|  |
| --- |
| Two-Factor Authentication (2FA) Using TOTP  Cryptography and Security |
| Lampros Chalatsis, Konstantinos Pogelis |
| January 2026  Source code repository: [Two\_Factor\_Authentication\_Project](https://github.com/lambros1234/Two_Factor_Authentication_Project) |

# Abstract

This project presents the design and implementation of a secure authentication system using Two-Factor Authentication (2FA) based on Time-Based One-Time Passwords (TOTP). The system combines password-based authentication with cryptographic mechanisms, including bcrypt hashing and HMAC-based OTP generation. The implementation demonstrates how 2FA can counter common security threats such as password compromise, brute-force attacks, and credential reuse. The project includes a working web application, session-based access control, and a demonstration of security attack scenarios.

# Table of Contents

Introduction ………………………………………………………………………… 4

Motivation and Problem Statement ………………………………………………… 5

Objectives …………………………………………………………………………... 6

System Architecture and Authentication Flow ……………………………………... 7

Project Structure……………………………………………………………… 7

Authentication Flow………………………………………………………….. 8

Flowchart…………………………………………………………………….. 9

Threat Model and Security Assumptions……………………………………………. 10

Implementation Details……………………………………………………………… 11

Password Hashing with bcrypt………………………………………………. 11

Session Management………………………………………………………… 12

TOTP Enrollment and Verification………………………………………….. 13

TOTP Setup………………………………………………………….. 14

TOTP Login Verification…………………………………………….. 14

Middleware-Based Access Control…………………………………………………. 15

Database Design Considerations……………………………………………………. 16

Security Analysis and Attack Demonstration……………………………………….. 17

Brute-Force Attack Resistance………………………………………………. 17

Replay Attack Considerations……………………………………………….. 18

Limitations…………………………………………………………………… 18

System Demonstration……………………………………………………………… 19

Future Improvements……………………………………………………………….. 28

Conclusion………………………………………………………………………….. 29

References…………………………………………………………………………… 30

# Introduction

**Two-factor authentication (2FA)** is an identity and access management method that adds a crucial security layer by requiring two different proofs of identification to access resources and data. It helps prevent cybercriminals from stealing users’ identities or accessing accounts, even when passwords have been compromised.

Passwords alone are often insufficient to secure user accounts due to risks such as phishing, credential reuse, and brute-force attacks. This project implements a 2FA system that combines passwords with time-based one-time passwords (TOTP), demonstrating how cryptographic primitives like bcrypt hashing and HMAC-based OTP generation are applied in real-world authentication systems.

# 

# Motivation and Problem Statement

We chose 2FA because it counters the major risks associated with password-only authentication, demonstrates practical use of cryptography, and is widely adopted as a standard in secure authentication practices.

Traditional password-based authentication is vulnerable to security risks such as password reuse across multiple websites, phishing attacks, and brute-force attempts. These vulnerabilities can lead to unauthorized access, data breaches, and loss of sensitive information. There is a need for a more secure authentication process that increases user account protection without reducing usability. This project addresses this problem by implementing 2FA using TOTP, providing an additional layer of security beyond standard passwords.

# 

# Objectives

* **Implement secure user registration and login using bcrypt**
  + Passwords are hashed with bcrypt to ensure that even if the database is compromised, attackers cannot recover the original passwords.
* **Generate and store per-user TOTP secrets for 2FA**
  + Each user has a unique secret key used to generate time-based OTPs, ensuring that the second factor is personal and secure.
* **Enable QR-code enrollment for authenticator apps**
  + Users can scan a QR code with an app like Google Authenticator to easily enroll their 2FA secret.
* **Verify TOTP codes during login**
  + Ensures that only users who provide the correct OTP and their password can access protected resources.
* **Protect routes using middleware**
  + Middleware enforces authentication and 2FA verification across protected routes, centralizing access control and improving security.
* **Explain the cryptography behind password hashing and TOTP**
  + Demonstrate understanding of how adaptive hashing, HMAC, and time-based OTPs contribute to secure authentication systems.

# System Architecture and Authentication Flow

## Project Structure

The project follows a modular architecture that separates concerns and improves maintainability. Each component is responsible for a specific aspect of the authentication process.

* **server.js** - Initializes the Express server, configures middleware, and defines protected routes
* **db.js** - Handles SQLite database initialization and connection management
* **routes/auth.js** - Implements user registration, login, logout, and password verification
* **routes/twofa.js** - Handles TOTP enrollment, QR code generation, and 2FA verification
* **middleware/authCheck.js** - Enforces authentication and two-factor verification
* **public/** - Contains HTML interfaces for authentication and user interaction
* **security-tests/** - Contains controlled scripts used to demonstrate security properties and attack resistance

## Authentication Flow

1. **Registration**: User submits username and password. The password is hashed using bcrypt and stored in the database.
2. **Login**: User enters credentials. Passwords are verified using bcrypt.
3. **2FA Verification**: If enabled, the user is prompted to enter a TOTP from their authenticator app.
4. **Middleware Check**: Protected routes are accessible only if the user has successfully logged in and, if required, verified 2FA.
5. **Access Granted**: The user can access the dashboard or other protected resources

## Flowchart

*Figure 1: Authentication and Two-Factor Verification Flow*

## Threat Model and Security Assumptions

This project assumes a standard web threat model in which attackers may obtain user passwords through phishing, credential reuse, or database leaks. The system assumes a standard web attacker with network access but without physical access to the user’s authenticator device or server-side secrets. The system is designed to mitigate the impact of such compromises by requiring a second authentication factor based on Time-Based One-Time Passwords (TOTP).

