Contents

[Project Statement 2](#_Toc206335259)

[Use Cases 2](#_Toc206335260)

[Use Case Diagram 3](#_Toc206335261)

[Domain Model 4](#_Toc206335262)

[Class Diagram 4](#_Toc206335263)

[Sequence Diagram 5](#_Toc206335264)

[State Diagram 10](#_Toc206335265)

[Activity Diagram 11](#_Toc206335266)

[Component Diagram 12](#_Toc206335267)

[Deployment Diagram 13](#_Toc206335268)

[Skeleton Classes and Tables Defined 13](#_Toc206335269)

[Design Patterns 19](#_Toc206335270)

SWNEG 837 Final Project

About Me: Software Engineer Masters student at PSU World Campus and Pre-Professional Engineer working with North Point Technology LLC.  
Problem: Design a secure authentication system: Exploring techniques like password hashing, multi-factor authentication, and secure token management.

# Project Statement

This project aims to solve weak, confusing sign-in by providing a simple, secure authentication system. It will let users register with a password, log in, use multi-factor authentication (e.g., TOTP or WebAuthn), reset passwords via one-time links, manage devices/sessions. Target users are app end-users (fast, clear sign-in), developers/partner apps (easy API integration), and admins/security staff (basic controls and logs). Business goals are to reduce account takeovers, increase user trust, and make integration quick. Non-functional needs: handle spikes with horizontal scaling; keep typical responses under ~300 ms and high uptime; protect data with strong password hashing, MFA, role-based access, TLS, and encryption at rest; keep code modular with good docs and automated tests; plus basic usability and accessibility, monitoring/alerts, and sensible data retention.

# Use Cases

1. **Register Account** (email + password stored with strong hashing)
2. **Login** (password → issue temp code → complete MFA → mint tokens)
3. **Enroll / Manage MFA** (TOTP or WebAuthn-as-second-factor; backup codes)
4. **Reset Password** (email one-time link/token; rotate secrets)
5. **Logout**

# Use Case Diagram

A diagram of a computer

AI-generated content may be incorrect.

This diagram shows five capabilities the system offers to the End User—register, login (password + TOTP), enroll TOTP, reset password (via email link), and manage sessions/tokens (refresh, logout)—and the single external dependency on the Mail Service for sending reset links. It clarifies scope and actors at a glance: all flows originate from the End User, while only reset relies on the Mail Service. This keeps the system surface area small and security-focused.

# Domain Model

A computer screen shot of a computer

AI-generated content may be incorrect.

The domain centers on User with one PasswordRecord (PHC hash), optional MFAEnrollment(s) for TOTP, Session(s) representing active logins, Token(s) bound to a session (access/refresh), and OneTimeAction for password resets. Multiplicities reflect reality: one user → one password hash; one user → many sessions/tokens/enrollments; one session → many tokens. This model captures only the data needed to support the use cases, nothing extra.

