When the user initiates a bank transaction, the server requests the client's SSL certificate for authentication. The client presents the certificate, proving its identity in a highly secure manner. The server, armed with the corresponding public key, encrypts the transaction data to ensure confidentiality during transmission. The client, possessing the private key, decrypts the data upon receipt, completing a secure and authenticated transaction.

This method adds a crucial layer of security to bank transactions, as possession of both the SSL certificate and its private key is required for authentication. This two-factor authentication significantly enhances resistance against unauthorized access, safeguarding the user's financial data and the integrity of the transaction. The SSL certificate generation process, underpinned by stringent identity verification and encryption practices, forms a cornerstone in fortifying the security posture of online banking transactions.

## CHARACTERISTICS OF CERTIFICATE

* + - * **Key Security Feature:**

This method adds a crucial layer of security to bank transactions, as possession of both the SSL certificate and its private key is required for authentication. This two-factor authentication significantly enhances resistance against unauthorized access, safeguarding the user's financial data and the integrity of the transaction. The SSL certificate generation process, underpinned by stringent identity verification and encryption practices, forms a cornerstone in fortifying the security posture of online banking transactions**.**

## Client Certificate in Authentication:

Client certificate authentication is a secure process used when a user or client device wishes to access a protected online resource. In this mechanism, the client possesses a digital certificate containing a public key and additional identifying information. When prompted by the server, the client presents this certificate, initiating an authentication process. The server, equipped with the public key of a trusted Certificate Authority, validates the certificate, confirming the client's identity. This confirmation is crucial in establishing trust between the client and server. Upon successful verification, a secure communication channel is established, often using protocols like HTTPS. The digital certificate serves as a cryptographic credential, enhancing the overall security of the online interaction by ensuring that only authenticated clients gain access to the requested resource

## Issuing Authority:

In SSL (Secure Sockets Layer) or its successor TLS (Transport Layer Security) protocols, client certificates play a crucial role in establishing secure communication between a client and a server. These certificates, containing a public key and additional identifying information, are issued by a trusted third-party entity known as a Certificate Authority (CA). The CA's role is pivotal; it verifies the identity of the certificate holder through a stringent authentication process before issuing the certificate. This verification process by the CA establishes trust in the authenticity of the presented client certificate. When a client attempts to connect to a server secured by SSL/TLS, the server can use the client's certificate to authenticate the client's identity, ensuring a secure and encrypted channel for data exchange. The involvement of a trusted CA in this process enhances the overall security of the SSL/TLS protocol, assuring that only

legitimate clients with valid certificates gain access to the protected resources.

## Secure Communication:

In the process of secure communication, especially during authentication, the server utilizes the public key extracted from the client certificate to encrypt the exchanged data. This encryption ensures that any information transmitted between the client and server is shielded and remains confidential. The public key, being publicly accessible, is used for encryption by the server. On the other side, the private key, exclusive to the certificate owner, plays a crucial role. It is employed by the client to decrypt the data received from the server. This asymmetric encryption mechanism ensures that even if the data is intercepted during transmission, it remains unreadable without the corresponding private key. In essence, the use of public and private key pairs in this cryptographic process enhances the security of the communication channel, safeguarding sensitive information exchanged between the client and server.

## Enhancing Security:

Client certificates play a pivotal role in enhancing security through the implementation of two-factor authentication. This approach requires the possession of both the client certificate and its associated private key for successful authentication. This dual-factor requirement adds an extra layer of security by necessitating not only something the user has (the certificate) but also something the user knows (the private key). The combination of these two elements fortifies the resistance against unauthorized access attempts, significantly reducing the risk of compromised credentials. This robust authentication mechanism contributes to a more secure environment, particularly in scenarios where

sensitive data or resources are involved, ensuring that only authorized individuals with both the physical certificate and the secret private key can gain access.

## APPLICTIONS

* + - * User Registration and Login:

In internet banking, users either register or log in to their accounts. During this process, client certificates are commonly employed to establish a secure and trusted connection with the banking server. This ensures that the users' identities are verified before accessing sensitive financial information.

* + - * Dynamic Client Certificate Generation:

Upon a successful login, a unique client certificate is generated for the user. This certificate is intricately linked to user details such as their mobile number, account number, and PAN (Permanent Account Number). This dynamic generation enhances security by providing a personalized and context-specific authentication credential.

* + - * Transaction Initiation with OTP:

Initiating a financial transaction triggers the generation of a one-time password (OTP). This additional layer of security adds a dynamic element to the authentication process, ensuring that each transaction requires a unique code for verification.

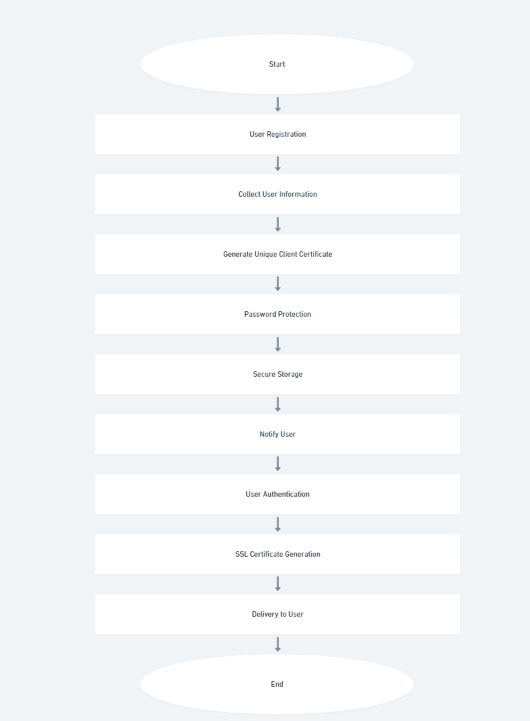
* + - * Submission of OTP and Client Certificate:

Users enter the generated OTP and submit both the OTP and the client certificate to the banking server during a transaction. This dual-factor authentication, combining something the user knows (OTP) with something the user has (client certificate), significantly strengthens the security posture.

