Authentication Service - Product Requirements Document (PRD)

Contents

[**1. Purpose 2**](#_Toc204795266)

[**2. Introduction/Overview: 2**](#_Toc204795267)

[**Scope 2**](#_Toc204795268)

[**In Scope 2**](#_Toc204795269)

[**3. Functional Requirements: 2**](#_Toc204795270)

[**1. User Registration 2**](#_Toc204795271)

[**2. User Login 2**](#_Toc204795272)

[**3. Token Management 3**](#_Toc204795273)

[**4. Password Reset Flow 3**](#_Toc204795274)

[**5. Logout 3**](#_Toc204795275)

[**6. Two-Factor Authentication (2FA) 3**](#_Toc204795276)

[**7. User Role and Permission Management 3**](#_Toc204795277)

[**8. Current User Info (/me endpoint) 3**](#_Toc204795278)

[**9. Audit Logging 3**](#_Toc204795279)

[**10. Rate Limiting 4**](#_Toc204795280)

[**11. Service Health Check 4**](#_Toc204795281)

[**4. Sequence Diagram for Authentication Service 4**](#_Toc204795282)

[**1. Login Flow: 4**](#_Toc204795283)

[**1.2. Token Refresh Flow 5**](#_Toc204795284)

[**1.3 2FA Verification Flow 5**](#_Toc204795285)

[**1.4 Password Reset Flow 6**](#_Toc204795286)

# 1. Purpose

Enable secure, reusable authentication for internal and third-party services. This will ensure the security structure defines services. There are two types one is public and another is private. The authentication service will restrict both. The Authentication service will be connected to User or Api- Gateway. When a user tries to connect to the application the authentication service will validate the user credential and generate a token and return to the user.

# 2. Introduction/Overview:

## Scope

The Authentication Service provides a centralized identity management solution for internal and external systems. It is responsible for securely managing user credentials, authentication flows (e.g., login, token refresh), authorization via roles and permissions, and optional two-factor authentication (2FA).

The service is designed as a standalone, stateless, scalable component that can be integrated with multiple front-end clients (web, mobile, APIs) and downstream services in a microservices or modular monolith architecture.

In Scope**:**

* User registration, login, logout
* JWT-based access and refresh tokens
* Token rotation and blacklisting
* Password hashing and reset
* Role-based access control (RBAC)
* 2FA using TOTP (Google Authenticator)
* Audit logging of critical events
* OpenAPI (Swagger) compliant APIs

# 3. Functional Requirements:

The authentication service must implement the following functionality**:**

### 1. User Registration

* + Users must be able to register with a valid email and password.
  + Email uniqueness must be enforced.
  + Password must be hashed securely (bcrypt or argon2).

### 2. User Login

* + Accepts email and password for authentication.
  + On success, returns a signed JWT access token and refresh token.
  + Optionally requires TOTP if 2FA is enabled.

### 3. Token Management

* + Generates access tokens (short-lived) and refresh tokens (long-lived).
  + Validates and revokes refresh tokens.
  + Supports token rotation for enhanced security.
  + Stores refresh tokens securely (in DB or Redis).

### 4. Password Reset Flow

* + Allows user to initiate password reset via email.
  + Sends a reset token link to user’s email.
  + Resets password upon token verification and password policy validation.

### 5. Logout

* + Invalidates the refresh token (removal or blacklisting).
  + Ensures access token becomes unusable on next request if blacklist enabled.

### 6. Two-Factor Authentication (2FA)

* + Users can enable/disable 2FA.
  + Supports TOTP (e.g., Google Authenticator).
  + Requires OTP verification during login if 2FA is active.

### 7. User Role and Permission Management

* + Assign one or more roles to a user.
  + Each role may be mapped to a set of permissions.
  + Permissions control access to downstream services.

### 8. Current User Info (/me endpoint)

* + Returns profile details and assigned roles of the authenticated user.

### 9. Audit Logging

* + Logs sensitive operations (login, logout, reset password, enable/disable 2FA).
  + Each log entry includes timestamp, IP, user ID, and event type.

### 10. Rate Limiting

* + Apply rate limits to login and reset password endpoints to prevent brute force attacks.
  + Can be enforced per IP or per email/user.

