# Feature Specification: Multi-Factor Authentication (MFA)

## 1. Introduction

- **1.1 Feature Name:** Multi-Factor Authentication (MFA)
- **1.2 Feature ID:** FEAT-005
- **1.3 Version:** 1.0
- **1.4 Date:** June 12, 2025
- **1.5 Author(s):** Maximo Valencia
- **1.6 Stakeholders:** Product Management, Engineering Team, Security Team, Legal/Compliance (if applicable).
- **1.7 Category:** Backend, Security

## 2. Overview

This feature will enhance the security of user accounts by implementing Multi-Factor Authentication (MFA). Users will be able to enable MFA, requiring a second verification step (e.g., TOTP authenticator app or SMS code) in addition to their password, thereby significantly reducing the risk of unauthorized access.

## 3. Goals and Objectives

- **3.1 Primary Goal:** To significantly improve the security posture of user accounts against unauthorized access.
- **3.2 Objectives:**
  - Provide users with the option to enable/disable MFA.
  - Support Time-based One-Time Password (TOTP) applications (e.g., Google Authenticator, Authy).
  - Generate and allow users to store recovery codes for emergency access.
  - Guide users through the MFA setup process clearly and securely.

## 4. User Stories

- As a security-conscious user, I want to enable MFA on my account to protect it from unauthorized access.
- As a user, I want to use my preferred authenticator app (e.g., Google Authenticator) for MFA.
- As a user, I want to be provided with recovery codes in case I lose access to my MFA device.
- As a user, I want a clear and easy process to set up and manage MFA.
- As an administrator, I want to ensure our application meets industry security standards.

**5. Functional Requirements**

- **FR-5.1 MFA Enrollment:**
    - A "Multi-Factor Authentication" section in User Settings.
    - A button to "Enable MFA."
    - Upon clicking, initiate a guided flow for MFA setup (see UI/UX).
    - Display a QR code and a secret key for TOTP app configuration.
    - Require the user to verify the setup by entering a code from their authenticator app.
- **FR-5.2 Login Flow Integration:**
    - If MFA is enabled, after successful password entry, prompt the user for an MFA code.
    - Verify the entered MFA code against the stored secret.
- **FR-5.3 Recovery Codes:**
    - Generate a set of unique, single-use recovery codes during MFA setup.
    - Display these codes to the user with a strong recommendation to save them securely.
    - Allow users to generate new recovery codes (invalidating old ones) from settings.
- **FR-5.4 MFA Disabling:**

    - Allow users to disable MFA from their settings, requiring password re-entry for security.
- **FR-5.5 Session Management:**
    - Consider "remember me" options (e.g., "Don't ask for 30 days on this device") if appropriate, ensuring proper security precautions.
- **FR-5.6 Brute-Force Protection:** Implement rate limiting on MFA code verification attempts.

## 6. Non-Functional Requirements

- **NFR-6.1 Security:**
    - Store MFA secrets securely (e.g., encrypted in the database).
    - Ensure all communications during MFA setup and verification are encrypted (HTTPS).
    - Adhere to best practices for TOTP implementation (time synchronization, cryptographic strength).
- **NFR-6.2 Usability:** The MFA setup process should be as straightforward as possible to encourage adoption.
- **NFR-6.3 Performance:** MFA verification should not add significant latency to the login process.
- **NFR-6.4 Audit Logging:** Log successful and failed MFA attempts.

## 7. UI/UX Specifications

- **7.1 Settings Section:** A dedicated and clearly labeled MFA section within user settings.
- **7.2 Setup Wizard:** A multi-step wizard for MFA enrollment:
    - Step 1: Introduction to MFA.
    - Step 2: Display QR code and secret key.
    - Step 3: Verification (enter code from app).
    - Step 4: Display and confirm recovery codes.
    - Step 5: Completion message.
- **7.3 Login Prompt:** A dedicated input field for the MFA code during login.

- **7.4 Error Handling:** Clear error messages for invalid codes or attempts.
- **7.5 Instructions:** Concise instructions for using authenticator apps and managing recovery codes.

## 8. Technical Design & Implementation Details (High-Level)

- **8.1 Technology Stack:** OAuth 2.0, JavaScript
- **8.2 Backend MFA Library:** Utilize a robust server-side library for TOTP generation and verification (e.g., speakeasy for Node.js, PyOTP for Python, GoogleAuthenticator for Java/C#).
- **8.3 Database Schema:** Add fields to the users table for mfa_secret (encrypted), mfa_enabled (boolean), and a separate table for recovery_codes.
- **8.4 API Endpoints:**
    - /api/user/mfa/enable
    - /api/user/mfa/verify (for setup and login)
    - /api/user/mfa/recovery-codes/generate
    - /api/user/mfa/disable
- **8.5 Client-Side QR Code Generation:** Use a client-side library to render the QR code from the secret key.

## 9. Test Cases (Examples)

- **TC-9.1:** Verify MFA setup process is completed successfully, and user can log in with MFA.
- **TC-9.2:** Verify MFA code validation during login (correct code works, incorrect code fails).
- **TC-9.3:** Test login with a recovery code after MFA device loss (simulated).
- **TC-9.4:** Verify disabling MFA works and user can log in with just password afterward.
- **TC-9.5:** Attempt multiple invalid MFA codes to ensure rate limiting is active.

- **TC-9.6:** Verify QR code is scannable by common authenticator apps.

## 10. Open Questions / Dependencies

- Will we support SMS-based MFA in the future? (Out of scope for V1).
- What is the desired recovery code expiration policy?
- How will we handle users who lose *both* their MFA device *and* their recovery codes? (Account recovery process, likely a separate feature).