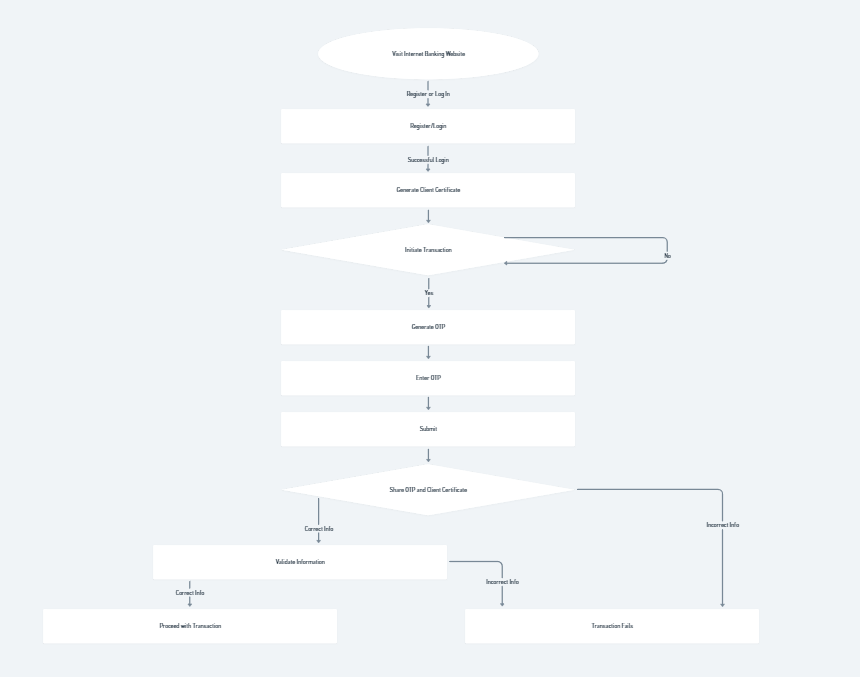
* + - * Server Validation and Transaction Processing:

The server validates the submitted information, checking the correctness of both the OTP and the client certificate. If the validation is successful, the transaction proceeds; otherwise, it fails. This stringent verification process helps prevent unauthorized access and fraudulent transactions.

## LAYERED ARCHITECTURE



**Figure 1.1 Layered architecture**



## Figure 1.2 Overall architecture

## COMPONENTS OF TRANSCATION PLATFORM

1. User Interface (UI)
2. Certificate Generation Module
3. One-Time Password (OTP) Generation
4. Server-Side Validation Module
5. Security and Encryption Layer
6. Certificate Authority (CA)
7. Transaction Processing Module
8. Logging and Audit Trail

## User Interface (UI):

The user interface is the frontend component where users interact with the internet banking system. It includes elements for registration, login, and transaction initiation. The UI should seamlessly guide users through the process of generating and using client certificates.

## Certificate Generation Module:

This module dynamically generates client certificates upon successful user login. It associates the certificate with user-specific details like mobile numbers, account numbers, and PAN. The generation process should ensure uniqueness and tie the certificate securely to the user's identity.

## One-Time Password (OTP) Generation:

A component responsible for generating one-time passwords (OTPs) when users initiate transactions. The OTP adds an additional layer of security, requiring users to enter a dynamically generated code along with their client certificate during sensitive operations.

## Server-Side Validation Module:

On the server side, a validation module verifies the accuracy of information provided by the user, including the OTP and client certificate. It checks against the stored user data and ensures that both elements are correct before allowing transaction processing.

## Security and Encryption Layer:

This layer handles the encryption and decryption of data during communication between the client and server. It ensures that sensitive

information, including client certificates and OTPs, is securely transmitted, preventing unauthorized access or interception

.

## Certificate Authority (CA):

The Certificate Authority is a trusted third-party entity responsible for issuing and validating client certificates. It plays a crucial role in establishing the authenticity of the certificates, enhancing overall security.

## Transaction Processing Module:

This module manages the execution of financial transactions once the user's identity and provided information, including the OTP and client certificate, are successfully validated. It interacts with the core banking system to ensure the accuracy and security of transactions.

## Logging and Audit Trail:

A logging and audit trail component records and stores relevant activities and events, providing a means for tracking user interactions and system activities. This is essential for security monitoring, compliance, and investigation in case of any suspicious activities.

## PROPERTIES

1. Confidentiality
2. Authentication
3. Uniqueness
4. Dynamic Verification
5. Two-Factor Authentication (2FA)
6. Transaction Integrity
7. Notification and Alerts
8. User-Friendly Installation

## PROPERTIES OF OTP AND CERTIFICATE-BASED SECURITY IN INTERNET BANKING:

**Confidentiality**

Both the client certificate and the OTP contribute to ensuring the confidentiality of user information and transactions. The password-protected client certificate and dynamic nature of OTPs prevent unauthorized access and interception.

## Authentication:

The use of a client certificate and OTP together enhances authentication. The client certificate, linked to user information, authenticates the user's identity, while the OTP provides an additional layer of dynamic verification during transaction initiation.

## Uniqueness:

Client certificates generated based on unique user details and the issuance of one-time passwords create a system with inherent uniqueness. This uniqueness adds to the complexity of the authentication process, reducing the likelihood of fraudulent activities.

## Dynamic Verification:

The issuance and verification of one-time passwords introduce a dynamic element to the authentication process. This dynamic nature, coupled with the time-sensitive validity of OTPs, strengthens security by preventing the reuse of codes.

## Two-Factor Authentication (2FA):

The combination of a password-protected client certificate and the use of OTPs constitutes a two-factor authentication (2FA) system. This adds an extra layer of security by requiring users to provide both something they have (client certificate) and something they know (OTP).

## Transaction Integrity

: The properties of OTPs and client certificates collectively contribute to ensuring the integrity of financial transactions. Validating both the OTP and certificate before transaction approval safeguards against unauthorized access and fraudulent activities.

## Notification and Alerts:

The system's capability to deny transactions with invalid OTPs or certificates and notify clients immediately adds a proactive layer to security. Timely alerts allow users to address any suspicious activity promptly, reducing the risk of financial fraud.

## User-Friendly Installation:

The instruction for clients to install the certificate in web and mobile browsers should be designed to be user-friendly. Ensuring a seamless installation process encourages users to comply with security measures without hindering their experience.

# CHAPTER 2

## LITERATURE SURVEY

In this chapter, an extensive review of literature related to OTP (One-Time Password) with Client Certificate-Based Authentication for secure web transactions is presented. The focus of the survey revolves around exploring authentication methods that combine the use of OTPs and client certificates to ensure robust security in web transactions.

The investigation has identified the significance of OTP and client certificate- based authentication, particularly in enhancing the security of web transactions. The primary protocol of interest is the integration of OTPs and client certificates, ensuring a multi-layered approach to user verification. This combination not only strengthens the authentication process but also contributes to the overall security posture of online transactions.

## SSL Test Suite: SSL Certificate Test Public Key Infrastructure (Merve Melis Şimşek , 2022)

Today, many internet-based applications, especially e-commerce and banking applications, require the transfer of personal data and sensitive data such as credit card information, and in this process, all operations are carried out over the Internet. Users frequently perform these transactions, which require high security, on web sites they access via web browsers. This makes the browser one of the most basic software on the Internet. The security of the communication between the user and the website is provided with SSL certificates, which is used for server authentication. Certificates issued by Certificate Authorities (CA) that have passed international audits must meet certain conditions. The criteria for the issuance of certificates are defined in the Baseline Requirements (BR) document published by the Certificate Authority/Browser (CA/B) Forum, which is accepted as the authority in the

WEB Public Key Infrastructure (WEB PKI) ecosystem. Issuing the certificates in accordance with the defined criteria is not sufficient on its own to establish a secure SSL connection. In order to ensure a secure connection and confirm the identity of the website, the certificate validation task falls to the web browsers with which users interact the most. In this study, a comprehensive SSL certificate public key infrastructure (SSL Test Suite) was established to test the behavior of web browsers against certificates that do not comply with BR requirements. With the designed test suite, it is aimed to analyze the certificate validation behaviors of web browsers effectively.