The TOTP mechanism assumes that the attacker does not have physical or logical access to the user’s authenticator device. Even if the attacker successfully logs in using valid credentials, access is denied unless a valid time-based code is provided.

Replay attacks against the TOTP mechanism are considered within the context of an authenticated session. Since TOTP codes are time-limited and expire every 30 seconds, previously valid codes cannot be reused successfully, protecting against replay attacks during the second authentication phase.

# Implementation Details

## Password Hashing

User passwords are protected using the bcrypt hashing algorithm. During registration, each password is hashed with a unique salt and a configurable number of salt rounds before being stored in the database. This ensures that plaintext passwords are never stored or transmitted.

**Bcrypt** is a cryptographic hash function designed for password hashing and safe storing in the backend of applications. It runs a complex hashing process, during which a user's password is transformed into a fixed-length thread of characters. It uses a one-way hash function, which means that once the password is hashed, it can't be reversed to its original form. Every time a user logs into their account, bcrypt hashes their password again and compares the new hash value stored in the system's memory to check if the password matches.

Instead of simply hashing the given password, bcrypt adds a random piece of data, called a **salt**, to create a unique hash that is almost impossible to break with automated guesses during brute-force attacks.

Additionally, bcrypt uses a configurable cost factor, known as salt rounds, which increases the computational effort required to compute each hash. This significantly slows down brute-force and dictionary attacks, making large-scale password cracking impractical even if the database is compromised.

## Session Management

**Session management** is the process of securely maintaining a user’s authentication state across multiple requests in a web application, despite the stateless nature of the HTTP protocol.

In this project, session management is used to track whether a user has successfully authenticated with a password and, if enabled, completed two-factor authentication. After successful verification, a server-side session is created to store the user’s identity and authentication status, allowing access to protected routes without repeatedly transmitting credentials.

The session explicitly tracks whether the user has completed the second authentication factor, preventing access to protected routes until both authentication steps are completed.

## TOTP Enrollment and Verification

Time-Based One-Time Passwords (TOTP) are a secure, standards-based method used in multi-factor authentication (MFA). TOTP generates short-lived numeric codes on a user’s device based on a shared secret and the current time.

During the enrollment process, the system generates a unique secret key for the user. This secret is securely transferred to an authenticator application, typically via a QR code. Both the server and the authenticator application store this secret, allowing them to independently generate the same time-based codes.

During verification, the user provides the current TOTP code generated by their authenticator application. The server validates this code by generating its own expected value using the shared secret and the current time window. Access is granted only if the codes match, ensuring that possession of the user’s device is required in addition to their password.

## 

## 

## 

## 

## 

### **TOTP Enrollment (Setup)**

1. **Initiation** - User accesses the service, registers, and enables TOTP in their account during the first login
2. **Key Generation** - The service generates a unique, shared secret for the user and displays it as a QR code
3. **App Setup** - Users open their authenticator app (Google Authenticator is used in this project), and use the “+” button to scan the QR code
4. **Confirmation** - The app then generates a code every 30 seconds. The user enters the current code from the app in the service’s prompt to complete the setup and activate the account

### **TOTP Verification (Login)**

1. **Login Attempt** - User enters username and password to the service
2. **MFA Prompt** - the service prompts for a TOTP code
3. **Code Generation** - User checks their authenticator app for the current-time code
4. **Validation** - User enters the code into the service. The server then calculates the expected code using a shared secret and the current time. If it matches, login succeeds

## 

## 

## 

## Middleware-Based Access Control

**Middleware** is a core concept in web application frameworks such as Express.js. it refers to functions that execute between receiving an HTTP request and sending an HTTP response. Middleware is commonly used to handle cross-cutting concerns such as authentication, authorization, logging, and input validation.

In this project, middleware is used to enforce access control and ensure that only authenticated users can access protected routes. Since HTTP is a stateless protocol, the application relies on session management to track logged-in users across multiple requests. It checks the session state before allowing access to sensitive resources.

A custom authentication middleware (authCheck.js) was implemented to verify whether a user is logged in by checking if a valid session object exists. If it doesn't, the middleware blocks access and redirects the user to the login page. This helps prevent unauthorized users from directly accessing protected pages such as the dashboard.

By centralizing authentication logic in the middleware, the application avoids duplicating security checks across multiple routes. This improves maintainability and reduces the risk of implementation errors.

Database Design Considerations

The database design of the system was created with security, simplicity, and clarity in mind.