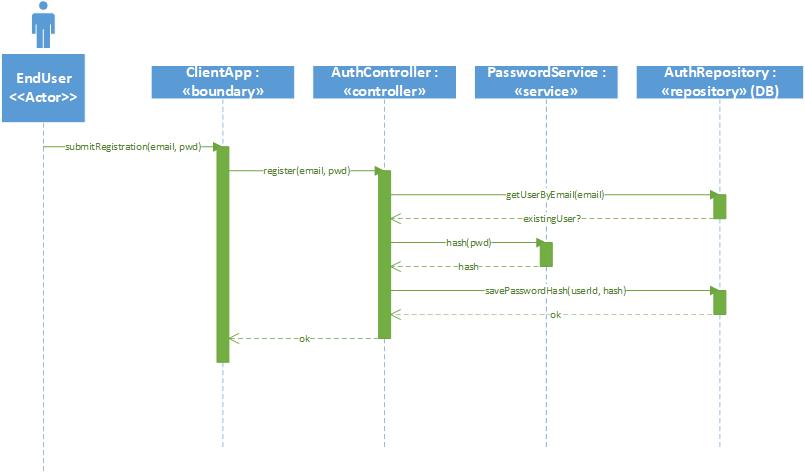
# Class Diagram

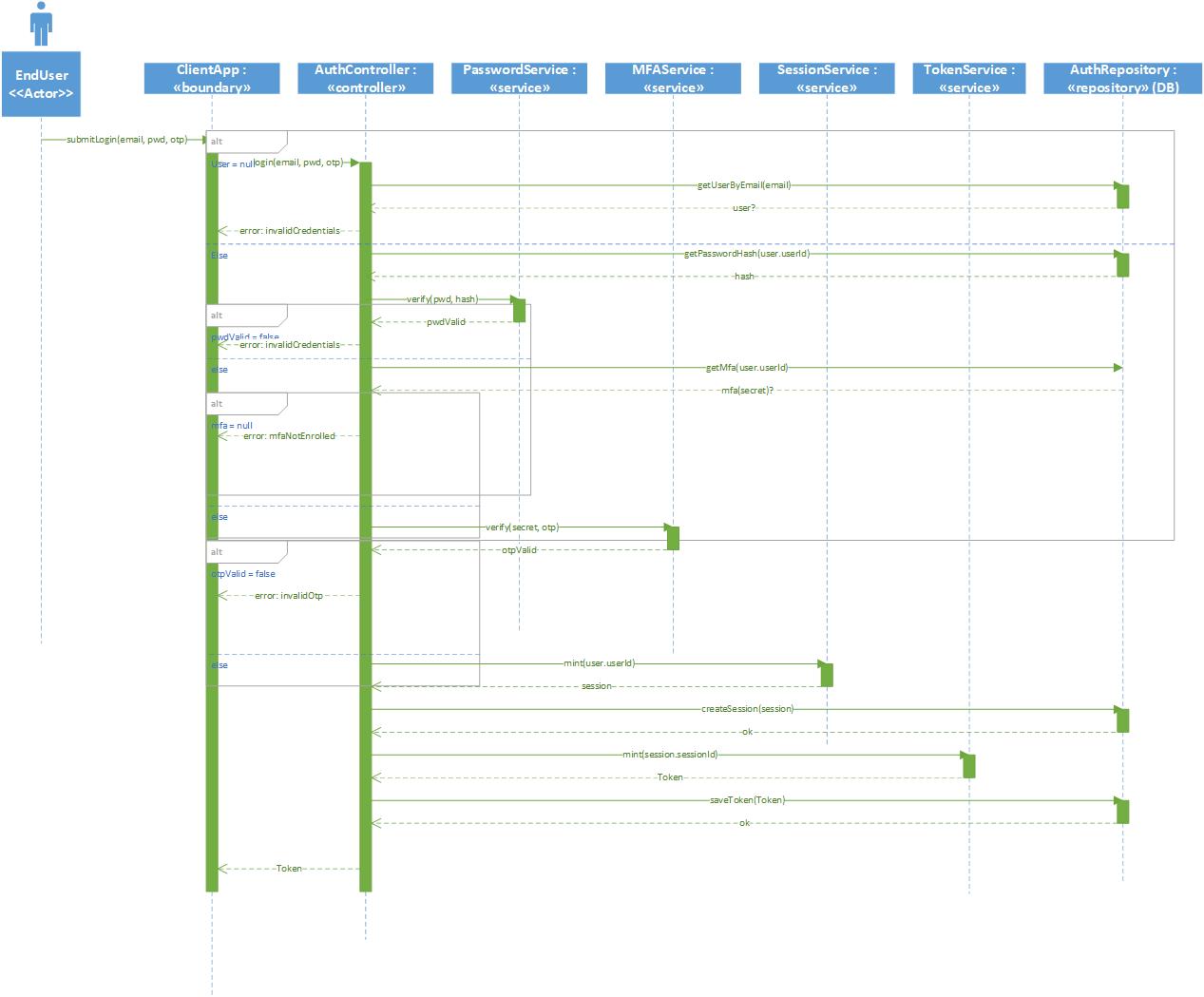
A screenshot of a computer

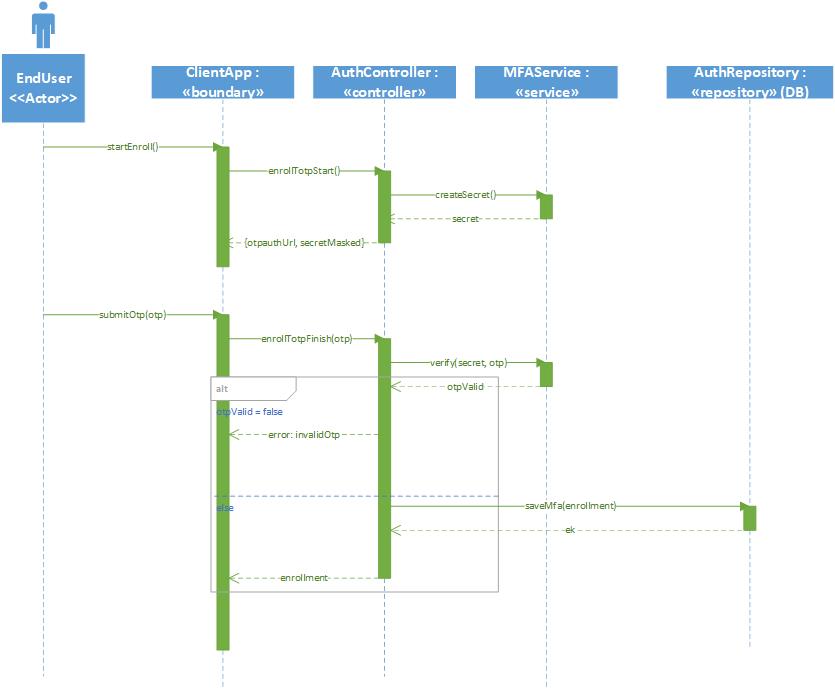
AI-generated content may be incorrect.

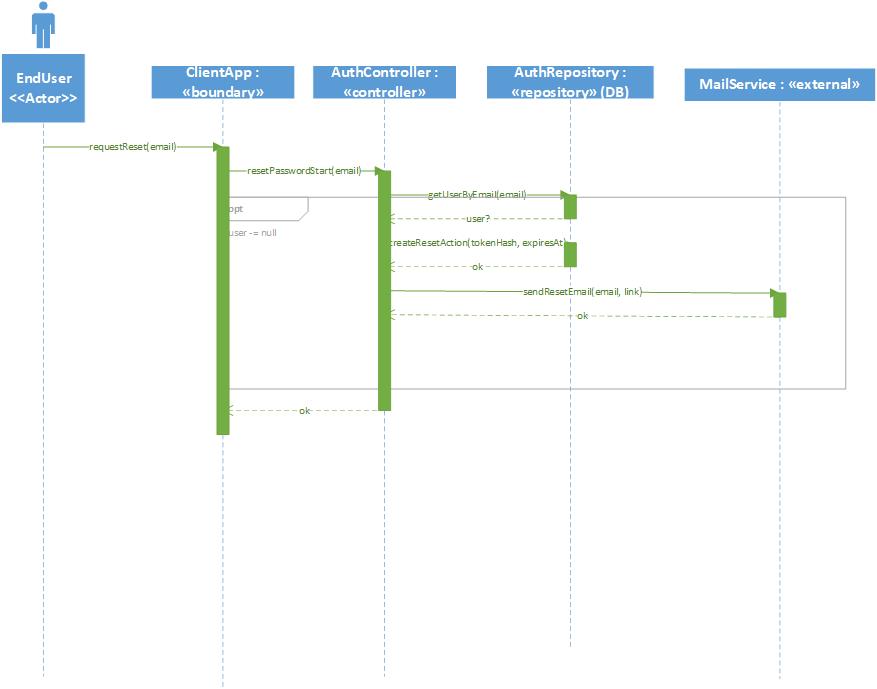
Application logic is split into a thin AuthController coordinating cohesive services: PasswordService (hash/verify), MFAService (TOTP verify), SessionService (create/revoke sessions), TokenService (mint/rotate/revoke tokens). A unified AuthRepository abstracts persistence for all entities. The controller depends on service/repository interfaces, keeping the code testable and easy to evolve (e.g., swap hashing or token format).