# CHAPTER 3

## HARDWARE AND SOFTWARE REQUIREMENTS

This chapter provides brief description about the requirements essential for our project.

## HARDWARE REQUIREMENT

Operating system : Linux

Hard disk : 40 GB

RAM : 2 GB (minimum)

## SOFTWARE REQUIREMENT

Web Server : Apache

SSL/TLS Support : mod\_ssl ,OpenSSL

Certificates : SSL Certificate,Client Certificates

# CHAPTER 4

## PROJECT DESCRIPTION

This chapter outlines the scope and objectives of implementing OTP (One-Time Password) with Client Certificate-Based Authentication for Secure Web Transactions. This project aims to enhance the security of web transactions by integrating OTP and client certificates, addressing limitations observed in traditional authentication methods. The Network Simulator serves as the primary tool for the project's simulation and evaluation.

## AIM

This project aims to bolster web transaction security through the implementation of OTP and Client Certificate-Based Authentication. By combining these technologies, it addresses limitations in traditional authentication methods. The Network Simulator will play a crucial role in simulating and evaluating the effectiveness of the proposed security enhancements.

## IMPLEMENTATION PROCESS

## OTP BASED AUTHENTICATION

OTP-based authentication is a security method designed to enhance user verification in digital systems. In this process, a one-time password (OTP) is generated and provided to the user for a single-use or within a short validity period. The generated OTP is delivered through channels like SMS, email, mobile apps, or hardware tokens. During login, the user enters the received OTP along with their traditional credentials, creating a two-step verification process. This approach strengthens security by requiring not only something the user knows (password) but also something they have (the temporary OTP). The system then validates the entered OTP against the generated one, granting access if correct and within the valid timeframe. OTP-based authentication

offers heightened security, especially in scenarios where traditional static passwords may be susceptible to compromise. Its dynamic and time-sensitive nature reduces the risk of unauthorized access and is commonly employed in applications such as online banking and secure websites, contributing to robust user account protection.

## Generation of OTP:

When a user attempts to log in, the system generates a unique one-time password. This password is typically short-lived and valid for a single use or a short duration.

## Delivery of OTP:

The generated OTP is then delivered to the user through a predefined channel. Common delivery methods include SMS, email, mobile apps, or hardware tokens. The goal is to provide the user with a dynamic and secure code that only they can access.

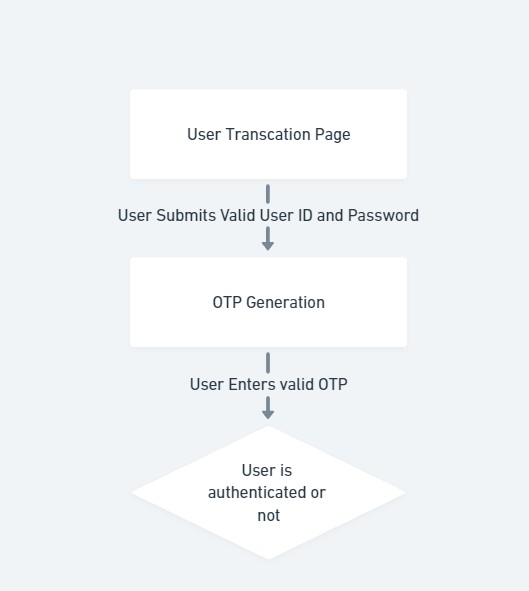
## User Input:

The user enters the received OTP into the login interface along with their regular credentials. This creates a two-step verification process: something the user knows (password) and something the user has (the temporary OTP).

## Validation:

The system validates the entered OTP against the one it generated for that specific login attempt. If the OTP is correct and within the valid timeframe, the user is granted access. Otherwise, access is denied.

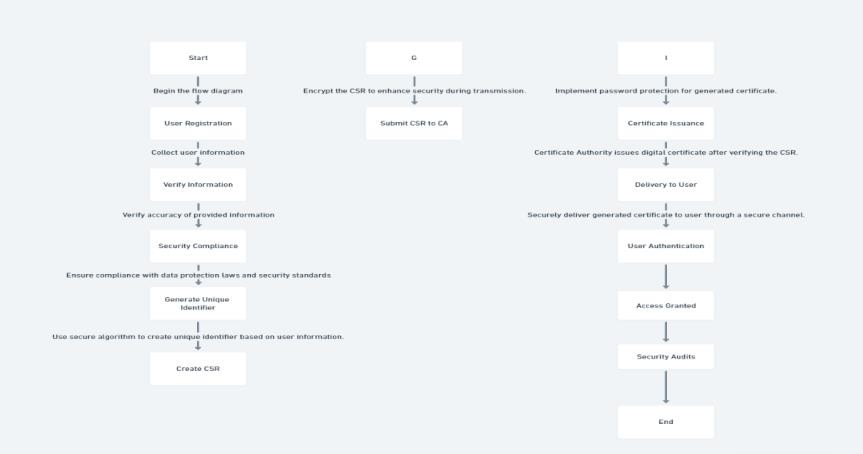
## OTP FLOW CHART



**Figure 4.1 OTP generation**

## STEPS TO GENERATE CERTIFICATE

## FLOW CHART TO GENERATE CERTIFICATE



## Figure 4.2 Certificate Generation

* + 1. **STEP TOGENERATE THE SSL CLIENT AUTHENTICATION WITH SELF SIGNED CA BASED ON APACHE**

## Prerequisites:

* + Server Debian or Ubuntu
  + An up-to-date date System
    - Apt update; apt upgrade
  + Running SSH session as root

## Installation of Apache and OpenSSL:

* + Install Apache web server and OpenSSL. apt install apache2 openssl ssl-cert
  + Activate SSL module for Apache.

## Let's Encrypt Certificate:

* + Obtain a Let's Encrypt certificate for basic SSL connections using Certbot.
    - apt-get install certbot python-certbot-apache
    - certbot –apache

## Generate Own CA:

* + Generate an RSA private key. mkdir openssl\_files

cd ./openssl\_files

* + Create a CA certificate

openssl genrsa -out MyOwnCA.key 4096

## Generate Client Certificates:

* + Generate client key.

openssl genrsa -out MyClientCert1.key 4096

* + Create a certificate signing request (CSR).
  + Generate the client certificate.
  + Convert the certificate to PFX format for browser compatibility.

openssl pkcs12 -export -out MyClientCert1.pfx -inkey MyClientCert1.key -in MyClientCert1.pem

## SSL Authentication Rules:

* + Add SSL authentication rules in the virtual host or .htaccess for directories or websites.