The database stores user credentials and authentication metadata in a dedicated **users** table. Passwords are never stored in plain text. Instead, they are hashed using the bcrypt algorithm before being saved. This ensures that even if the database were to be compromised, attackers wouldn't be able to recover the original passwords.

To support 2FA, the database schema includes extra fields to track the 2FA state. It includes a **twofa\_secret** field, which stores the Base32-encoded secret required for TOTP generation and verification. This secret is only stored after a user completes the 2FA setup process. The **twofa\_enabled** field is implemented as an integer flag (0 or 1) to indicate whether 2FA is active for a user.

### **Users Table Structure**

* id (INTEGER, Primary Key)
* username (TEXT, Unique)
* password (TEXT, bcrypt hash)
* twofa\_secret (TEXT, nullable)
* twofa\_enabled (INTEGER, default 0)

# Security Analysis and Attack Demonstration

This section analyzes the security properties of the implemented authentication system and demonstrates how the chosen cryptographic mechanisms mitigate common attack scenarios. Controlled attack simulations were performed in a test environment to evaluate the system’s resistance to brute-force and replay attacks.

All attack demonstrations were conducted on test accounts created specifically for evaluation purposes and did not involve real user data or production environments.

## Brute-Force Attack Resistance

Brute-force attacks attempt to gain unauthorized access by systematically guessing user passwords. In traditional authentication systems that store passwords insecurely or rely on weak hashing algorithms, such attacks can be highly effective once an attacker gains access to the authentication database.

In this project, password-based authentication is protected using the bcrypt hashing algorithm. Bcrypt is specifically designed to resist brute-force attacks by incorporating a configurable cost factor that increases the computational effort required to compute each password hash. As a result, each password verification operation is intentionally slow, significantly limiting the number of guesses an attacker can attempt per second.

A controlled brute-force demonstration script was implemented to illustrate this effect. The script simulates repeated login attempts against test accounts. Due to bcrypt’s computational cost, each attempt requires a noticeable amount of time, making large-scale brute-force attacks impractical even in an offline attack scenario.

Additionally, the use of two-factor authentication further reduces the impact of brute-force attacks. Even if an attacker successfully guesses or obtains a user’s password, access to the system is still denied unless a valid time-based one-time password is provided

## Replay Attack Considerations

Replay attacks involve capturing valid authentication data and attempting to reuse it to gain unauthorized access. In the context of two-factor authentication systems, replay attacks typically target the second authentication factor, such as one-time passwords.

The implemented system uses Time-Based One-Time Passwords (TOTP), where each code is valid only for 30 seconds. This time-based constraint significantly limits the usefulness of captured codes.

A replay attack demonstration was conducted by attempting to reuse a previously valid TOTP code after its expiration. The server-side verification logic rejected the expired code, preventing successful authentication. This demonstrates that the system is resistant to replay attacks during the second authentication phase.

It is important to note that replay attacks are only meaningful after the primary authentication step has been completed. Without valid username and password credentials, an attacker cannot reach the TOTP verification stage. The combination of password authentication and time-limited OTPs provides a layered defense against replay attempts.

## Limitations

Despite the security benefits provided by the implemented system, certain limitations remain.

First, the system does not implement rate limiting or account lockout mechanisms for login attempts. While bcrypt significantly slows brute-force attacks, additional controls, such as request throttling, would further enhance security against online attacks. Second, TOTP security assumes that the user’s authenticator device remains secure. If an attacker gains access to both the user’s password and their authenticator device, the system cannot distinguish between legitimate and malicious access.

System Demonstration

This section presents a practical demonstration of the implemented two-factor authentication system. The goal of the demonstration is to illustrate how the system operates during normal usage, from user registration to accessing protected resources, and to demonstrate the practical enhancement of security through two-factor authentication.

All demonstrations were performed using a locally deployed version of the application and test user accounts created specifically for evaluation purposes.

## Demonstration Environment

The system was implemented and tested using the following environment:

* **Backend**: Node.js with Express.js
* **Database**: SQLite
* **Authentication**: bcrypt for password hashing, TOTP for seconds-factor authentication
* **Client Interface**: HTML and CSS frontend
* **Authenticator Application**: Google Authenticator

The application was executed locally using a development server, and all communication occurred over **localhost**

How to Run the Application

To run the system locally, the following steps are required:

1. Install Node.js (version 18 or newer)
2. Clone the project repository
3. Install dependencies using:

|  |
| --- |
| npm install |

1. Start the server using:

|  |
| --- |
| node src/server.js |

# Open a web browser and navigate to:

|  |
| --- |
| http://localhost:3000 |

After starting the application, users can register, log in, enable two-factor authentication, and access protected routes such as the dashboard

## 

## 

## 

## 

## Demonstration Scenario

The demonstration follows a complete authentication lifecycle, including registration, login, two-factor verification, and access control enforcement

