# Product Requirements Document (PRD) for Custom User Account and Login Flow

## 1. Introduction ✨

This document outlines the requirements for a custom user authentication and account management system for shuffleandsync.org. The system will provide a secure, reliable, and user-friendly experience for creating and managing user accounts, separate from the existing Twitch and Google OAuth options. Given the platform's nature, **enterprise-grade security** is a core requirement. This system will be built using **Prisma** as the ORM, connecting to a PostgreSQL database.

## 2. Goals & Objectives 🎯

- **Establish a secure, independent user authentication system:** Provide a login and registration alternative to OAuth.
- **Enforce enterprise-grade security:** Protect user data with best-in-class security practices, including robust password handling, MFA, and secure session management.
- **Create a seamless user experience:** Ensure the sign-up and login processes are intuitive and have clear feedback for the user.
- **Leverage existing infrastructure:** Integrate with our **Prisma**-based database to manage user data effectively and securely.

## 3. User Flow & Features 💻

### 3.1. User Registration Flow

1. **User Initiation:** A user clicks "Sign Up" on the homepage.
2. **Registration Form:** The user is presented with a form to enter their:
    - **Email Address:** Used as the primary unique identifier.
    - **Password:** Must meet strong password requirements (see Section 4.1).
3. **Email Verification:**
    - Upon submission, the system sends a verification email to the user's provided address.
    - The email contains a unique, **time-limited link or code**.
    - The user must click the link to verify their email. This prevents account creation with unverified emails.
4. **Account Creation:** Once the email is verified, the user's account is activated, and they are logged in.

### 3.2. User Login Flow

1. **User Initiation:** A user clicks "Log In."
2. **Login Form:** The user enters their email and password.
3. **Authentication:**
    - The system verifies the credentials.
    - If correct, the user is logged in and their session is created.

    - If incorrect, a generic error message is displayed: "Incorrect email or password." This is a security measure to prevent user enumeration attacks.
4. **Account Lockout:** After a specified number of failed login attempts (e.g., 5 within 10 minutes), the account is temporarily locked to prevent brute-force attacks.

### 3.3. Password Management

- **Forgot Password:** A user can request a password reset via email. A **time-limited, single-use token** is sent to the user's email address.
- **Change Password:** Users can change their password from their account settings. The user must enter their old password to confirm their identity.

## 4. Technical Requirements & Security 🔒

### 4.1. Password Security

- **Hashing & Salting:** Passwords must **NEVER** be stored in plaintext. They must be hashed using a **cryptographically secure, one-way algorithm** like **Argon2** (recommended), scrypt, or bcrypt. A unique, random salt must be generated for each password and stored alongside the hash.
- **Password Policy:**
    - Minimum of **12 characters**.
    - Must include a combination of uppercase letters, lowercase letters, numbers, and symbols.
    - Password blacklists should be implemented to prevent the use of common or compromised passwords.
- **Prisma Implementation:** The user model in Prisma will include a passwordHash field to store the hashed password and salt. This process must be handled server-side before the data reaches the database.

### 4.2. Multi-Factor Authentication (MFA)

- **Requirement:** MFA will be a **mandatory, opt-out feature** for all users. During the initial account setup, users will be prompted to enable it.
- **Supported Methods:**
    - **TOTP (Time-Based One-Time Password):** Users can link an authenticator app (e.g., Google Authenticator, Authy). A secret key is stored securely with the user's record in the database.
    - **WebAuthn (FIDO2):** Support for hardware security keys (e.g., YubiKey) or biometric authentication (e.g., Face ID, Touch ID). This is the highest priority for enterprise-grade security.
- **Implementation:** The Prisma schema will need to be extended to store MFA secrets, recovery codes, and device registrations.

### 4.3. Session Management

- **Stateless Authentication (JWTs):** We will use **JWT (JSON Web Tokens)** for stateless authentication.

    - Upon successful login, the server issues a short-lived **access token** and a long-lived **refresh token**.
    - The access token is stored securely in **memory** or a JavaScript variable. It's used for all API calls.
    - The refresh token is stored in a **secure, HttpOnly cookie** to protect against XSS attacks. It's used to request a new access token when the current one expires.
- **Session Revocation:** Users must be able to view and revoke active sessions from their account settings. This will require a table in the database to manage refresh token validity.

### 4.4. Prisma Schema Design

The user schema should be designed for security and scalability. Below is a conceptual representation.

```
model User {
  id              String    @id @default(uuid())
  email           String    @unique
  passwordHash    String
  isEmailVerified Boolean   @default(false)
  mfaEnabled      Boolean   @default(false)
  mfaSecret       String?
  mfaRecoveryCodes String[]
  sessions        Session[]
  // ... other user-related fields
}

model Session {
  id           String   @id @default(uuid())
  refreshToken String   @unique
  userAgent    String
  ipAddress    String
  createdAt    DateTime @default(now())
  expiresAt    DateTime
  user         User     @relation(fields: [userId], references: [id])
  userId       String
}
```

## 5. Analytics & Monitoring 📊

- **Audit Logging:** Implement comprehensive logging for all authentication-related events, including:
    - Successful and failed login attempts.
    - Account creation and deletion.
    - Password changes and resets.
    - MFA activation/deactivation.
- **Security Monitoring:** Integrate with a security information and event management (SIEM) system to monitor for suspicious activity, such as a high volume of failed logins from a single IP address.

- **User Analytics:** Track the conversion rate of the sign-up flow and identify any friction points in the user journey.

## 6. Success Metrics ✅

- **Security:** Zero reported security vulnerabilities related to the custom auth flow within the first 6 months.
- **Adoption:** A target of 15% of new users choosing the custom sign-up option within 3 months of launch.
- **User Experience:** An average time of less than 3 minutes to complete the sign-up process.
- **Performance:** All login and registration API endpoints must have a response time of less than 200ms.