SSLVerifyClient require

* + Configure SSLVerifyClient and SSLVerifyDepth. SSLVerifyDepth 10

## Prepare CA Directory:

* + Create a directory for SSL-CAs. mkdir /etc/ssl\_clientauth\_cas

chown root.www-data /etc/ssl\_clientauth\_cas/ chmod 750 /etc/ssl\_clientauth\_cas/

* + Copy CA certificate and create symlinks with certificate hashes. cp /root/openssl\_files/MyOwnCA.pem /etc/ssl\_clientauth\_cas/ c\_rehash /etc/ssl\_clientauth\_cas/

## Update Apache 2 Config:

* + Update Apache configuration to use CA certificates from the specified directory.

LogLevel debug

openssl verify -CApath /etc/ssl\_clientauth\_cas/ YourCertificateToCheck.pem

## Restart Apache:

* + Restart Apache to apply the changes.

## Verification:

* + Use OpenSSL to verify the SSL client certificate.

## FLOW CHART FOR CERTICATE SETUP



**Figure 4.3 Flow Chart For Setup Phase For Certificate**

## SCOPE OF THE PROJECT

The scope of this project encompasses the comprehensive enhancement of web transaction security through the implementation of a sophisticated authentication system, integrating One-Time Passwords (OTPs) and Client Certificate-Based Authentication. The project aims to address inherent limitations in conventional authentication methods by introducing a dynamic

and secure approach to user verification. The focus is on fortifying the security posture of web transactions against unauthorized access and potential threats. The project's reach extends to the development of a robust security framework that ensures not only the confidentiality and integrity of sensitive data but also a seamless and user-friendly transaction experience. Additionally, the project incorporates the utilization of a Network Simulator to simulate and evaluate the proposed security enhancements, providing a controlled environment for assessing the effectiveness of the implemented measures. The envisioned outcome is a heightened level of security in online transactions, fostering trust and confidence among users while mitigating risks associated with unauthorized access and fraudulent activities.

## STEPS TO INTEGRATE OTP AND CLIENT CERTIFICATE- BASED AUTHENTICATION

Integrating One-Time Passwords (OTPs) and Client Certificate-Based Authentication involves several steps to ensure a seamless and secure implementation. Below is a detailed explanation of each step:

## System Assessment:

Evaluate the existing authentication infrastructure and assess the compatibility of the system with OTP and Client Certificate-Based Authentication.

## Define Authentication Policies:

Clearly define the authentication policies, including the scenarios where OTP and Client Certificate-Based Authentication will be employed. Specify the criteria for the use of each method.

## Generate Client Certificates:

Implement a process to generate unique client certificates for users. This involves utilizing a Certificate Authority (CA) to issue certificates after verifying the identity of each user

## Integrate Certificate Authority (CA):

Establish a connection with a trusted CA for certificate issuance and validation. Ensure that the CA is integrated seamlessly with the authentication system.

## Configure Web Server:

Configure the web server to recognize and validate client certificates. This may involve updating server settings and defining rules for certificate acceptance.

## Implement SSL/TLS:

Enable SSL/TLS (Secure Sockets Layer/Transport Layer Security) to encrypt the communication between the client and server. This ensures the secure transmission of client certificates.

## OTP Generation Setup:

Set up an OTP generation system that dynamically generates one-time passwords. Choose a method for OTP delivery, such as SMS, email, or a mobile app.

## User Registration and Enrollment:

Implement a process for user registration and enrollment in the authentication system. This includes the enrollment of client certificates and the registration of mobile numbers or other contact methods for OTP delivery.

## User Authentication Flow:

Define the flow of user authentication, considering scenarios where either OTP or Client Certificate-Based Authentication will be utilized. Ensure a user-friendly interface for OTP entry.

## Logging and Monitoring:

Implement logging and monitoring mechanisms to track authentication attempts, including successful and unsuccessful logins. This helps in identifying potential security issues and analyzing user behavior.

## Testing and Quality Assurance:

Conduct thorough testing to ensure the seamless integration of OTP and Client Certificate-Based Authentication. Perform quality assurance checks for potential vulnerabilities and user experience issues.

## User Training and Communication:

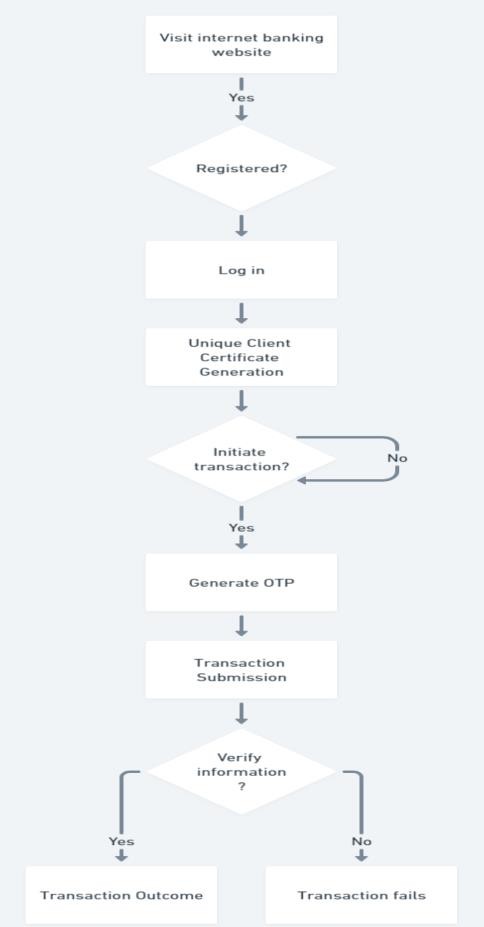
Provide clear communication and training to users regarding the new authentication methods. Explain the benefits and steps involved in using OTP and client certificates.

## Rollout and Deployment:

Gradually roll out the integrated authentication system, monitoring its performance and addressing any issues that arise during deployment.

## Continuous Improvement:

Establish a framework for continuous improvement. Collect user feedback, monitor system performance, and be prepared to make adjustments to enhance security and user experience over time



**Figure 4.4 Flow Graph OTP-Certificate**

# CHAPTER 5

## METHODOLOGY

## openSSL:

OpenSSL is an open-source software library that provides a robust implementation of various cryptographic functions and protocols. Its primary purpose is to secure data communication over computer networks, particularly the internet. Here's a brief overview of OpenSSL's key features and functionalities

## Cryptography Functions:

OpenSSL supports a wide range of cryptographic algorithms, including symmetric and asymmetric encryption, digital signatures, and hash functions.

## SSL/TLS Protocol Implementation:

OpenSSL implements the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, essential for securing data transmission over networks.

## Certificate Authority (CA) Operations:

OpenSSL facilitates the creation and management of digital certificates, including the ability to act as a Certificate Authority for issuing and signing certificates.

## Public Key Infrastructure (PKI) Support:

OpenSSL supports the infrastructure for managing public and private keys, enabling secure key exchange, digital signatures, and certificate verification.

## Secure Socket Layer (SSL) and Transport Layer Security (TLS) Libraries:

OpenSSL includes libraries for implementing SSL and TLS, allowing developers to create secure network communication in their applications.