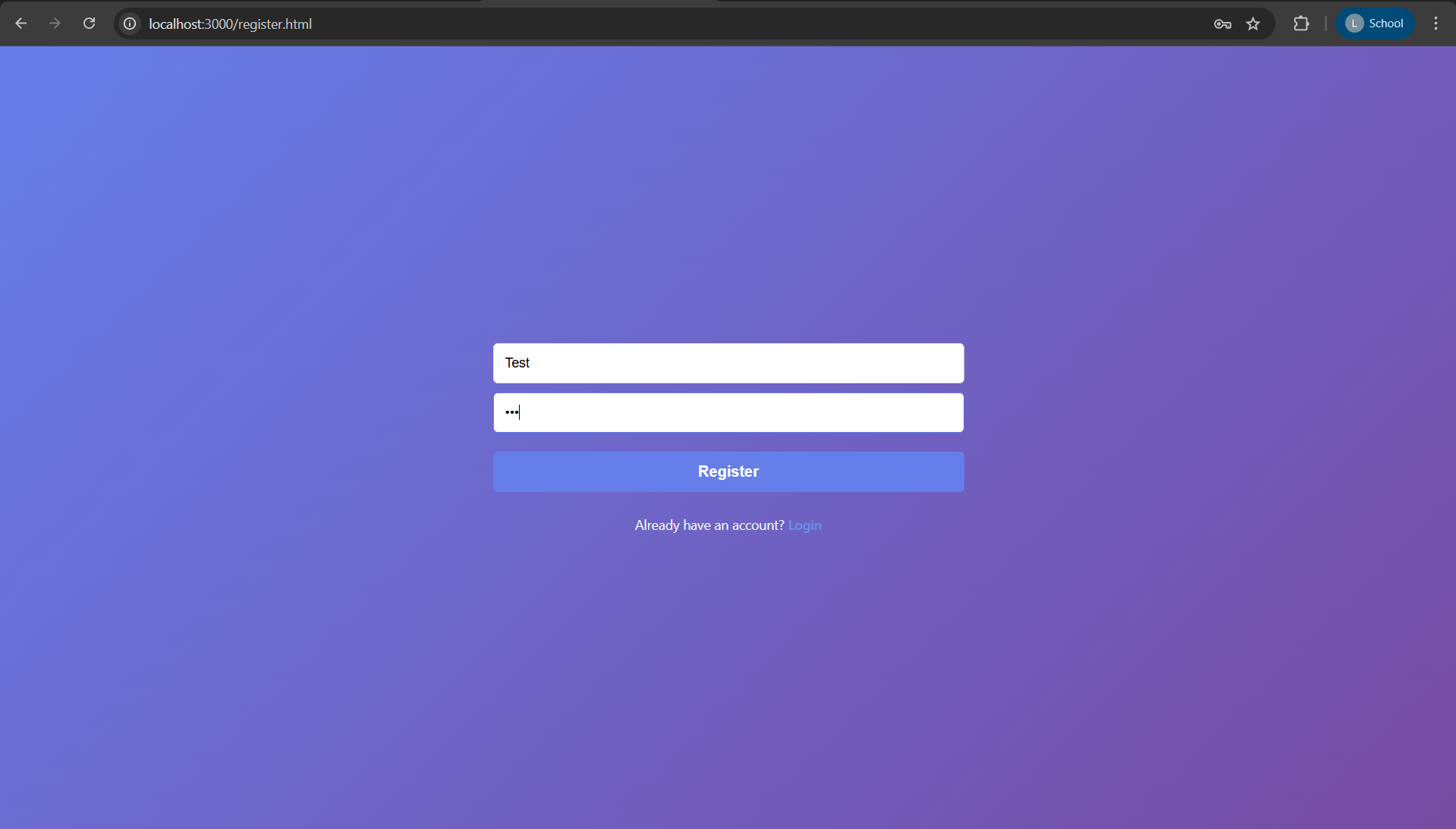
### **1. User Registration**

A new user registers by providing a username and password through the registration interface.

The server hashes the password using bcrypt before storing it in the database. No plaintext passwords are ever stored in the database.

Outcome:

