COLLEGE CODE	: 9605 : CAPE INSTITUTE OF TECHNOLOGY
COLLEGE NAME	: BE.CSE(AI&ML)
DEPARTMENT	:5D1A9FD9E079DF5B80B423DFA20A5641
STUDENT NM-ID	
ROLL NO	: 960523148025
DATE	: 27-10-2025

Completed the project named as Phase - 5

TECHNOLOGY PROJECT NAME : IBM -FE- LOGIN AUTHENTICATION SYSTEM

SUBMITTED BY,

NAME	:L.BALA SUNIL
MOBILE NO	:8248890462

IBM-FE LOGIN AUTHENTICATION SYSTEM

Project Demonstration & Documentation

Objectives

The primary objective of the Login Authentication System project demonstration and documentation is to provide a comprehensive and professional overview of the system's functionality, design, and implementation. This section focuses on illustrating how the authentication process ensures secure and seamless user access through efficient registration, login, and password management modules. The demonstration highlights the integration of data validation, encryption techniques, and session handling to maintain the confidentiality and integrity of user credentials. Furthermore, the documentation aims to present the complete workflow of the project, explaining both the technical and functional aspects in a clear and structured manner. It serves as a reference for developers, evaluators, and users to understand the system's logic, usability, and overall performance in real-world scenarios.

1.Final Demo walkthrough

1.1 Overview of the Project

The Login System Authentication project is designed to provide a secure, reliable, and user-friendly authentication process. This system ensures that unauthorized access is prevented, sensitive data is protected, and users experience a smooth login process.

The project comprises three main modules:

1. Registration Module – This module allows new users to register in the system by providing a username, email, and password. Passwords are validated for strength and securely stored using encryption and hashing techniques to prevent unauthorized access.

2. Login Module – Registered users can log in by entering their credentials. The system verifies the username and password combination. Upon correct input, users gain access to the system; otherwise, appropriate error messages are displayed. Optional Multi-Factor Authentication (MFA) can be implemented to add an extra layer of security.

3. Password Recovery Module – This module assists users who have forgotten their passwords. Through OTP or email verification, users can securely reset their passwords, enhancing both security and user convenience.

Key Features:

Strong authentication with encrypted and hashed passwords.

Secure session management with automatic session timeout.

User-friendly interface for effortless registration, login, and password recovery.

Scalability for future enhancements such as biometric authentication or AI-based anomaly detection.

Purpose of the Walkthrough:

To demonstrate the complete flow of each module step by step.

To explain the sequence: Registration → Login → Dashboard → Logout.

To practically showcase security features such as encryption, session management, and MFA.

To provide a clear understanding of the project functionality to users and evaluator

1.2 Working Principle

The Login System Authentication operates on a structured and secure workflow to ensure proper verification of users while maintaining data confidentiality. The system follows a step-by-step process for registration, login, and password recovery.

1. User Registration:

New users provide essential details such as username, email, and password.

The system validates the inputs for correctness, email format, and password strength.

Passwords are hashed and encrypted before storing them in the database to prevent unauthorized access.

Upon successful registration, the user receives a confirmation, allowing them to proceed to the login module.

2. UserLogin:

Registered users enter their credentials (username and password).

The system verifies the credentials against the database records.

Correct credentials allow access to the user dashboard, while incorrect attempts trigger error messages.
Optional Multi-Factor Authentication (MFA) may be implemented, where an OTP is sent via email or SMS for additional security.

3. Session Management:

Once logged in, the system maintains a secure session for the user.

Inactivity triggers automatic session timeout to prevent unauthorized usage.

Users can manually log out, which immediately ends the session and ensures data security.

4. Password Recovery:

In case of forgotten passwords, users can request a password reset.

The system verifies the user through email or security questions.

After verification, users can create a new password, which is securely updated in the database.

5. Security Implementation:

All sensitive information, including passwords and session tokens, are encrypted.

The system is designed to handle multiple simultaneous users securely.

Invalid login attempts are logged, and repeated failures may temporarily lock the account to prevent brute force attacks

1.3 Demo Flow

The Demo Flow section provides a step-by-step walkthrough of how the Login System Authentication operates during a practical demonstration. It showcases the sequence of actions, user interactions, and system responses.

1. System Launch:

The user opens the login system application or web interface.

A welcome screen is displayed with options: Register, Login, and Forgot Password.

2. Registration Process (If New User):

User clicks “Register” and fills in required details: username, email, password, confirm password.

System validates the entries for format, password strength, and uniqueness.

Upon successful validation, the password is hashed and encrypted, and user information is stored securely in the database.

A confirmation message is displayed, and the user is redirected to the login page.

3. Login Process (For Registered User):

User selects “Login” and enters username and password.

System verifies credentials against stored database values.

Successful Login: User is redirected to the dashboard/home page.

Failed Login: System displays an error message. Multiple failed attempts may trigger account lock or security measures.

4. Multi-Factor Authentication (Optional):

If MFA is enabled, an OTP is sent to the user’s email or mobile number.

User enters the OTP to complete authentication.

Successful verification grants access to the system.

5. Dashboard Interaction:

After login, the user can access personal profile information, settings, and system functionalities.

System maintains a secure session with automatic timeout in case of inactivity.

6. Forgot Password Flow:

User clicks “Forgot Password”.

System verifies the user via email or security question.

User sets a new password, which is encrypted and updated in the database.

Confirmation message displayed; user can now log in with the new password.

7. Logout Process:

User selects “Logout” to terminate the session.

System ensures all session data is cleared, maintaining security and privacy force attacks.

1.4 Features Demonstration

The Features Demonstration section highlights the key functionalities and security features of the Login System Authentication project. This section provides a practical view of how the system ensures secure and reliable user authentication.

1. Registration Feature:

Demonstrates how new users can register securely.

Shows input validation for username, email format, and password strength.

Demonstrates password encryption and secure storage in the database.

Highlights confirmation messages after successful registration.

2. Login Feature:

Shows how registered users log in using credentials.

Demonstrates verification against stored hashed passwords.

Highlights error messages for incorrect username or password.

Optional demonstration of Multi-Factor Authentication (MFA) through OTP.

3. Password Recovery Feature:

Demonstrates the Forgot Password process.

Shows email verification or security question workflow.

Demonstrates how users can reset and securely update their password.

4. Dashboard Feature:

Shows the user interface after successful login.

Demonstrates personalized user profile, settings, and logout options.

Highlights session management, including automatic logout for inactivity.

