Secure Authentication System Development

Iteration Summary and Future Plans

Faizan ul Haq(21i1771), Mubashir Zaidi(21i1764), Maria Khan(21i1352) DS-M

October 15, 2024

Contents

1 Introduction

This report outlines the progress made in Iteration 1 of the Secure Authentication System Development project and provides an overview of the upcoming work planned for Iterations 2, 3, and 4.

2 Iteration 1: Environment Setup and Basic User Authentication

2.1 Summary of Work Completed

In Iteration 1, we established the foundational components of the secure authentication system:

- **Database Configuration**:
  - Initialized a SQLite database to store user data.
  - Defined the `users` table schema with fields for username, email, and securely hashed passwords.
- **Basic User Authentication**:
  - Implemented user registration functionality that collects username, email, and password.
  - Ensured passwords are securely stored using bcrypt hashing.
  - Developed user login functionality that authenticates users based on email and password.
  - Included basic session management to keep users logged in after authentication.
  - Added logout functionality to allow users to end their sessions.
- **Frontend Implementation**:
  - Incorporated custom HTML and CSS for the login and registration pages, enhancing the user interface.
  - Ensured forms capture user input securely and interact correctly with backend routes.
- **Application Routing**:
  - Configured the Flask application to land on the login page by default.
  - Set up navigation between login, registration, and dashboard pages.

2.2 Key Achievements

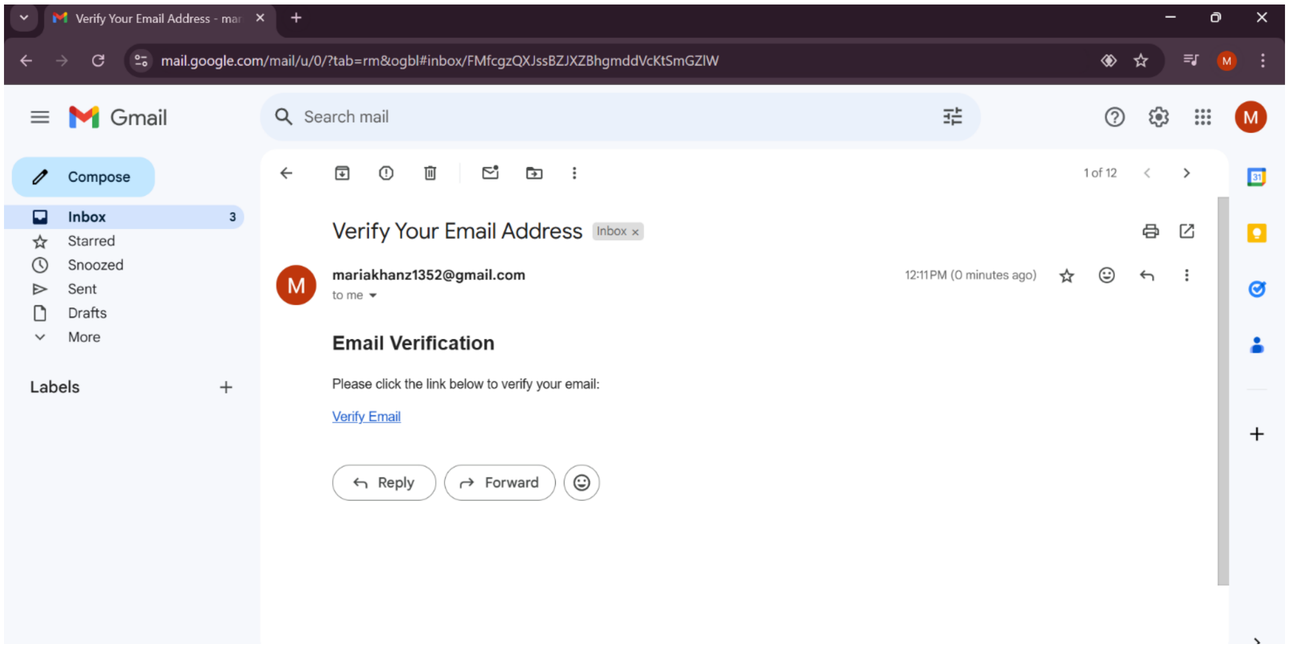
- Successfully set up a secure development environment.
- Implemented secure password storage using bcrypt.
- Created a user-friendly interface with custom styling.
- Established session management for user authentication.

3 Iteration 2: Email Verification and Enhanced Session Management

3.1 Planned Work

In Iteration 2, we further broadened the code by enhancing the authentication system by adding email verification and improving session management:

- **Email Verification**:
  - The code includes an email sending function send_verification_email, which uses the smtplib library to handle SMTP communication. In the HTML page it contains a unique verification link and sends it via Gmail SMTP server using the specified sender's email and password.
  - In the /register route, after a user registers by providing a username, email, and password, the system validates the email format and encrypts the password. A unique verification token is generated and saved in the database with a one-day expiration time. If the registration succeeds, the send_verification_email function is called to email the user the verification link, allowing them to confirm their email.
  - The /verify_email/<token> route is defined to handle email verification. When a user clicks the verification link, the code retrieves the user's data based on the token and checks that the token is valid (not expired). If the token is valid, it updates the user record in the database, marking the email as verified and removing the token and expiration time.
  - The users table includes an is_verified field, which defaults to FALSE. In the /login route, before allowing access, the code checks if the is_verified flag is set to TRUE. If the email isn't verified, a message informs the user that they need to verify their email before they can log in.