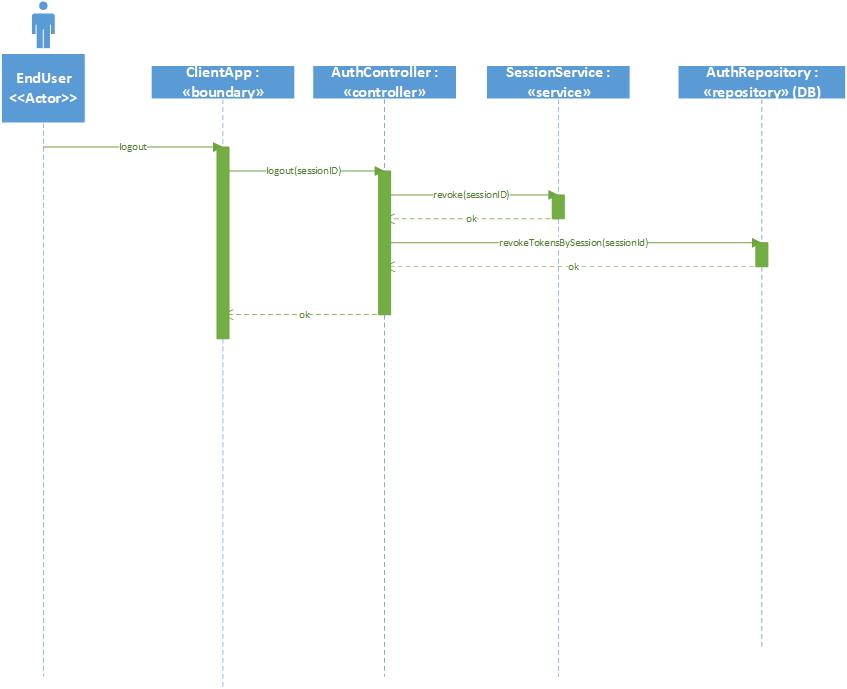
# Sequence Diagram





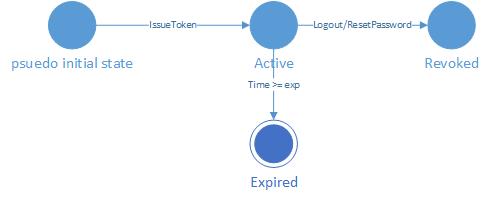






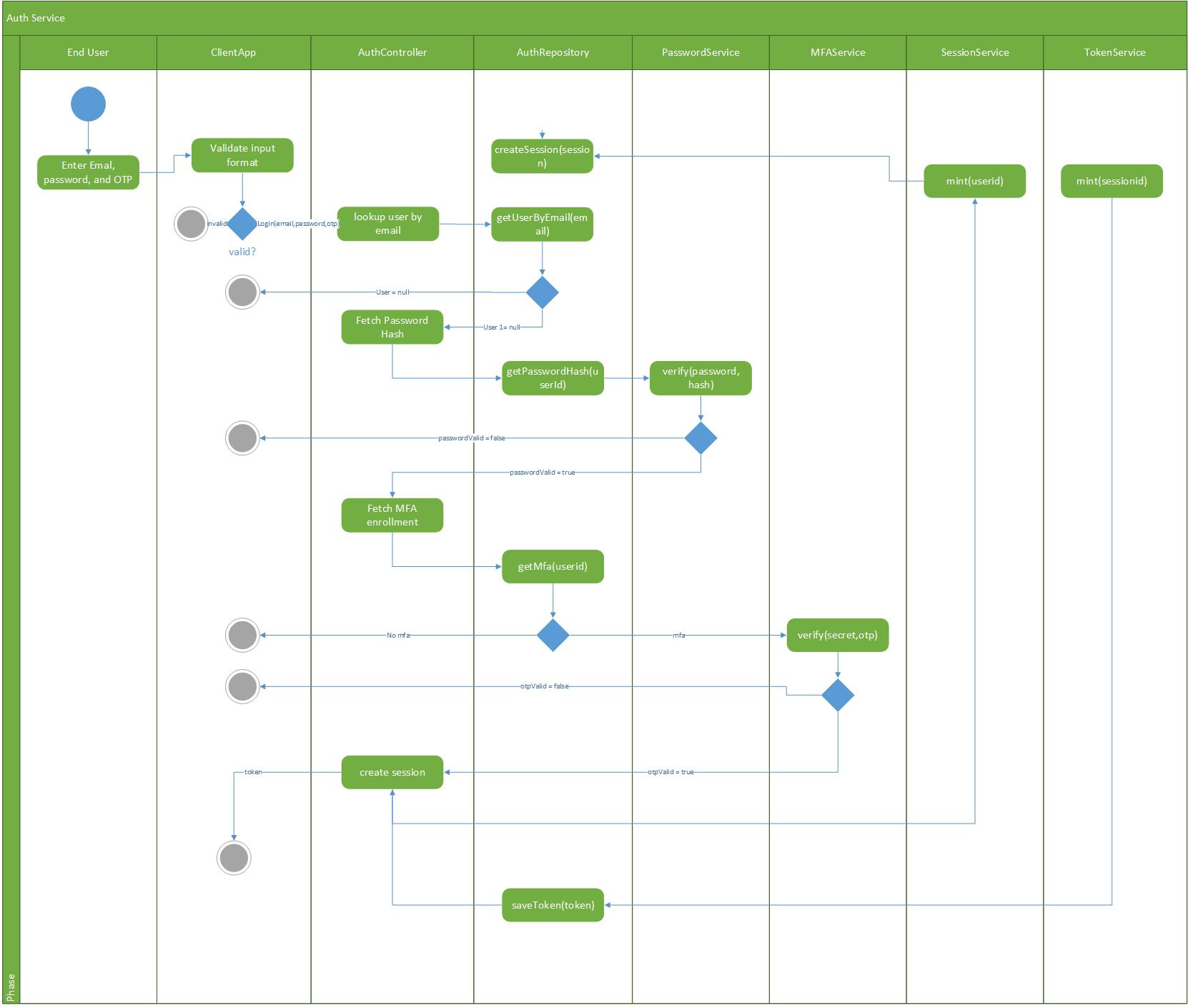
Each framed interaction shows the happy path and guarded error branches. *Login* (single call with email/password/otp) loads the user and hash, verifies password and TOTP, then creates a session and issues an access + refresh token pair. *Enroll TOTP* persists a verified seed; *Reset Password* creates and later consumes a one-time action, updating the password hash and revoking tokens; *Refresh* rotates tokens one-time; *Logout* revokes a session’s tokens.