### 11. Service Health Check

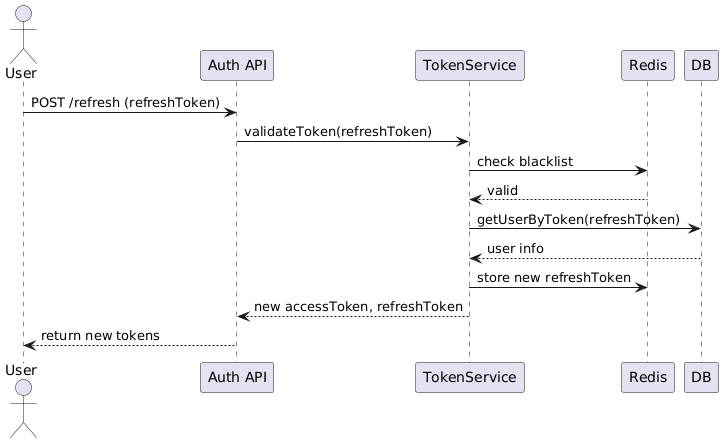
* + Exposes a /health endpoint for readiness/liveness probes**.**

## 4. Sequence Diagram for Authentication Service

### 1. Login Flow:

**Fig 3.1:** Login flow for Authentication service

### 1.2. Token Refresh Flow

****

**Fig 3.2:** Token validation sequence diagram

### 1.3 2FA Verification Flow

A diagram of a software code

AI-generated content may be incorrect.

**Fig 3.3: 2fA validation sequence**

### 1.4 Password Reset Flow

**A diagram of a software application

AI-generated content may be incorrect.**

**Fig 3.4: Password reset sequence**

# A diagram of a computer program5 Class Diagram

**Fig 5.1 Authentication Class Diagram**

# 6. Authentication Service API Design

## Authentication & Token APIs

**1 POST /auth/register**

**Registers a new user.**

**Request Body:**

**{**

**"email": "user@example.com",**

**"password": "StrongPass123!",**

**"name": "John Doe"**

**}**

**Response: 201 Created**

**{**

**"message": "Registration successful"**

**}**

* 1. **POST /auth/login**

**Authenticates user and issues tokens.**

**Request Body:**

**{**

**"email": "user@example.com",**

**"password": "StrongPass123!"**

**}**

**Response: 200 OK**

**{**

**"accessToken": "JWT\_ACCESS",**

**"refreshToken": "JWT\_REFRESH",**

**"2faRequired": true**

**}**

**1.3. POST /auth/refresh**

**Refreshes access token.**

**Request Body:**

**{**

**"refreshToken": "JWT\_REFRESH"**

**}**

**Response:**

**{**

**"accessToken": "NEW\_JWT\_ACCESS",**

**"refreshToken": "NEW\_JWT\_REFRESH"**

**}**

**1.4. POST /auth/logout**

**Logs the user out and invalidates the refresh token.**

**Request Body:**

**{**

**"refreshToken": "JWT\_REFRESH"**

**}**

**Response: 200 OK**

**{**

**"message": "Logged out successfully"**

**}**

### 2. 2FA APIs

**2.1 POST /auth/2fa/setup**

**Initiates 2FA and returns QR or secret key.**

**Response:**

**{**

**"secret": "BASE32SECRET",**

**"qrUrl": "otpauth://totp/..."**

**}**

**6. POST /auth/2fa/verify**

**Verifies 2FA code during login or setup.**

**Request Body:**

**{**

**"code": "123456"**

**}**

**Response:**

**{**

**"accessToken": "JWT\_ACCESS",**

**"refreshToken": "JWT\_REFRESH"**

**}**

### 3. Password Reset APIs

**3.1. POST /auth/forgot-password**

**Triggers email with reset link.**

**Request Body:**

**{**

**"email": "user@example.com"**

**}**

**Response:**

**{**

**"message": "Reset link sent"**

**}**

**8. POST /auth/reset-password**

**Resets user password using token.**

**Request Body:**

**{**

**"token": "reset-token-from-email",**

**"newPassword": "NewStrongPass123!"**

**}**

**Response:**

**{**

**"message": "Password has been reset"**

**}**

### 4. Role & Permission APIs

**GET /auth/roles**

**Returns list of roles (admin only).**

**POST /auth/roles**

**Create a role.**

**POST /auth/assign-role**

**Request Body:**

**{**

**"userId": "uuid",**

**"role": "ADMIN"**

**}**

**Health Check**

**GET /auth/health**

**Returns service status.**

**Response:**

**{**

**"status": "UP",**

**"uptime": "21345s"**

**}**

|  |  |  |
| --- | --- | --- |
| **Kaushik Nandy** | **Created** | **V 1.0** |
|  |  |  |
|  |  |  |