The user account is successfully created with secure password storage

****

*Figure 2: Registration Page*

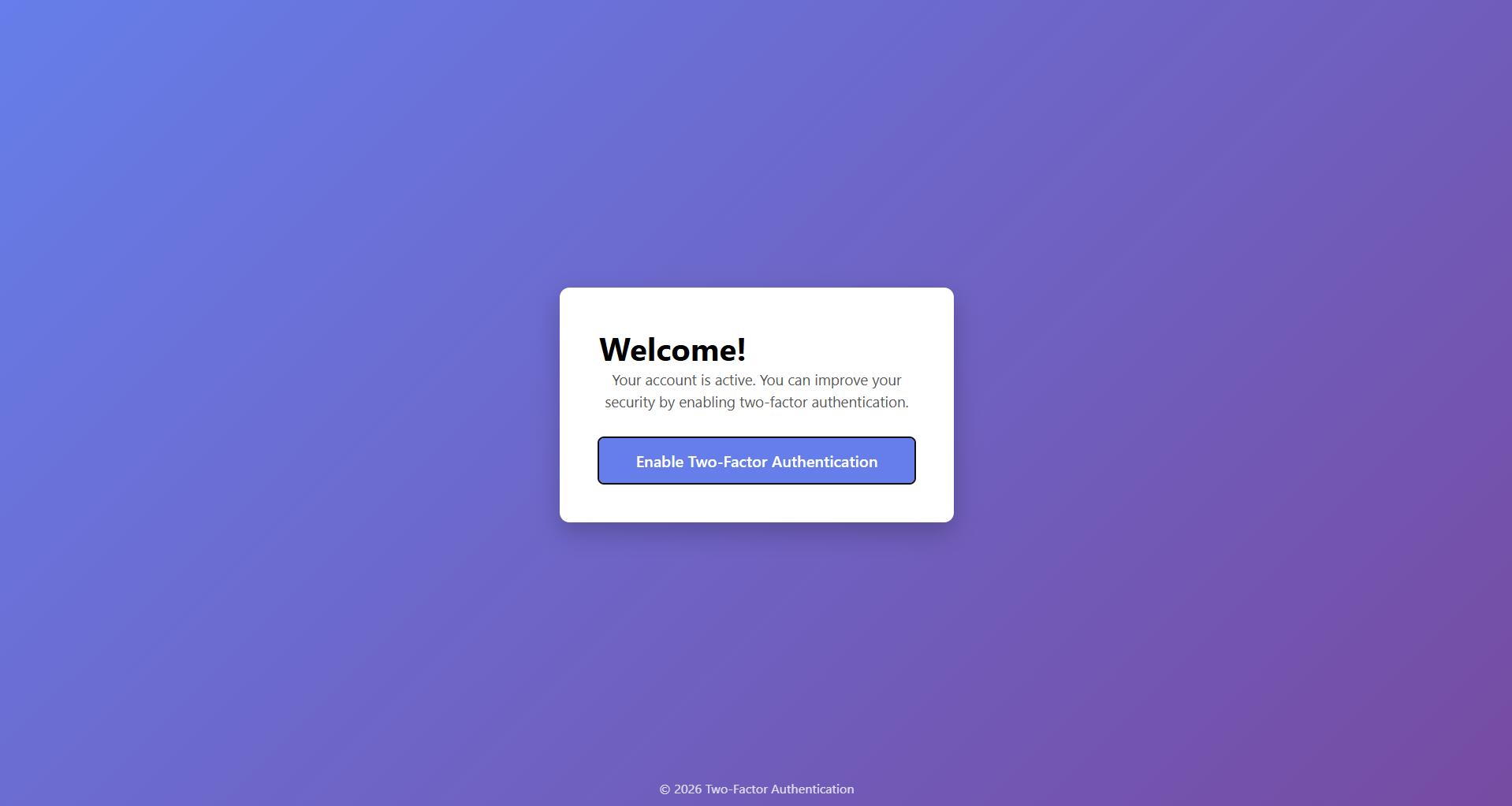
**2. Initial Login with Password**

The registered user attempts to log in using their username and password. The server retrieves the stored bcrypt hash and verifies the provided password.

If the password is valid, the server creates a session and determines whether two-factor authentication is enabled for the account.

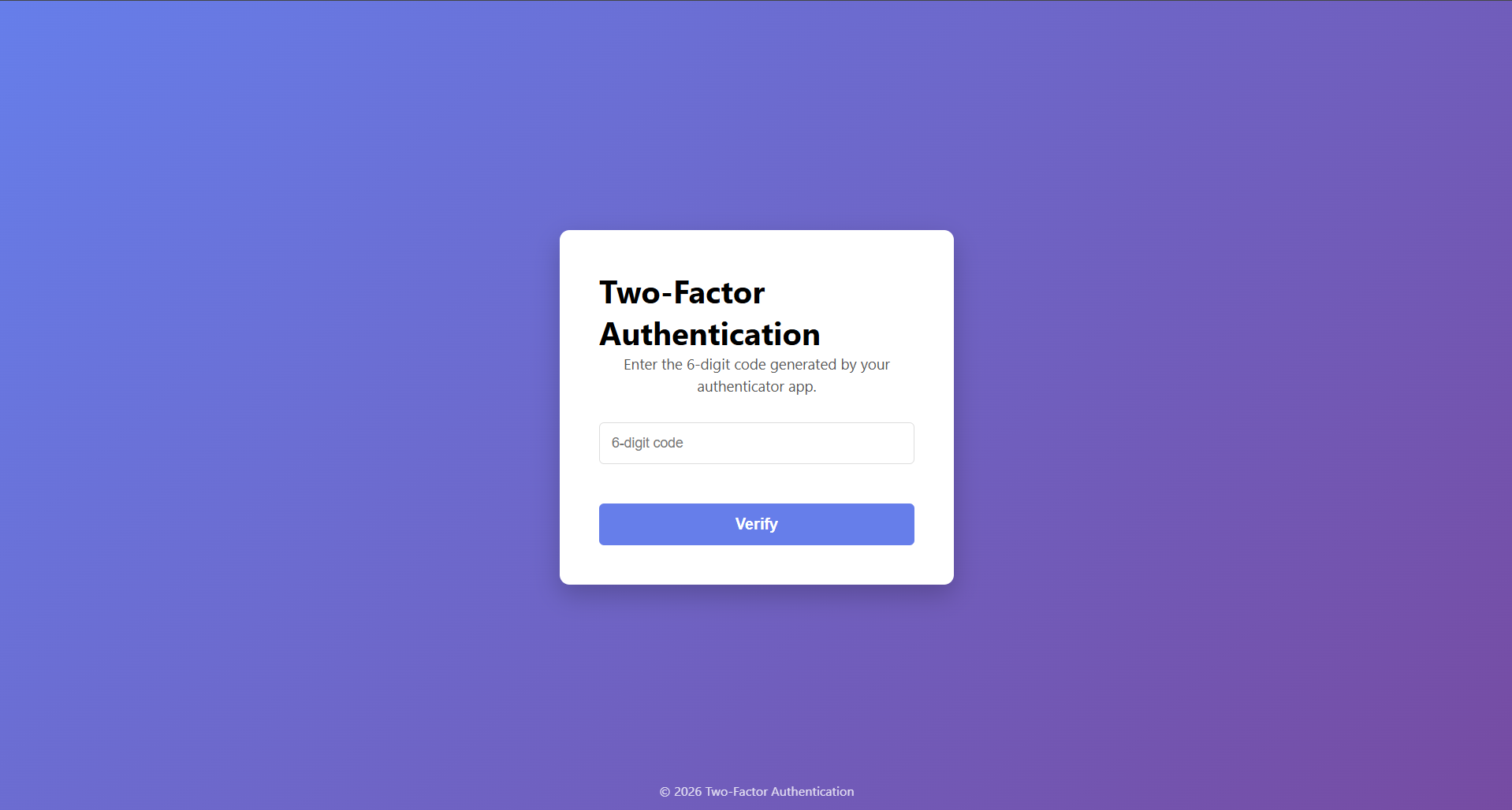
Outcome:

* If 2FA is not enabled, the user is redirected to the 2fa-dashboard



*Figure 3: Prompt to Enable 2FA*

* If 2FA is enabled, the user is redirected to the TOTP verification page



*Figure 4:TOTP Verification Page*

**3. TOTP Enrollment (Setup)**

For users who enable two-factor authentication, the system generates a unique TOTP secret.

This secret is presented to the user as a QR code, which is scanned using an authenticator application such as Google Authenticator.

The authenticator application begins generating time-based one-time passwords synchronized with the server

Outcome:

The user successfully enrolls in two-factor authentication.

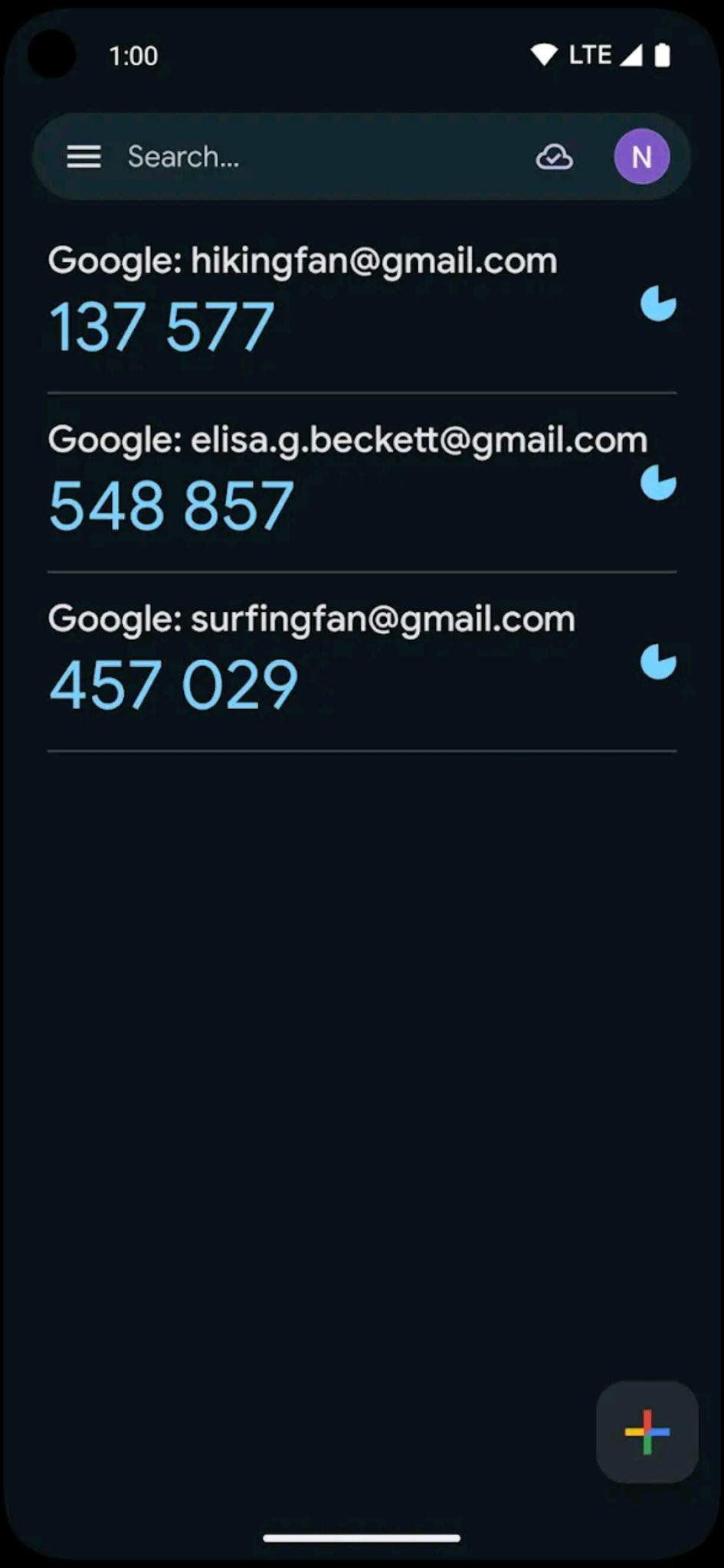


*Figure 5:TOTP Enrollment Setup Page*