5. Security Features:

Password Encryption: Demonstrates how passwords are stored securely.

Session Management: Demonstrates secure handling of active sessions.

Invalid Login Attempts: Shows account lock mechanism after multiple failed attempts.

Optional MFA: Demonstrates extra layer of security with OTP verification.

6. User Experience and Usability:

Demonstrates intuitive and easy-to-use interface.

Highlights smooth navigation between registration, login, recovery, and dashboard modules.

Shows real-time validation messages to improve user interaction.

2Project Report

2.1 Abstract

The Login System Authentication project is designed to develop a secure, reliable, and user-friendly authentication system. With the growing need for data security and privacy in digital applications, this project focuses on ensuring safe access for authorized users while preventing unauthorized access.

The system comprises three main modules: Registration, Login, and Password Recovery, supported by a secure dashboard and session management mechanism. Advanced security measures such as password encryption, hashing, and optional multi-factor authentication (MFA) are implemented to protect sensitive user information.

This project also emphasizes user convenience and smooth interaction, providing easy registration, efficient login, and straightforward password recovery processes. The system is designed to be scalable and flexible, allowing for future enhancements like biometric authentication or AI-based anomaly detection.

The demo walkthrough highlights the complete functionality of the system, demonstrating registration, login, password recovery, and security features step-by-step. Through this project, the importance of secure authentication in modern digital environments is illustrated, and the objectives of data protection, usability, and reliability are achieved.

Keywords: Login System, Authentication, Security, Multi-Factor Authentication, Password Encryption, User Verification, Session Management.

2.2 Introduction

In today’s digitalera, secure user authentication is one of the most critical aspects of any software application or online platform. With the increasing amount of sensitive information stored and processed digitally, ensuring that only authorized users gain access is essential for maintaining data privacy, system integrity, and user trust.

The Login System Authentication project aims to design and implement a robust and reliable authentication system that provides secure access while maintaining a smooth and user-friendly interface. This system serves as a primary barrier against unauthorized access, hacking attempts, and data breaches, which are common threats in modern applications.

Key Motivations for the Project:

Security Concerns: Traditional login systems without encryption or multi-layer verification are vulnerable to attacks such as brute force, session hijacking, and credential theft.

User Convenience: Users expect a fast, simple, and seamless login experience without compromising security.

Data Protection: Sensitive user data such as passwords, personal details, and session information must be stored and transmitted securely.

Scalability: Modern applications require systems that can handle multiple users, future enhancements like biometric authentication, and integration with AI-based anomaly detection.

Scope of the Project:

Implementation of secure registration, login, and password recovery modules.

Use of encryption, hashing, and optional Multi-Factor Authentication (MFA) to protect user credentials.

Session management and automatic logout features to prevent unauthorized access.

User-friendly dashboard and interface design for easy interaction.

2.3 Problem Definition

With the rapid growth of digital applications and online platforms, security of user information has become a major concern. Many traditional login systems face multiple challenges due to weak authentication mechanisms, poor data protection practices, and lack of advanced security features.

Key Problems Identified:

Unauthorized Access:

Many existing systems allow users to access sensitive data with weak passwords or single-layer authentication, making them vulnerable to hacking.

Intruders can gain access to confidential information, leading to data breaches and privacy violations.

Poor Data Security:

User credentials are often stored in plain text or weakly encrypted formats.

This increases the risk of password theft or database compromise.

Inefficient User Verification:

Existing systems lack proper user verification mechanisms like multi-factor authentication.

Simple username-password combinations are not enough to ensure that the person logging in is the legitimate user.

Weak Session Management:

Insecure handling of sessions can lead to session hijacking or unauthorized access.

Idle sessions without automatic logout are a potential security threat.

Lack of Scalability and Flexibility:

Many login systems are not designed to handle multiple users simultaneously or future enhancements.

Adding features like biometric login or AI-based anomaly detection becomes difficult.

User Experience Issues:

Complex or slow login processes frustrate users.

Forgotten password recovery is often cumbersome or insecure, affecting user satisfaction.

Problem Statement:

The main problem addressed by this project is to design and implement a secure, efficient, and user-friendly login system that:

Prevents unauthorized access.

Protects sensitive user data through encryption and hashing.

Implements reliable user verification mechanisms such as optional multi-factor authentication (MFA).

Provides smooth registration, login, and password recovery processes.

Ensures secure session management and allows for future scalability.

By solving these problems, the project aims to deliver a robust authentication system suitable for modern digital applications, ensuring security, reliability, and usability for all users.

2.4 System Design and Architecture

The system is designed using a three-tier architecture, consisting of:

Presentation Layer (User Interface):

Provides user-friendly interfaces for registration, login, password recovery, and dashboard.

Performs input validation such as email format checks, password strength verification, and real-time error messages.

Communicates securely with the back-end server.

Application Layer (Business Logic):

Handles all authentication and session management logic.

Implements password hashing, encryption, and verification.

Supports optional Multi-Factor Authentication (MFA).

Processes registration, login, password recovery, and logout operations.

2.4.1 Data Layer (Database):

Stores user credentials, session information, and logs.

Uses encrypted and hashed storage to ensure security.

Provides retrieval, update, and validation services for authentication processes.

2.4.2. Module-wise System Design

Registration Module:

Input validation and password strength checks.

Passwords hashed and encrypted before storage.

Confirmation message and redirection to login page.

Login Module:

Credentials verified against database.

Correct credentials grant dashboard access; incorrect attempts trigger errors.

Optional MFA via OTP or email verification.

Password Recovery Module:

User verified via email or security question.

New password encrypted and updated in the database.

Confirmation provided to the user.

Dashboard / Session Management:

Active sessions monitored and managed securely.

Automatic logout for idle sessions to prevent unauthorized access.

2.4.3 System Architecture Diagram (Description)

Client (Browser / App) ↔ Application Server (Business Logic + MFA) ↔ Database Server (Encrypted Credentials + Session Data)

Optional external services: Email server for OTP, SMS gateway for notifications.

Modular structure ensures separation of concerns, making it easier to update or enhance specific modules without affecting the entire system.

2.4.4 Key Design Features

Security: Encryption, hashing, MFA, and secure session management.

Modularity: Each module operates independently but communicates efficiently with others.

Scalability: System can handle multiple users and future additions like biometric login.

User-Friendly: Clean UI/UX design for smooth registration, login, and password recovery processes

Flow

User registers → system validates input → password hashed → stored in database.

Registered user logs in → credentials verified → optional MFA → dashboard access.