# State Diagram



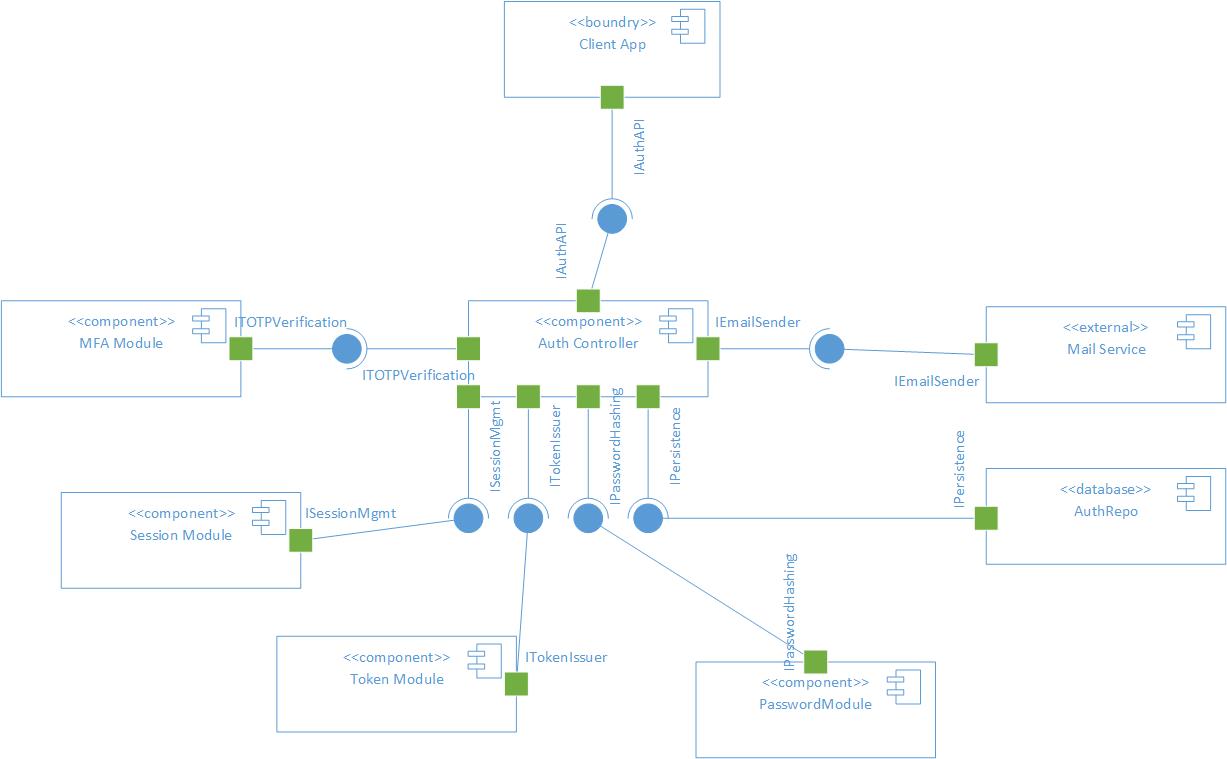
Tokens start Active, move to Revoked on logout/rotation/reset, and naturally reach Expired when their TTL elapses. Modeling this lifecycle makes revocation rules explicit and ties directly to the revoked flag and token expiry enforcement.