**4. Two-Factor Authentication Verification**

After entering valid username and password credentials, users with 2FA enabled are prompted to enter a six-digit TOTP code generated by their authenticator application.

The server verifies the code using the shared secret and the current time window



*Figure 6: Google Authenticator*

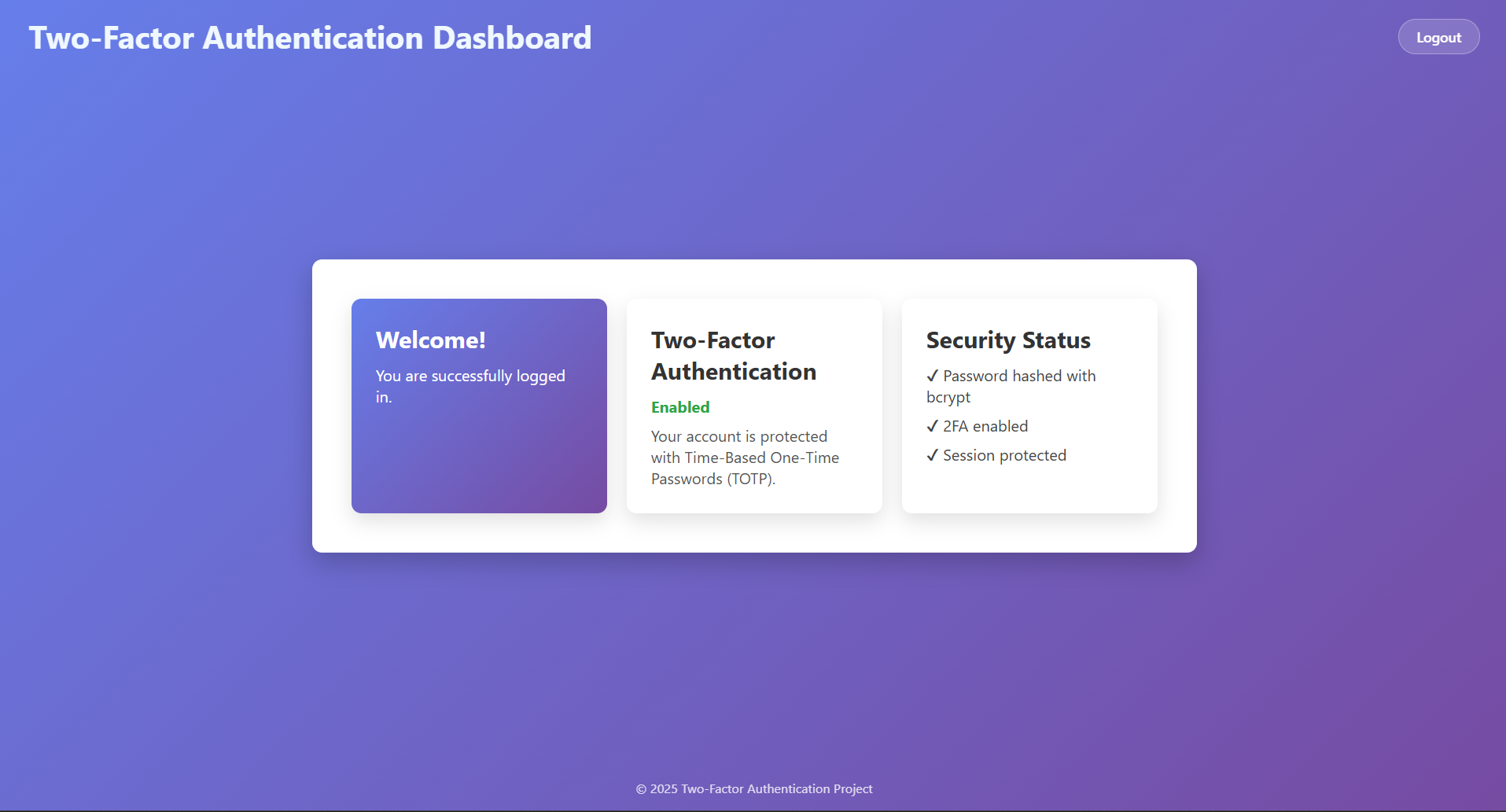
Outcome:

* If the code is valid, authentication succeeds, and access is granted
* If the code is invalid or expired, access is denied

**5. Access to Protected Resources**

Once authentication is complete, the user gains access to protected routes such as the dashboard. Access control is enforced using middleware that checks the user’s session and authentication state.