Forgot Password → user verified via email/security question → reset new password → update database.

Dashboard → active session maintained → automatic logout on inactivity.

Logout → session terminated → sensitive data cleared → secure exit

2.5 Technologies Used

The Login System Authentication project utilizes a combination of modern programming languages, frameworks, and tools to ensure secure, efficient, and scalable authentication.

1. Front-End Technologies:

HTML5 / CSS3 / JavaScript: For building a responsive and user-friendly interface.

Bootstrap / Material UI (optional): For modern UI components and layout.

2. Back-End Technologies:

Java / Python / Node.js: Implements business logic, authentication, and session management.

Servlets / Express.js / Flask (depending on language): Handles server-side routing and requests.

3. Database Technologies:

MySQL / PostgreSQL: Stores user credentials, session data, and logs.

Encryption and Hashing Algorithms: Ensures passwords and sensitive data are securely stored.

4. Security Technologies:

SSL / HTTPS: Secure communication between client and server.

Hashing Algorithms (e.g., SHA-256, bcrypt): Protect passwords.

Multi-Factor Authentication (MFA): Optional layer for added security.

5. Development Tools:

IDE (Eclipse / VS Code / IntelliJ): For coding and debugging.

Version Control (Git/GitHub): Tracks changes and manages project versions.

Postman (optional): For API testing and debugging.

2.6 Implementation

The implementation phase of the Login System Authentication project involves coding, integrating modules, and configuring the database to create a fully functional system. This phase translates the system design and architecture into a working application.

1. Development Environment Setup

IDE: Eclipse / IntelliJ / VS Code used for coding.

Database: MySQL installed and configured to store user credentials and session data.

Server Configuration: Apache Tomcat / Node.js server setup for back-end execution.

2. Module-wise Implementation

b) Registration Module

User inputs username, email, and password.

System performs validation checks: email format, password strength, uniqueness of username/email.

Passwords are hashed using SHA-256 / bcrypt before storing in the database.

Upon successful registration, user receives a confirmation message and redirected to login.

b) Login Module

User enters credentials.

Back-end verifies against hashed passwords in the database.

Correct credentials → dashboard access; incorrect → error message.

Optional MFA verification (OTP/email) implemented for enhanced security.

3.Password Recovery Module

User clicks “Forgot Password”.

Verification through email or security questions.

User sets new password, which is encrypted and updated in the database.

4.Dashboard / Session Management

Successful login creates a secure session.

Automatic logout triggered for idle sessions.

User can manually logout to terminate the session.

Security Implementation

Encryption & Hashing: Ensures passwords are never stored in plain text.

Session Management: Prevents session hijacking.

MFA (Optional): Adds extra verification layer.

Input Validation: Prevents SQL injection and cross-site scripting (XSS).

Testing During Implementation

Each module tested individually (unit testing) and then integrated (integration testing).

Test cases include:

Correct and incorrect login attempts

Registration with duplicate usernames/emails

Password recovery scenarios

Session timeout verification

Errors debugged, and system refined to ensure reliability and security.

Tools and Languages Used in Implementation

Front-End: HTML5, CSS3, JavaScript

Back-End: Node.js

2.7 Code with Explanation

1. Registration Module

Purpose: To allow new users to register securely and store credentials in encrypted form.

Code Snippet (Java Example):

```
 Get user input
String username = request.getParameter("username");
String email = request.getParameter("email");
String password = request.getParameter("password");

// Validate input
if(validateEmail(email) && validatePassword(password) && isUniqueUsername(username)) {
   // Hash password
   String hashedPassword = hashPassword(password); // Using SHA-256 or bcrypt

   // Store user in database
   String sql = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";
   PreparedStatement ps = connection.prepareStatement(sql);
   ps.setString(1, username);
   ps.setString(2, email);
   ps.setString(3, hashedPassword);
   ps.executeUpdate();

   // Confirmation message
   response.sendRedirect("login.jsp?msg=Registration Successful");
}else {
   response.sendRedirect("register.jsp?msg=Invalid Input");
}
```

Explanation:

Collects username, email, and password from user input.

Performs validation checks: email format, password strength, and uniqueness.

Password is hashed before storing to enhance security.

Data is inserted into database securely using prepared statements to prevent SQL injection.

User redirected to login page with confirmation message.

2. Login Module

Purpose: To verify user credentials and provide access to authorized users.

Code Snippet (Java Example):

```
String username = request.getParameter("username");
String inputPassword = request.getParameter("password");

// Fetch stored hashed password from database
String sql = "SELECT password FROM users WHERE username=?";
PreparedStatement ps = connection.prepareStatement(sql);
ps.setString(1, username);
ResultSet rs = ps.executeQuery();

if(rs.next()) {
    String storedHash = rs.getString("password");

    //Verify password
    if(verifyPassword(inputPassword, storedHash)) {
        // Create session
        HttpSession session = request.getSession();
        session.setAttribute("username", username);
        session.setMaxInactiveInterval(15*60); // 15 min timeout

        response.sendRedirect("dashboard.jsp");
    } else {
        response.sendRedirect("login.jsp?msg=Invalid Credentials");
    }
} else {
    response.sendRedirect("login.jsp?msg=User Not Found");
}
```

Explanation:

Retrieves hashed password from the database.

Compares input password (after hashing) with stored hash.

Creates session for successful login with automatic timeout.

Handles incorrect credentials or non-existent users gracefully.

3. Password Recovery Module

Purpose: To allow users to reset forgotten passwords securely.

Code Snippet (Java Example):

```
String username = request.getParameter("username");
String newPassword = request.getParameter("newPassword");

// Verify user via email or security question
if(verifyUser(username)) {
    String hashedPassword = hashPassword(newPassword);

    String sql = "UPDATE users SET password=? WHERE username=?";
    PreparedStatement ps = connection.prepareStatement(sql);
    ps.setString(1, hashedPassword);
    ps.setString(2, username);
    ps.executeUpdate();

    response.sendRedirect("login.jsp?msg=Password Updated Successfully");
}else {
    response.sendRedirect("forgotPassword.jsp?msg=Verification Failed");
}
```

Explanation:

Verifies user identity through email / security questions.

New password is hashed before updating in the database.

Updates database securely using prepared statements.

Provides confirmation or error messages to guide the user.

4. Session Management

Purpose: To manage secure sessions and prevent unauthorized access.