## Command-Line Tools:

OpenSSL provides a set of command-line tools for performing cryptographic operations, generating key pairs, creating certificates, and more.

## Cross-Platform Compatibility:

OpenSSL is designed to run on various operating systems, including Unix-like systems (Linux, BSD), Microsoft Windows, and macOS.

## Open Source and Community-Driven:

As an open-source project, OpenSSL is continuously developed and maintained by a community of contributors. This collaborative approach ensures updates, bug fixes, and security enhancements.

## SSL/TLS Certificates:

OpenSSL is commonly used to generate and manage SSL/TLS certificates, allowing websites to establish secure connections with users' browsers.

## Cryptographic Standards Compliance:

OpenSSL adheres to various cryptographic standards and is widely used in industry applications, securing data in web servers, applications, and network devices.

## 5.2 SHA-256

SHA-256, which stands for Secure Hash Algorithm 256-bit, is a widely used cryptographic hash function that belongs to the SHA-2 family. It was designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). SHA-256 is commonly used for various security applications and protocols, including digital signatures, certificate generation, and blockchain.

## Message Input:

SHA-256 takes an input message of any length and processes it in fixed- size blocks (512 bits or 64 bytes).

## Padding:

If necessary, the message is padded to ensure its length is a multiple ofthe block size. Padding includes the original message length, ensuring that the hash is unique for different inputs.

## Initialization Vectors:

SHA-256 uses specific constants (known as "initialization vectors") as the starting point for its computations.

## Message Schedule:

The message block is divided into 16 words, and additional words are derived using a specific schedule based on the original message.

## Compression Function:

SHA-256 employs a series of bitwise operations, logical functions (AND, OR, XOR), and modular additions to compress the data in each block into a fixed-size output.

## Round Operations:

The compression function is applied multiple times in rounds. SHA-256 consists of 64 rounds, each with a different set of constants and operations.

## Hash Value:

* + - * After processing all the blocks and rounds, the final hash value isproduced. The hash is a fixed-size output of 256 bits (32 bytes).

## Key Characteristics of SHA-256:

* + - * **Security Strength:**

SHA-256 is designed to be a collision-resistant hash function. It means that it should be computationally infeasible to find two different inputs that produce the same hash value.

## Deterministic:

For the same input, SHA-256 always produces the same hash output. This determinism is crucial for cryptographic applications.

## Fast Computation:

While SHA-256 is secure, it is also designed to be computationally efficient, allowing it to generate hash values relatively quickly.

# CHAPTER 6

## CONCLUSIONS AND FUTURE ENHANCEMENTS

This chapter concludes about the efficiency of OTP WITH CLIENT CERTFICATE BASED AUTHENTICATION FOR SECURE WEB

TRANSACTION and the future enhancement of our project.

## Conclusions

In conclusion, the integration of One-Time Passwords (OTPs) and Client Certificate-Based Authentication stands as a commendable stride towards bolstering web transaction security. This project has successfully addressed the limitations of traditional authentication methods by introducing a dynamic and multifaceted approach. The combined security framework not only enhances resistance against unauthorized access and fraud but also ensures a user-friendly and seamless transaction experience. The use of a Network Simulator has facilitated a controlled evaluation of the implemented security measures, providing valuable insights into the effectiveness of the enhancements.The project's impact is not confined to immediate security benefits; it extends to fostering user trust, which is paramount in online transactions. The successful implementation of OTPs and client certificates signifies a commitment to safeguarding user data and transaction integrity. As we conclude, the project sets the stage for a new standard in web transaction security, emphasizing the importance of continuous adaptation and innovation to stay ahead of evolving cyber threats.

## Future Enhancements

Looking towards the future, there are promising avenues for further enhancement of the web transaction security system implemented through the integration of One-Time Passwords (OTPs) and Client Certificate-Based Authentication. One potential avenue is the integration of advanced biometric authentication methods, such as fingerprint or facial recognition, to provide an additional layer of identity verification and user convenience. Additionally, exploring the application of machine learning algorithms for anomaly detection could enable the system to dynamically adapt to evolving security threats, identifying and responding to unusual user behavior patterns.

The incorporation of blockchain technology stands as another promising future enhancement, ensuring the immutability and enhanced security of user authentication records. This could provide a tamper-resistant audit trail, adding a layer of transparency and trust to the authentication process. Further advancements may involve the implementation of adaptive authentication mechanisms that dynamically adjust security measures based on user behavior and risk factors, tailoring the level of security to the specific context.

Enhanced monitoring and reporting capabilities could also be developed, offering real-time insights into authentication events and enabling proactive responses to potential security incidents. Extending the authentication framework to cover a broader range of platforms, including mobile applications, is essential to ensure a consistent and secure user experience across various channels. Moreover, comprehensive user education programs could be launched to raise awareness about emerging cyber threats and promote best practices for secure online behavior.

* 1. **APPENDICIES SORCE CODE:**

# Login.js

import React, { useState } from 'react'; import axios from 'axios';

import { Link, useNavigate } from 'react-router-dom'; import './login.css';

function Login() {

const [username, setUsername] = useState(''); const [pwd, setPwd] = useState('');

const [loginStatus, setLoginStatus] = useState(''); const navigate = useNavigate();

const handleLogin = (event) => { event.preventDefault(); axios.post('https://172.16.23.161:3000/login', { username: username,

pwd: pwd,

}).then((response) => {

if (response.data.message) {

// setLoginStatus(response.data.message); alert("Incorrect Username or Password");

} else { navigate('/otpverify');

}

})

.catch(error => { console.error('Axios Error:', error);

});

}

return (

<div className='body'>

<div className="vh-100 d-flex justify-content-center align-items-center ">

<div className="custom-form">

<h2 className="text-center mb-4 text-primary">Login to Bank server</h2>

<hr></hr>

<form onSubmit={handleLogin}>

<div className="mb-3">

<label htmlFor="exampleInputEmail1" className="form-label">User Name</label>

<input type="text" className="form-control border border-primary" value={username}

onChange={(event) => setUsername(event.target.value)} />

</div>

<div className="mb-3">

<label htmlFor="exampleInputPassword1" className="form- label">Password</label>

<input type="password" className="form-control border border-primary" value={pwd}

onChange={(event) => setPwd(event.target.value)} />

</div>

<div className="d-grid">

<button className="btn btn-primary" type="submit">Login</button>

<h1>{loginStatus}</h1>

</div>

</form>

<div className='text'>

<div className="mt-3 text-center">

<p className="mb-0">Don't have an account? </p>

<Link to="/register" className="btn btn-primary">Register</Link>

</div>

</div>

</div>

</div>

</div>

);

}

export default Login;