Direct attempts to access protected pages without completing authentication result in redirection to the login page.



*Figure 7: Main Dashboard*

Outcome:

Only authenticated users with completed two-factor verification can access protected resources

## 

## 

## Running Security Test Scripts

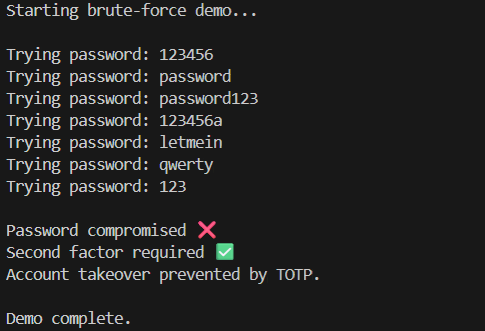
To demonstrate the system’s resistance to common attacks, controlled security test scripts were implemented in the **security-tests/** directory. The server must be running before the security test demonstrations.

**1. Brute Force Demonstration**

The brute force demonstration script can be executed using:

|  |
| --- |
| node security-tests/bruteforce-demo.js |

# This script performs repeated authentication attempts against a test account. Due to the computational cost introduced by bcrypt hashing, each attempt requires a noticeable amount of time, illustrating resistance against large-scale brute-force attacks.



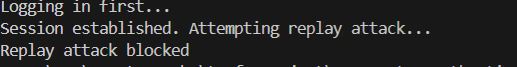
*Figure 8: Brute-Force Demonstration Output*

**2. Replay Attack Demonstration**

The replay attack simulation can be executed using:

|  |
| --- |
| node security-tests/replay-demo.js |

This script attempts to reuse expired TOTP codes to demonstrate that time-based one-time passwords cannot be successfully replayed.



*Figure 9: Replay Attack Demonstration Output*

Summary of Demonstration

The system demonstration confirms that:

* Passwords are securely stored using bcrypt hashing
* Two-factor authentication is correctly implemented using TOTP
* Expired or invalid one-time passwords are ejected
* Protected routes are inaccessible without full authentication

This demonstrates that the implemented system provides effective layered authentication and enforces access control as intended.

# 

# 

# 

# Future Improvements

The core functionality of the system has been implemented, including:

* Secure password hashing using bcrypt
* Session-based authentication
* TOTP enrollment using QR codes
* TOTP verification during login using Google Authenticator

The following aspects are currently under development or planned for future completion:

* User interface and usability improvements to enhance the overall user experience
* Additional security mechanisms, such as rate limiting, are used to protect against brute-force login attempts
* Improved session management features, including session expiration and renewal policies

Conclusion

This project presented the design and implementation of a secure web-based authentication system using Two-Factor Authentication (2FA) based on Time-Based One-Time Passwords (TOTP). The system combines traditional password-based authentication with cryptographic security mechanisms, including bcrypt hashing for password protection and HMAC-based one-time password generation for second-factor verification.

The implementation demonstrates how 2FA significantly improves security compared to password-only authentication. Even in scenarios where user credentials are compromised, unauthorized access is prevented without possession of the user’s authenticator device. The use of time-limited OTPs further strengthens security by protecting against replay attacks and reducing the effectiveness of credential theft.

Through controlled attack demonstrations, the system’s resistance to brute-force and replay attacks was evaluated. The results show that bcrypt’s adaptive hashing greatly increases the cost of password guessing attacks, while TOTP’s short validity window effectively prevents reuse of authentication codes. Middleware-based access control and session management ensure that protected resources are accessible only after successful completion of all authentication steps.

While the system provides a solid foundation for secure authentication, certain enhancements could further improve its robustness. Future work may include implementing rate limiting, account lockout policies, enhanced session expiration mechanisms, and additional monitoring features. Despite these limitations, the project successfully demonstrates the practical application of cryptographic principles in real-world authentication systems and highlights the security benefits of Two-Factor Authentication using TOTP.

# References

* [What is 2FA (Two-Factor Authentication)](https://www.ibm.com/think/topics/2fa)
* [TOTP: Time-Based One-Time Password Algorithm](https://datatracker.ietf.org/doc/html/rfc6238)
* [Password Hashing using bcrypt](https://medium.com/@bhupendra_Maurya/password-hashing-using-bcrypt-e36f5c655e09)
* [Express.js Tutorial](https://www.geeksforgeeks.org/node-js/express-js/)
* [Speakeasy: One-Time Password Library for](https://github.com/speakeasyjs/speakeasy) [Node.js](http://node.js)[.](https://github.com/speakeasyjs/speakeasy)
* [SQLite Documentation](https://www.sqlite.org/docs.html)