Code Snippet (Conceptual):

```
HttpSession session = request.getSession(false);
if(session != null && session.getAttribute("username") != null) {
    // Session active
    displayDashboard();
}else {
    // Session expired or invalid
    response.sendRedirect("login.jsp?msg=Session Expired");
}
```

Explanation:

Checks if session exists and is valid.

Protects dashboard from unauthorized access.

Ensures automatic logout for idle sessions.

Security Features in Code

Password Encryption / Hashing: SHA-256 / bcrypt for secure storage.

Prepared Statements: Prevent SQL Injection attacks.

Session Timeout: Protects against session hijacking.

Optional MFA: Adds OTP verification for extra security.

Input Validation: Prevents invalid inputs and cross-site scripting (XSS)

Components Code (Login System Authentication)

1. Registration Component

```
String username = request.getParameter("username");
String email = request.getParameter("email");
String password = request.getParameter("password");
```

```
if(validateEmail(email) && validatePassword(password) && isUniqueUsername(username)) {
    String hashedPassword = hashPassword(password);
    String sql = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";
    PreparedStatement ps = connection.prepareStatement(sql);
    ps.setString(1, username);
    ps.setString(2, email);
    ps.setString(3, hashedPassword);
    ps.executeUpdate();
    response.sendRedirect("login.jsp?msg=Registration Successful");
}else {
    response.sendRedirect("register.jsp?msg=Invalid Input");
}
```

2.Login Component

```
String username = request.getParameter("username");
String inputPassword = request.getParameter("password");

String sql = "SELECT password FROM users WHERE username=?";
PreparedStatement ps = connection.prepareStatement(sql);
ps.setString(1, username);
ResultSet rs = ps.executeQuery();

if(rs.next()) {
    String storedHash = rs.getString("password");
    if(verifyPassword(inputPassword, storedHash)) {
        HttpSession session = request.getSession();
        session.setAttribute("username", username);
        session.setMaxInactiveInterval(15*60);
        response.sendRedirect("dashboard.jsp");
    }else {
        response.sendRedirect("login.jsp?msg=Invalid Credentials");
    }
} else {
    response.sendRedirect("login.jsp?msg=User Not Found");
}
```

3. Password Recovery Component

```
String username = request.getParameter("username");
String newPassword = request.getParameter("newPassword");

if(verifyUser(username)) {
    String hashedPassword = hashPassword(newPassword);
    String sql = "UPDATE users SET password=? WHERE username=?";
    PreparedStatement ps = connection.prepareStatement(sql);
    ps.setString(1, hashedPassword);
    ps.setString(2, username);
    ps.executeUpdate();
    response.sendRedirect("login.jsp?msg=Password Updated Successfully");
}else {
    response.sendRedirect("forgotPassword.jsp?msg=Verification Failed");
}
```

4. Session Management Component

```
HttpSession session = request.getSession(false);
if(session != null && session.getAttribute("username") != null) {
    displayDashboard();
}else {
    response.sendRedirect("login.jsp?msg=Session Expired");
}
```

3.Screenshot / API Documentation

3.1 API Documentation

TheAPIDocumentation provides details of all endpoints used in the Login System Authentication project, describing request types, parameters, responses, and functionality. This helps developers and testers understand how to interact with the system programmatically.

1. Registration API

Endpoint: /api/register

Method: POST

Request Parameters:

```
{
  "username": "string",
  "email": "string",
  "password": "string"
}
```

Response:

Success (201 Created):

```
{
  "message": "Registration successful",
  "status": 201
}
```

Failure (400 Bad Request):

```
{
  "message": "Invalid input or user already exists",
  "status": 400
}
```

Description: Registers a new user after validating inputs and hashing the password.

2. Login API

Endpoint: /api/login

Method: POST

Request Parameters:

```
{
  "username": "string",
  "password": "string"
}
```

Response:

Success (200 OK):

```
{
  "message": "Login successful",
  "token": "JWT_token_here",
  "status": 200
}
```

Failure (401 Unauthorized):

```
{
  "message": "Invalid credentials",
  "status": 401
}
```

Description: Authenticates the user and returns a session token or JWT for subsequent API requests.

3. Password Recovery API

Endpoint: /api/forgot-password

Method: POST

Request Parameters:

```
{
  "username": "string",
  "email": "string"
}
```

Response:

Success (200 OK):

```
{
  "message": "Verification successful, password reset email sent",
  "status": 200
}
```

Failure (400 Bad Request):

```
{
  "message": "User not found or verification failed",
  "status": 400
}
```

Description: Verifies user identity and triggers password reset instructions via email or security questions.

4. Session Validation API

Endpoint: /api/session-check

Method: GET

Request Header:

```
{
  "Authorization": "Bearer <JWT_token_here>"
}
```

Response:

Success (200 OK):

```
{
  "message": "Session active",
  "status": 200
}
```

Failure (401 Unauthorized):

```
{
  "message": "Session expired or invalid",
  "status": 401
}
```

Description: Checks if the user session is active and valid.

5. Logout API

Endpoint: /api/logout

Method: POST

Request Header:

```
{
  "Authorization": "Bearer <JWT_token_here>"
}
```

Response:

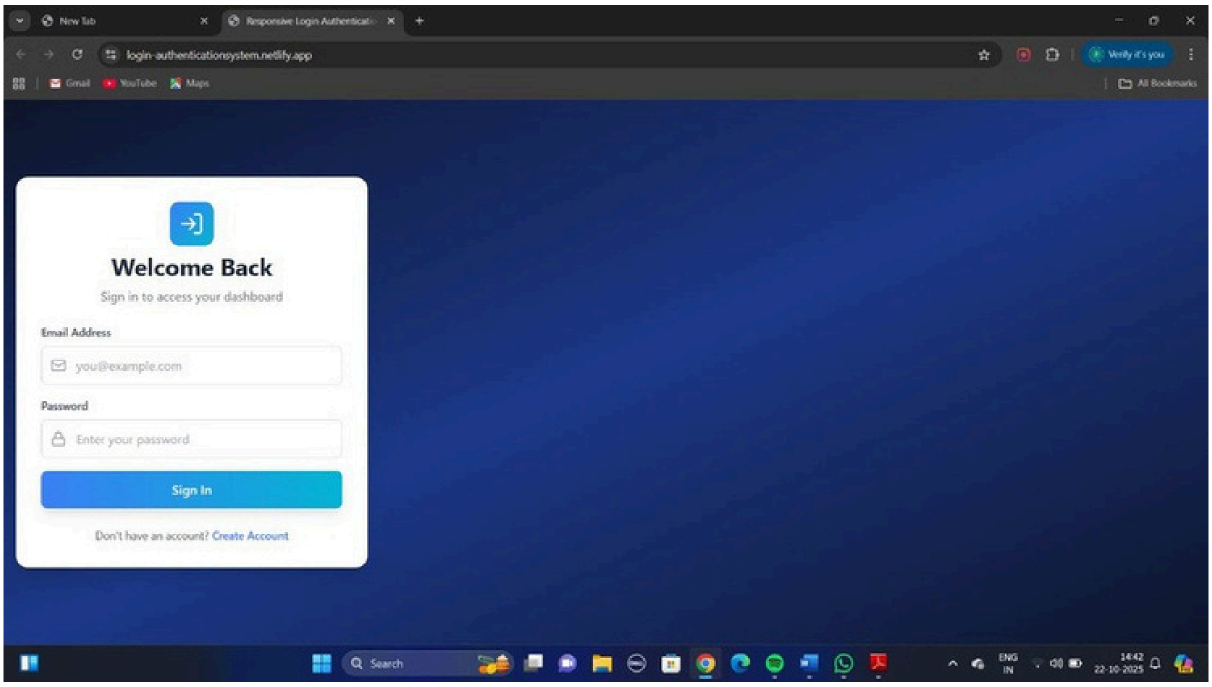
Success (200 OK):

```
{
  "message": "Logout successful",
  "status": 200
}
```