# Register.js

import React, { useState } from 'react'; import axios from 'axios';

import './login.css';

import 'bootstrap/dist/css/bootstrap.css';

import { useNavigate } from 'react-router-dom'; import { Link } from 'react-router-dom'; function Register() {

const [username, setUsername] = useState(''); const [password, setPassword] = useState(''); const [name, setName] = useState('');

const [gmail, setGmail] = useState(''); const [panNum, setPanNum] = useState(''); const [aadhar, setAadhar] = useState(''); const [mobile, setMobile] = useState(''); const [message, setMessage] = useState(''); const navigate = useNavigate();

const handleInsertSubmit = async (event) => { event.preventDefault();

try {

const insertResponse = await axios.post('https://172.16.23.161:3000/insert', { username: username,

password: password, name: name,

gmail: gmail, pan\_num: panNum, aadhar: aadhar, mobile: mobile,

});

setMessage(insertResponse.data.message); navigate('/');

} catch (error) { console.error(error); setMessage('Error registering user');

}

};

return (

<div className="vh-100 d-flex justify-content-center align-items-center">

<div className="col-md-5 p-5 shadow-sm border rounded-5">

<h1 className="text-center mb-4 text-primary">Register with Bank Server</h1>

<form onSubmit={handleInsertSubmit}>

<div className="form-group">

<input type="text"

className="form-control" placeholder="Username" value={username}

onChange={(event) => setUsername(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<input type="password"

className="form-control" placeholder="Password" value={password}

onChange={(event) => setPassword(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<input type="text"

className="form-control" placeholder="Name" value={name}

onChange={(event) => setName(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<input type="email"

className="form-control" placeholder="Email (Gmail)" value={gmail}

onChange={(event) => setGmail(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<input type="text"

className="form-control"

placeholder="PAN Number" value={panNum}

onChange={(event) => setPanNum(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<input type="text"

className="form-control" placeholder="Aadhar Number" value={aadhar}

onChange={(event) => setAadhar(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<input type="tel"

className="form-control" placeholder="Mobile Number" value={mobile}

onChange={(event) => setMobile(event.target.value)} required

/>

</div>

<br />

<div className="form-group">

<button type="submit" className="btn btn-primary w-100">Register</button>

</div>

{message && <p className="text-center">{message}</p>}

</form>

<br />

<Link to="/">

<button className="btn btn-primary w-100">LOGIN</button>

</Link>

</div>

</div>

);

export default Register;

# otpverify.js

import React, { useState, useEffect } from 'react'; import axios from 'axios';

//import { sha512 } from 'crypto-hash'; import CryptoJS from 'crypto-js'; import html2canvas from 'html2canvas'; import jsPDF from 'jspdf';

import { useNavigate } from 'react-router-dom';

//const crypto = require('crypto'); function App() {

const [email, setEmail1] = useState('');

const [userDetails, setUserDetails] = useState(null);

const [hashedDetails, setHashedDetails] = useState(''); const [otp, setOtp] = useState('');

const [message, setMessage] = useState(''); const navigate = useNavigate(); useEffect(() => {

// Fetch user details when the component mounts if (email) {

axios.get(`https://172.16.23.161:3000/get-user-details?email=${email}`)

.then(response => { setUserDetails(response.data);

})

.catch(error => { console.error(error);

setMessage('Error fetching user details');

});

}

}, [email]);

const generateCertificate = () => {

const certificateElement = document.getElementById('certificate');

if (!certificateElement) { console.error('Certificate element not found.'); return;

}

// Ensure the element is visible certificateElement.style.display = 'block';

html2canvas(certificateElement, { scale: 2 }).then(canvas => { try {

const imgData = canvas.toDataURL('image/png'); const pdf = new jsPDF('portrait', 'px', 'a4'); pdf.addImage(imgData, 'PNG', 0, 0);

// Convert the PDF to a Base64 string

const pdfBase64 = pdf.output('datauristring');

// Protect the PDF with a password (hashedDetails as the password)

const hashedDetails = getHashedUserDetails(); // You should define this function to get the hashed details.

pdf.setEncryption(hashedDetails, hashedDetails, [1, 2, 3], 'AES-256');

// Store the password-protected PDF in local storage localStorage.setItem('certificatePDF', pdf.output('datauristring'));

// Provide a link or button to retrieve or download the PDF

alert('PDF has been stored in local storage and protected with the hashed user details. You can retrieve or download it later.');

} catch (error) {

console.error('Error generating PDF:', error);

// Handle the error as needed

} finally {

// Restore the element's visibility to its original state certificateElement.style.display = 'none';

}

});

};

// Define a function to get the hashed user details (similar to hashUserDetails) const getHashedUserDetails = () => {

if (userDetails) {

const { pan\_num, aadhar, mobile } = userDetails;

const detailsToHash = `${pan\_num}${aadhar}${mobile}`; return CryptoJS.SHA512(detailsToHash).toString();

} else { return '';

}

};

const hashUserDetails = () => { if (userDetails) {

const { pan\_num, aadhar, mobile } = userDetails;

// Concatenate user details into a single string

const detailsToHash = `${pan\_num}${aadhar}${mobile}`;

// Use crypto-js to compute the SHA-512 hash

const hashedDetails = CryptoJS.SHA512(detailsToHash).toString(); setHashedDetails(hashedDetails);

} else {

setMessage('User details not available');

}

};

const generateOTP = () => { axios.post('https://172.16.23.161:3000/generate-otp', { email: email

}).then(response => { setMessage(response.data.message);

})

.catch(error => { console.error(error); setMessage('Error generating OTP');

});

};

const logout = () => {

// Clear the user's session (e.g., remove tokens or user data from local storage)

// Redirect the user to the login page

navigate('/'); // Use the navigate function to change the route

};

const verifyOTP = () => { axios.post('https://172.16.23.161:3000/verify-otp', { email, otp })

.then(response => { setMessage(response.data.message);

})

.catch(error => { console.error(error); setMessage('Error verifying OTP');

});

};

return (

<div className="App">

<header className="App-header">

<h1>Welcome to My React App</h1>

<p>Enjoy your React journey!</p>

<form>

<label>Enter your email: </label>

<input type="email" value={email}

onChange={(e) => setEmail1(e.target.value)}

/>

<button type="button" onClick={generateOTP}>Generate OTP</button>

</form>

<form>

<label>Enter the OTP: </label>

<input type="text" value={otp}

onChange={(e) => setOtp(e.target.value)}

/>

<button type="button" onClick={verifyOTP}>Verify OTP</button>

</form>

<button type="button" onClick={hashUserDetails}>Hash Details</button>

<button type="button" onClick={generateCertificate}>Generate Certificate</button>

<p>User Details: {JSON.stringify(userDetails)}</p>

<p>Hashed User Details: {hashedDetails}</p>

<button type="button" onClick={logout}>Logout</button>

<p>{message}</p>

</header>

<div id="certificate" style={{ display: 'none' }}>

<h1>Certificate of Achievement</h1>

<p>Aadhar: {userDetails?.aadhar}</p>

<p>PAN Number: {userDetails?.pan\_num}</p>

<p>Mobile Number: {userDetails?.mobile}</p>

</div>

</div>

);

}

export default App;