Verify Your Email Address

mariakhanz1352@gmail.com
to me

12:11 PM (0 minutes ago)

Email Verification

Please click the link below to verify your email:

Verify Email

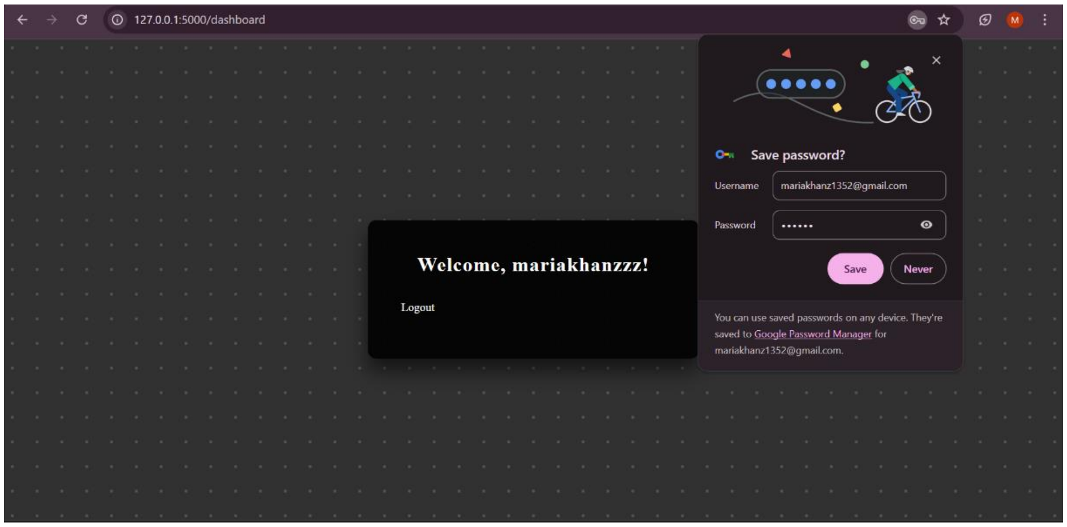
Verify Your Email Address - mar
mail.google.com/mail/u/0/?tab=rm&ogbl#inbox/FMfcgzQXJssBZJXZBhgmddVcKtSmGZlW
Gmail
Search mail
Compose
Inbox 3
Starred
Snoozed
Sent
Drafts
More
Labels
1 of 12
Inbox
Reply
Forward

- **Enhanced Session Management**:
  - The app.secret_key setting is configured to secure session data. Session cookies store user ID and username securely, limiting access to these sensitive details.
  - Flask's built-in session management leverages the secret key, ensuring data integrity and encryption for

session cookies. Additional security measures like HTTPS and secure cookie flags could be recommended for production deployment.

- Flash messages communicate account status clearly to users. During registration, the user is informed to verify their email, and in case of login attempts with an unverified account, they receive a prompt to complete verification first.

- The code includes logic to check the user's verification status during login. Unverified users attempting to log in are redirected to the login page with an error message, while verified users are directed to the dashboard. Even if we close the tab without logging out, the user will be able to resume on his account directly when website will open without the need to login again.



127.0.0.1:5000/dashboard

Welcome, mariakhanzzz!

Logout

Save password?

Username mariakhanz1352@gmail.com

Password ••••••

You can use saved passwords on any device. They're saved to Google Password Manager for mariakhanz1352@gmail.com.

- **Frontend Updates**:
  - We updated the templates (web pages) to show a message after registration, telling users to check their email for a verification link.
  - We created messages to inform users if their account activation was successful or if there was an issue. When users click the link in their email, they'll see a clear message saying whether their account is now verified or if they need a new link.
  - If someone tries to log in without verifying their email, they'll see a message reminding them to verify first. This way, users know exactly why they can't log in and what they need to do next.

https://github.com/FaizanUlHaq262/Info-Security-Project.git

4 Iteration 3: OTP Integration for Two-Factor Authentication

4.1 Planned Work

In Iteration 3, we will integrate One-Time Password (OTP) functionality to add an extra layer of security:

- **OTP Generation and Verification**:
  - Implement a secure method to generate OTPs.

  - Send OTPs to users’ registered email addresses after successful password authentication.
  - Allow users to request a new OTP if the previous one expires.

- **Login Workflow Enhancement**:
  - Modify the login process to include OTP verification as a second step.
  - Update the frontend to include an OTP input field after password authentication.
  - Ensure that the OTP verification process is user-friendly and secure.

- **Security Enhancements**:
  - Securely store and handle OTPs to prevent reuse or interception.
  - Implement safeguards against OTP brute-force attacks.

5 Iteration 4: Security Features, Testing, and Documentation

5.1 Planned Work

In Iteration 4, the focus will be on bolstering security, thorough testing, and preparing final documentation:

- **Advanced Security Measures**:
  - Implement rate limiting on login attempts and OTP requests using tools like `Flask-Limiter`.
  - Use parameterized queries to prevent SQL injection attacks.
  - Ensure sensitive data is not exposed in logs or error messages.
- **Comprehensive Testing**:
  - Perform functional testing of all user flows to ensure they work as intended.
  - Conduct security testing, including penetration testing to identify and fix vulnerabilities.
  - Verify that password hashing and OTP handling are secure.
- **Documentation and Final Deliverables**:
  - Finalize the codebase with comments and documentation.