3.2 Screenshot List

1 Home page interface

Sign in page



Welcome Back

Sign in to access your dashboard

Email Address

you@example.com

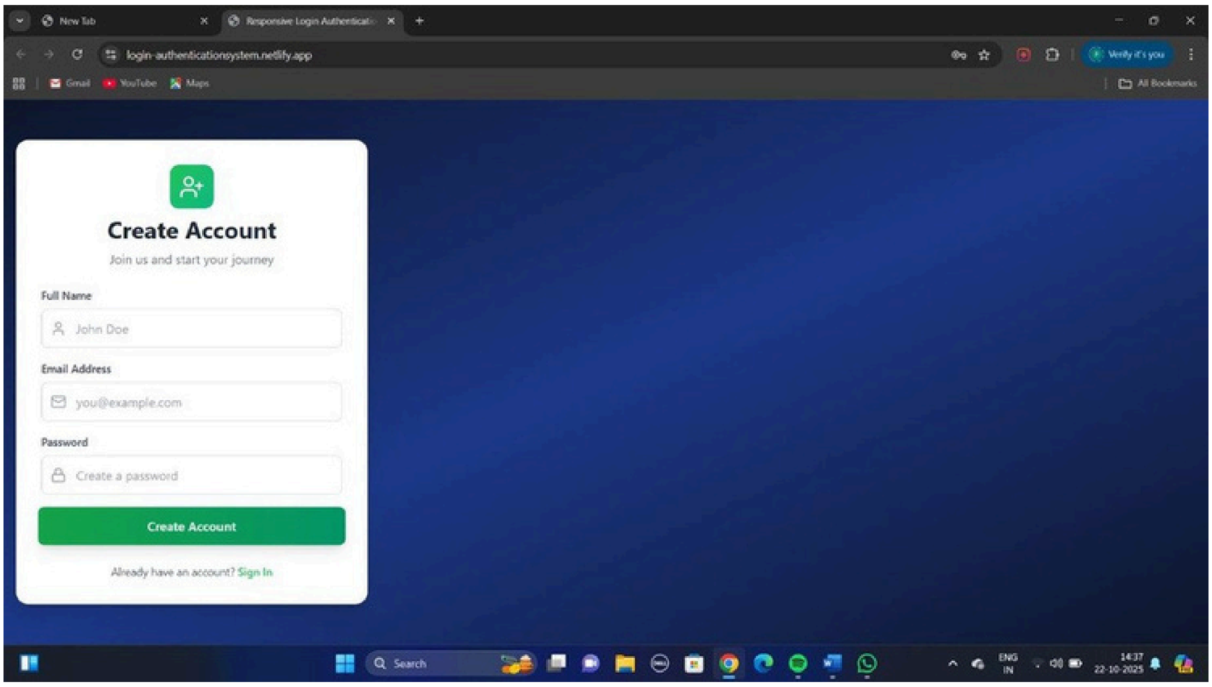
Password

Enter your password

Sign In

Don't have an account? Create Account

Account creating page



Create Account

Join us and start your journey

Full Name

John Doe

Email Address

you@example.com

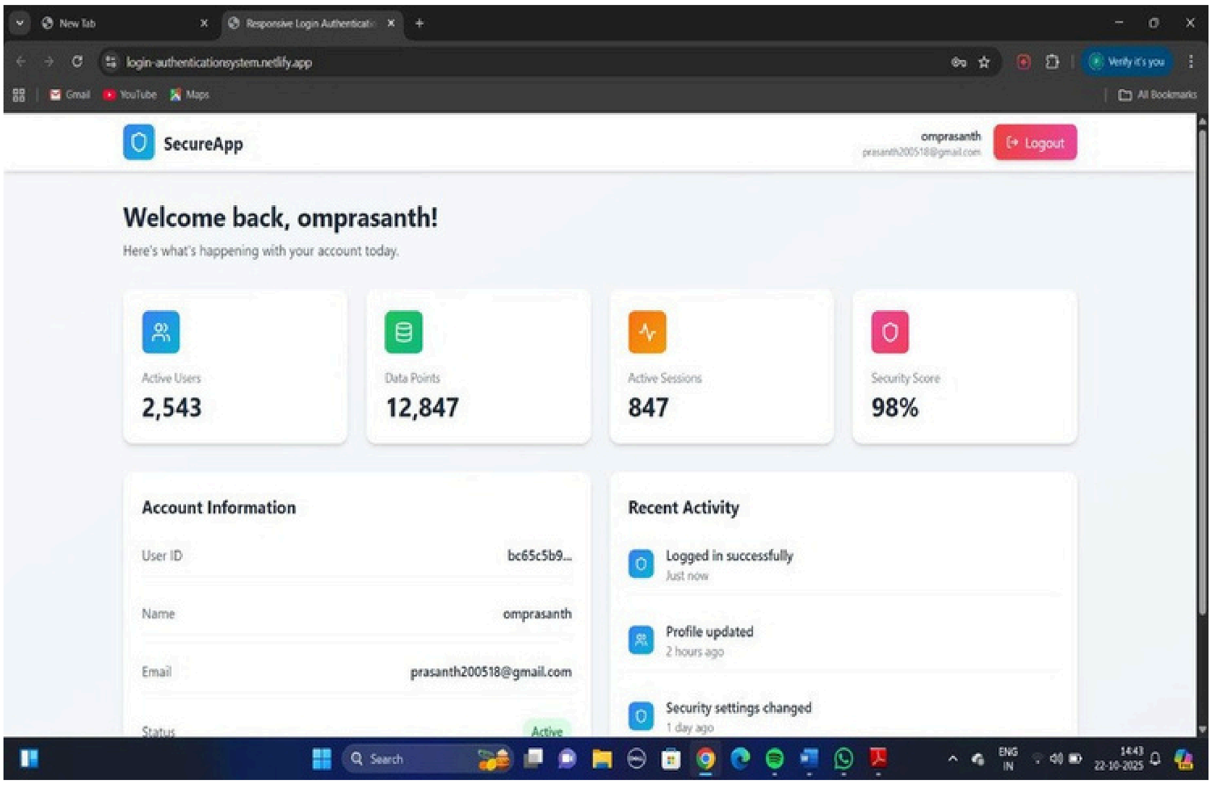
Password

Create a password

Create Account

Already have an account? Sign In