## Server.js

const express = require('express');

const bodyParser = require('body-parser'); const cors = require('cors');

const nodemailer = require('nodemailer'); const mysql = require('mysql');

const html2canvas = require('html2canvas'); const jsPDF = require('jspdf');

//const cryptoHash = require('crypto-hash'); const app = express();

var count=0;

// Define the functionalities app.use(cors()); app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true })); app.use(express.static('public'));

app.use((req, res, next) => {

res.setHeader('Content-Security-Policy', "default-src 'none'; font-src 'self' http://localhost:3002/confirm");

next();

});

const connection = mysql.createConnection({ host: 'localhost',

user: 'root',

password: "", database: 'bank'

});

app.post('/login', (req, res) => { const uname = req.body.username; const pwd = req.body.pwd; connection.query(

"SELECT \* FROM login WHERE username = ? AND password = ?", [uname, pwd],

(err, result) => { if (err) {

console.error('MySQL Error:', err); res.status(500).send({ message: 'Internal Server Error' });

}

if (Array.isArray(result) && result.length > 0) { console.log(uname);

res.send(result);

} else {

res.send({ message: "error" });

}

}

);

});

// Handle the insert route and send the verification email app.post('/insert', (req, res) => {

const username = req.body.username; const password = req.body.password; const name = req.body.name;

const gmail = req.body.gmail;

const panNum = req.body.pan\_num; const aadhar = req.body.aadhar; const amount = req.body.amount; const mobile = req.body.mobile;

// Check if the username already exists in the 'login' table

const checkUsernameQuery = 'SELECT \* FROM login WHERE username = ?'; connection.query(checkUsernameQuery, [username], (err, result) => {

if (err) { console.log(err);

res.send({ message: 'Error checking for existing username' });

} else {

console.log('Result:', result); // Add this line to inspect the result if (result.length > 0) {

// Username already exists, send a message res.send({ message: 'Username already exists' });

} else {

// Username is not present, proceed with the insertion

const insertLoginQuery = `INSERT INTO login (username, password) VALUES (?, ?)`;

connection.query(insertLoginQuery, [username, password], (err, result) => { if (err) {

console.log(err);

res.send({ message: 'Error inserting data into login table' });

} else {

// Insert data into the 'user\_data' table

const insertUserDataQuery = `INSERT INTO user\_data (username, email, pan\_num, aadhar\_num, mobile\_num) VALUES (?, ?, ?, ?, ?)`;

connection.query(insertUserDataQuery, [username, gmail, panNum, aadhar, mobile], (err, result) => {

if (err) { console.log(err);

res.send({ message: 'Error inserting data into user\_data table' });

} else {

console.log('Data inserted into login and user\_data tables'); res.send({ message: 'Registration successful' });

}

});

}

});

}

};

});

})

const transporter = nodemailer.createTransport({ service: 'Outlook365', // e.g., 'Gmail'

auth: {

user: 'kannan.20it@sonatech.ac.in', pass: 'sm@\_2003'

}

});

function generateOTP() {

return Math.floor(100000 + Math.random() \* 900000);

}

const otps = new Map(); app.post('/generate-otp', (req, res) => { const email = req.body.email;

const otp = generateOTP();

const timestamp = Date.now(); // Get the current timestamp in milliseconds

// Store the OTP with email and timestamp association otps.set(email, { otp: otp.toString(), timestamp: timestamp });

// Send the OTP via email const mailOptions = {

from: 'kannan.20it@sonatech.ac.in', to: email,

subject: 'Your OTP',

text: `Your OTP is: ${otp}`

};

transporter.sendMail(mailOptions, (error, info) => { if (error) {

console.error('Email Error:', error); res.status(500).send({ message: 'Internal Server Error' });

} else {

res.send({ message: 'OTP generated and sent via email' });

}

});

});

app.post('/verify-otp', (req, res) => { const email = req.body.email;

const userOTP = req.body.otp; const storedData = otps.get(email);

if (storedData && userOTP === storedData.otp) {

// Check if the OTP is still valid (within 5 minutes) const currentTime = Date.now();

const otpTimestamp = storedData.timestamp;

const timeDifference = currentTime - otpTimestamp;

const validityPeriod = 5 \* 60 \* 1000; // 5 minutes in milliseconds if (timeDifference <= validityPeriod) {

res.send({ message: 'OTP is correct and within the validity period!' });

} else {

res.send({ message: 'OTP is correct but has expired. Please request a new OTP.'

});

}

} else {

res.send({ message: 'OTP is incorrect or expired. Please try again.' });

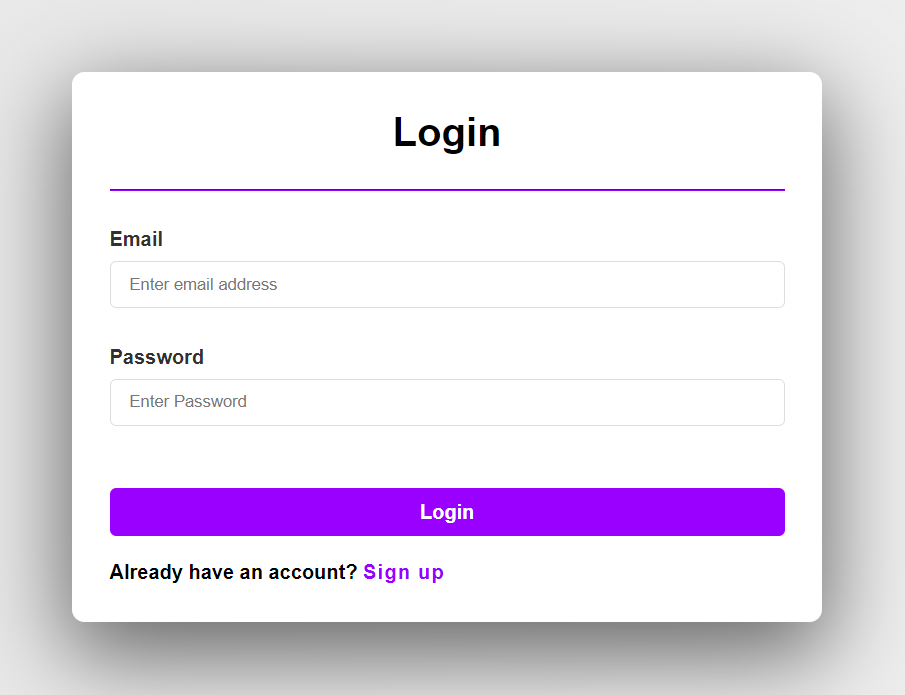
}

});