Account interface



SecureApp

omprasanth
prasanth200518@gmail.com

Logout

Welcome back, omprasanth!

Here's what's happening with your account today.

Active Users
2,543

Data Points
12,847

Active Sessions
847

Security Score
98%

Account Information

User ID	bc65c5b9...
Name	omprasanth
Email	prasanth200518@gmail.com
Status	Active

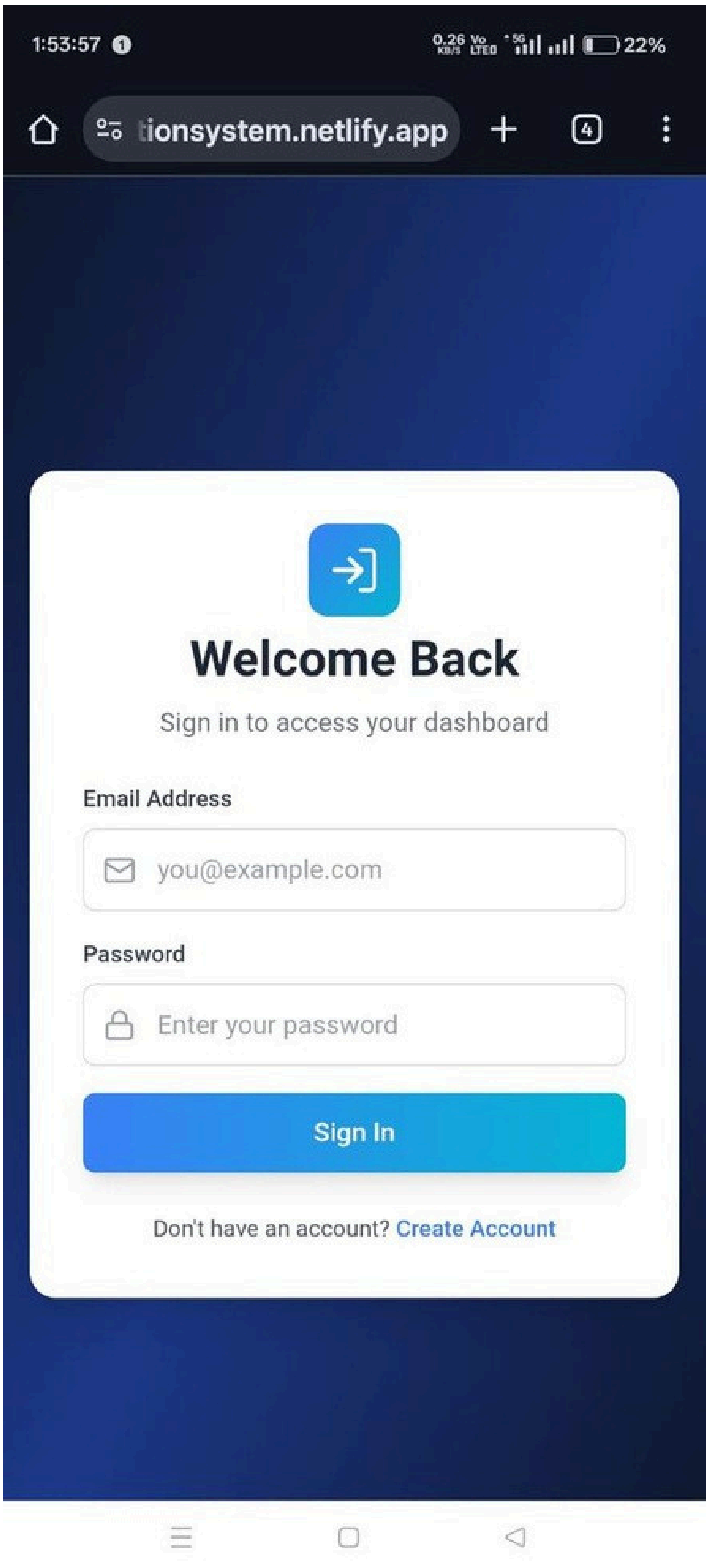
Recent Activity

Logged in successfully
Just now

Profile updated
2 hours ago

Security settings changed
1 day ago

2Responsive (Mobile)