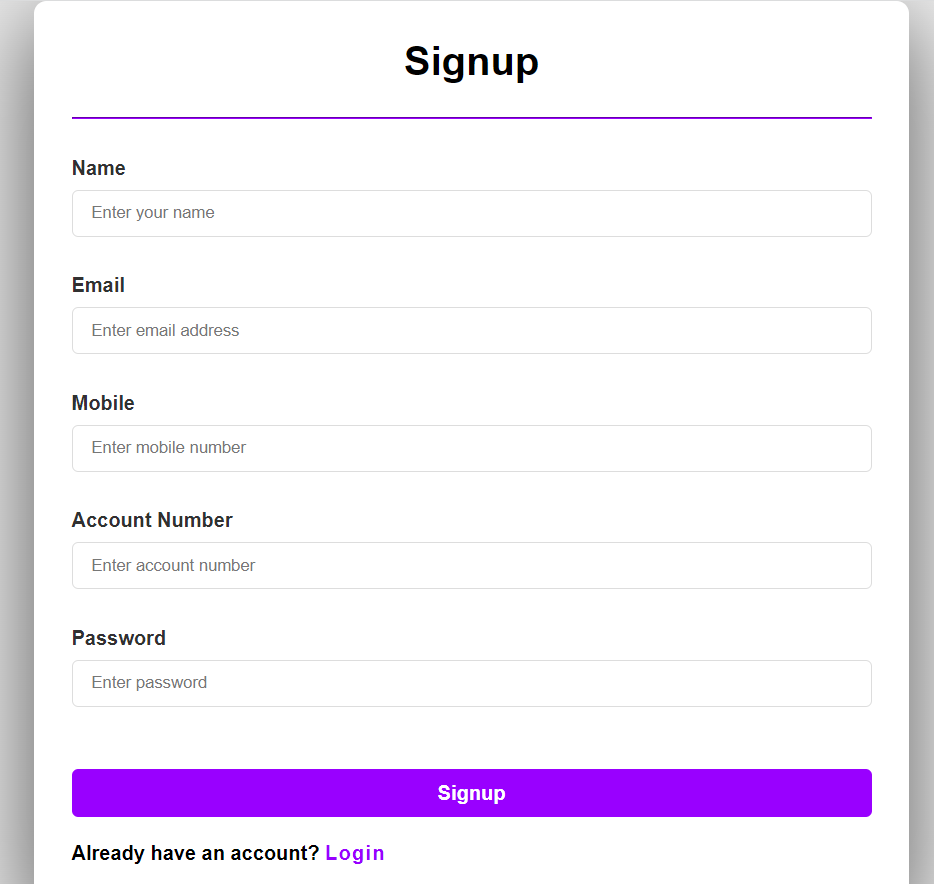
// Start the server app.listen(3000, () => { console.log('Server is running');

}

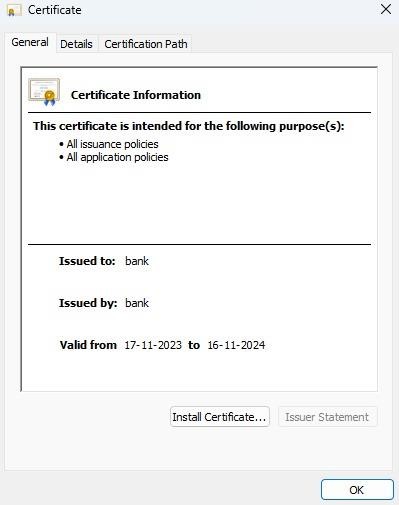
## Screenshots:



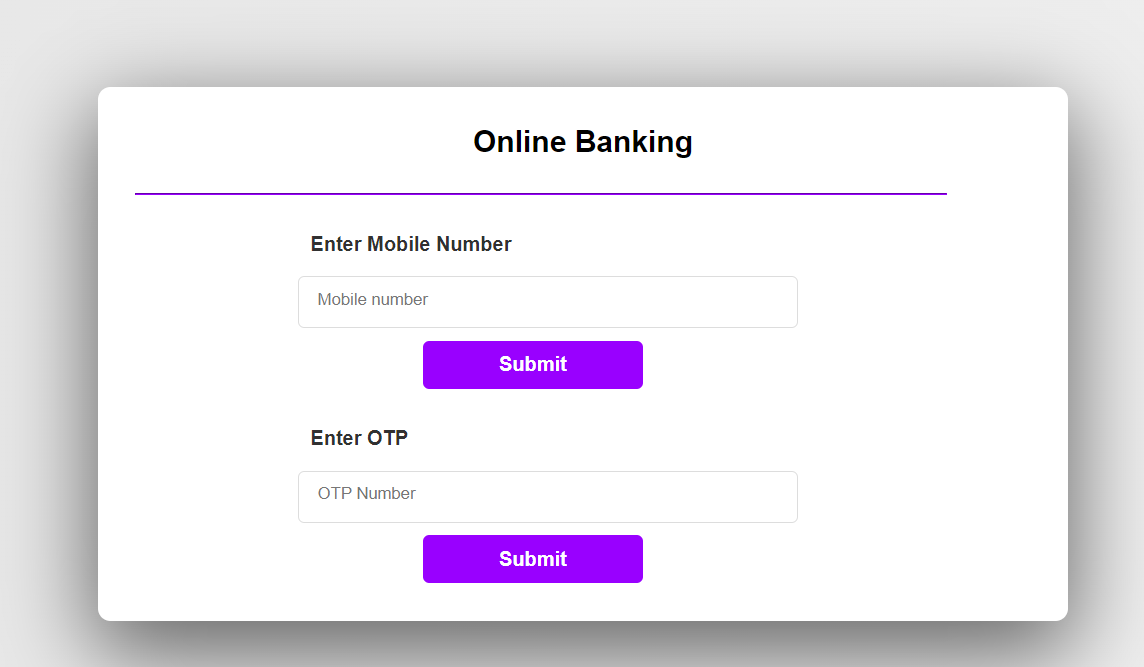
**Figure 6.3.1 Login Page**



## Figure 6.3.2 Registration Page



**Figure 6.3.3 Based On User Information Generate A Certificate**



## Figure 6.3.4 OTP Main Page For Transactions

## REFERENCES

1. L. Zhang et al., "Analysis of SSL certificate reissues and revocations in the wake of heartbleed", Proc. 2014 Conf. Internet Meas. Conf., pp. 489- 502, 2014.
2. . D. Cooper, "Internet x. 509 public key infrastructure certificate and certificate revocation list (CRL) profile", 2008.
3. F. F. Elwailly, C. Gentry and Z. Ramzan, "Quasimodo: Efficient certificate validation and revocation", Proc. Int. Workshop Public Key Cryptography, pp. 375-388, 2004.
4. T. P. Hormann, K. Wrona and S. Holtmanns, "Evaluation of certificate validation mechanisms", Comput. Commun., vol. 29, no. 3, pp. 291-305, 2006.
5. E. Stark, L.-S. Huang, D. Israni, C. Jackson and D. Boneh, "The case for prefetching and prevalidating TLS server certificates", Proc. 19th Annu. Netw. Distrib. Syst. Security Conf., pp. 1-12, 2012.
6. D. Wendlandt, D. G. Andersen and A. Perrig, "Perspectives: Improving SSH-style host authentication with multi-path probing", Proc. USENIX Annu. Tech. Conf., pp. 321-334, 2008

.