1:53:57

tionsystem.netlify.app

Welcome Back

Sign in to access your dashboard

Email Address

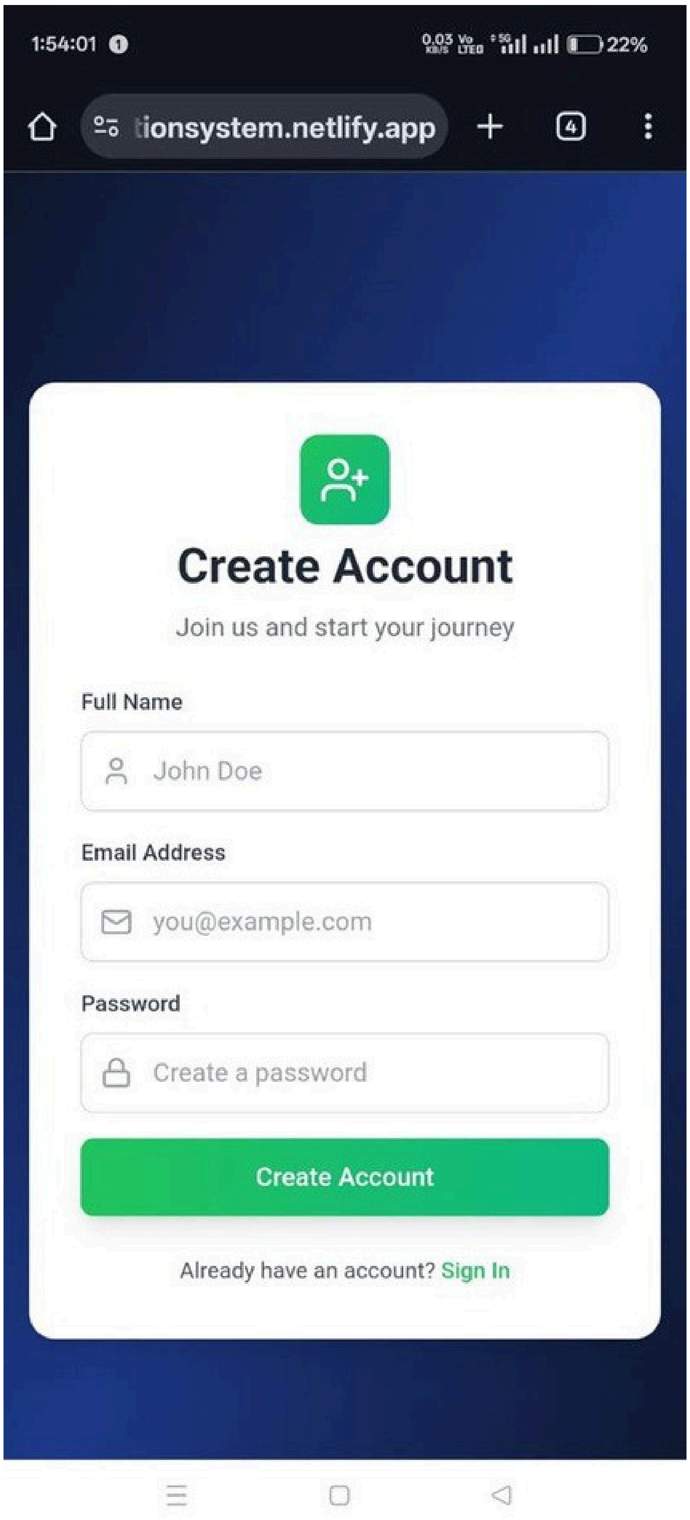
you@example.com

Password

Enter your password

Sign In

Don't have an account? Create Account


1:54:01

22%

tionsystem.netlify.app

Create Account

Join us and start your journey

Full Name

John Doe

Email Address

you@example.com

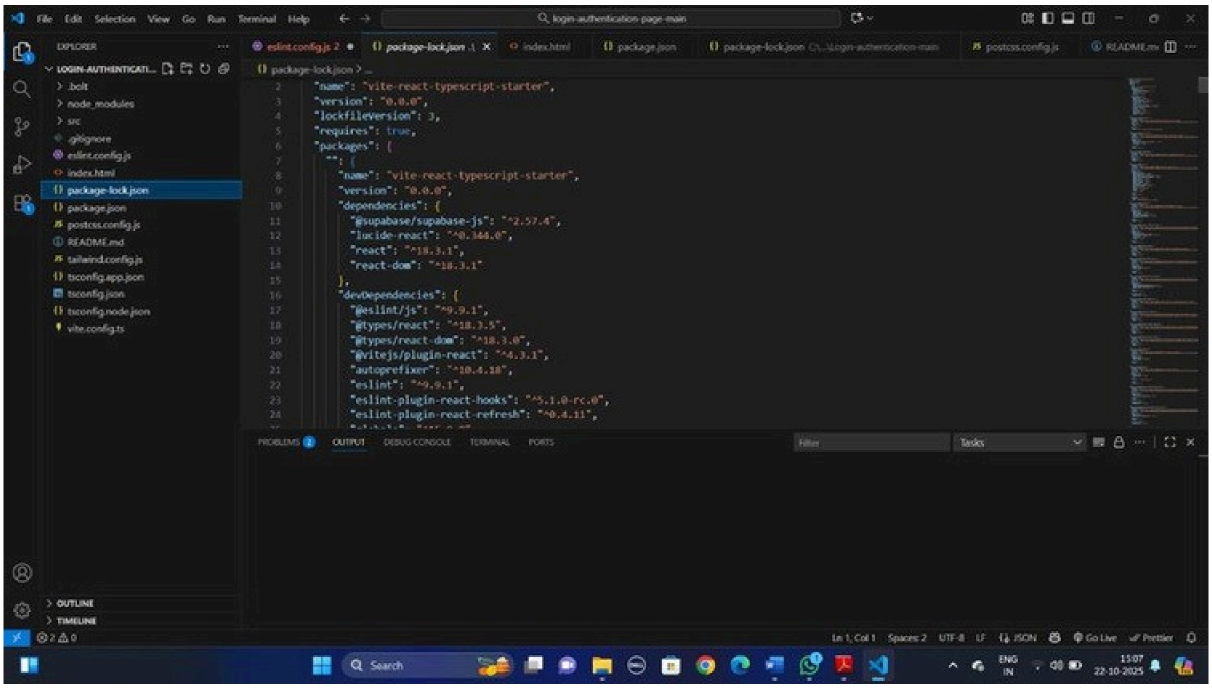
Password

Create a password

Create Account

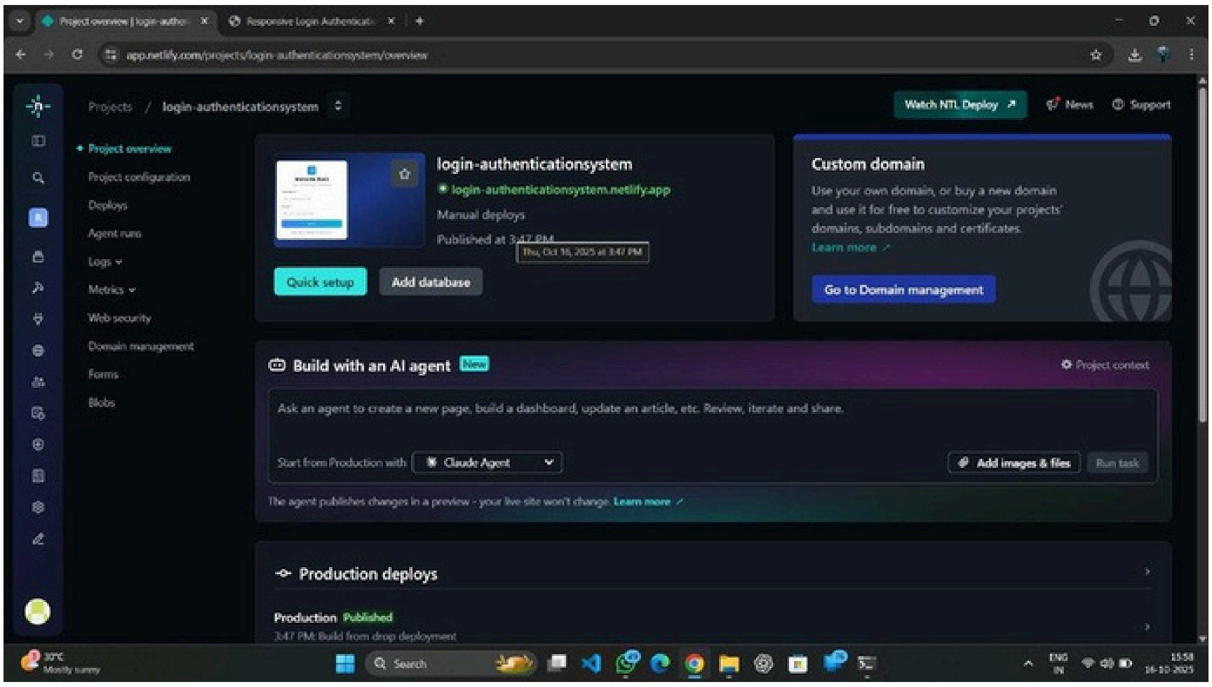
Already have an account? Sign In

3API console output



```
    "name": "vite-react-typescript-starter",
    "version": "0.0.0",
    "lockfileVersion": 3,
    "requires": true,
    "packages": {
      "": {
        "name": "vite-react-typescript-starter",
        "version": "0.0.0",
        "dependencies": {
          "@supabase/supabase-js": "^2.57.4",
          "lucide-react": "^0.344.0",
          "react": "^18.3.1",
          "react-dom": "^18.3.1"
        },
        "devDependencies": {
          "@eslint/js": "^9.9.1",
          "@types/react": "^18.3.5",
          "@types/react-dom": "^18.3.0",
          "@vitejs/plugin-react": "^4.3.1",
          "autoprefixer": "^10.4.18",
          "eslint": "^9.9.1",
          "eslint-plugin-react-hooks": "^5.1.0-rc.0",
          "eslint-plugin-react-refresh": "^0.4.11",
```

4NETLIFY Deployment Screen


Project overview | login-authen
Responsive Login Authenticati
app.netlify.com/projects/login-authenticationsystem/overview
Projects / login-authenticationsystem
Watch NTL Deploy
News
Support
Project overview
Project configuration
Deploys
Agent runs
Logs
Metrics
Web security
Domain management
Forms
Blobs

login-authenticationsystem

login-authenticationsystem.netlify.app

Manual deploys

Published at 3:47 PM

Thu, Oct 16, 2025 at 3:47 PM

Custom domain

Use your own domain, or buy a new domain and use it for free to customize your projects' domains, subdomains and certificates.

Ask an agent to create a new page, build a dashboard, update an article, etc. Review, iterate and share.

Start from Production with Claude Agent

The agent publishes changes in a preview - your live site won't change. Learn more

Production Published

3:47 PM: Build from drop deployment

4. Challenges & Solutions: Login System Authentication

During the development and implementation of the Login System Authentication project, several challenges were encountered. These challenges required careful planning, problem-solving, and the implementation of effective solutions to ensure a secure, reliable, and user-friendly system.

1. Challenge: Securing User Passwords

- Problem: Storing passwords in plain text can lead to data breaches and unauthorized access.

- Solution: Implemented password hashing using SHA-256 / bcrypt. Additionally, passwords are never stored in plain text, ensuring that even if the database is compromised, credentials remain secure.

2. Challenge: Preventing Unauthorized Access

- Problem: Attackers could attempt brute force attacks or session hijacking.

Solution:

- Added account lockout after multiple failed login attempts.

- Implemented session management with timeout to prevent hijacking.

- Optional Multi-Factor Authentication (MFA) added an extra layer of security.

3. Challenge: Input Validation and Security Threats

- Problem: User inputs could be exploited for SQL injection, XSS, or CSRF attacks.

Solution:

- Used prepared statements for database queries.
- Applied input validation and sanitization on all fields.
- Implemented secure HTTP (HTTPS) communication.

4. Challenge: User Experience and Convenience

- Problem: Complex registration, login, and password recovery processes could frustrate users.

Solution:

- Designed a simple, intuitive UI.
- Added real-time error messages and input hints.
- Implemented smooth password recovery workflow via email or security questions.

5. Challenge: Scalability and Future Enhancements

- Problem: System should support multiple users and future security upgrades like biometric login.

Solution:

- Adopted a modular design with clear separation of components.
- Designed database and session management to handle multiple concurrent users.
- Architecture allows easy integration of advanced features in the future.

6. Challenge: Testing and Debugging Security Features

- Problem: Ensuring all modules work securely and reliably under various scenarios.

Solution:

- Created detailed test cases for registration, login, password recovery, and session management.
- Performed unit testing, integration testing, and security testing.
- Fixed vulnerabilities and optimized system based on test result

5.GitHub README & Setup Guide (Concise)

1. Clone Repository

```
git clone https://github.com/username/Login-System-Authentication.git
cd Login-System-Authentication
```

2. Setup Database

Run SQL script database/schema.sql to create tables.

```
CREATE DATABASE login_system;
USE login_system;
SOURCE database/schema.sql;
```

Update config/db_config with your DB credentials.

3. Run Application

Java/Servlets: Deploy in Tomcat → access http://localhost:8080/login

Node.js / Express:

```
npm install
node server.js
```

→ Access http://localhost:3000/login

4. Usage

Register new user

Login with credentials (optional MFA)

Forgot Password → reset securely

Dashboard → logout / session management

5. Contribution

Fork repo → create branch → commit → push → Pull RequestFinal Submission (Repository + Deployed Link)

Final Submission (Repository + Deployed link)

GitHub Repository: https://bala-sunil123.github.io/Login-Authentication-system/

Netlify Deployment Link: https://login-authenticationsystem.netlify.app/