# Activity Diagram



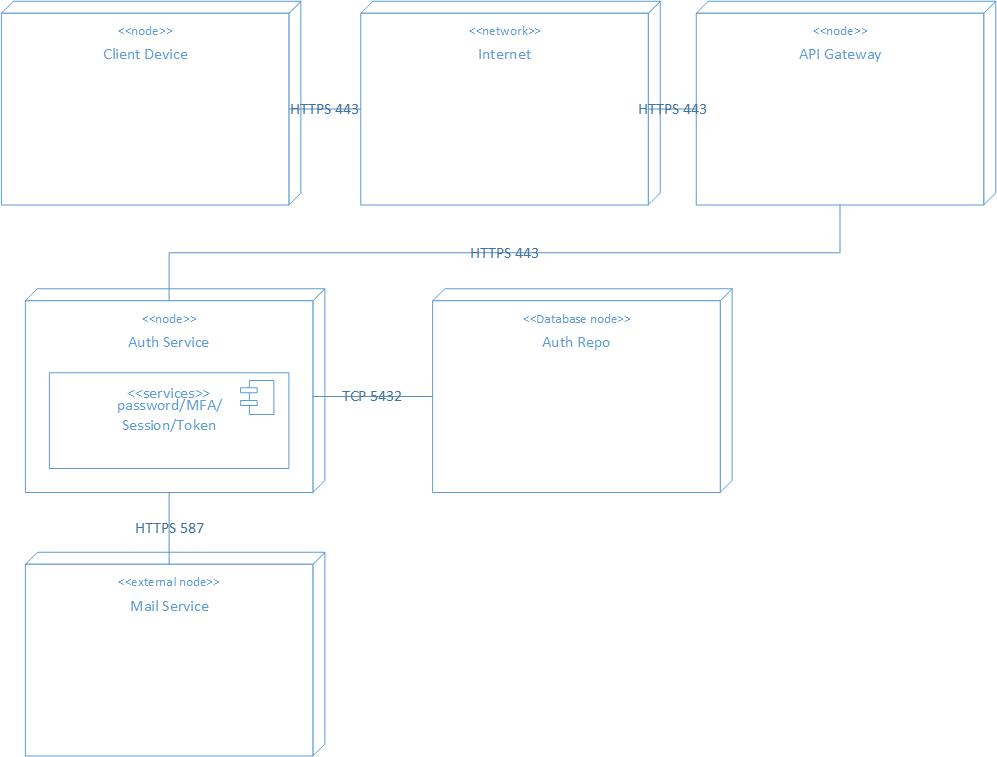
The swimlanes show responsibility hand-offs: End User → Client App → AuthController, with calls to PasswordService and MFAService for verification, then Session/Token services to establish the authenticated session. Decision nodes handle invalid input, unknown email, bad password, and bad OTP, ensuring every failure path terminates cleanly.

# Component Diagram



The Auth API provides IAuthAPI to the Client App and requires IPasswordHashing, ITOTPVerification, ISessionMgmt, ITokenIssuer, IPersistence, and IEmailSender. Password/MFA/Session/Token modules provide those interfaces; AuthRepository provides IPersistence and talks to Auth DB (IRelationalStore), and Mail Service provides IEmailSender. Matching names make the wiring unambiguous.

# Deployment Diagram



A stateless Auth Service runs behind an HTTPS load balancer, storing state in a private Auth DB and calling an external Mail Service to send reset links. This keeps the runtime simple (one deployable), secure (TLS, private DB), and scalable (horizontal replicas).

# Skeleton Classes and Tables Defined

User

| **Attribute** | **Type** | **Constraints / Notes** |
| --- | --- | --- |
| userId | UUID | **PK** |
| email | String | Unique login identifier |

**PasswordRecord**

| **Attribute** | **Type** | **Constraints / Notes** |
| --- | --- | --- |
| userId | UUID | **PK**, **FK → users.user\_id** (1:1) |
| hash | String | Store full **PHC** string (e.g., Argon2id) |

**MFAEnrollment (TOTP)**

| **Attribute** | **Type** | **Constraints / Notes** |
| --- | --- | --- |
| enrollmentId | UUID | **PK** |
| userId | UUID | **FK → users.user\_id** |
| secret | Bytes | TOTP seed (encrypt at rest) |

**Session**

| **Attribute** | **Type** | **Constraints / Notes** |
| --- | --- | --- |
| sessionId | UUID | **PK** |
| userId | UUID | **FK → users.user\_id** |

**Token**

| **Attribute** | **Type** | **Constraints / Notes** |
| --- | --- | --- |
| tokenId | UUID | **PK** |
| sessionId | UUID | **FK → sessions.session\_id** |
| type | String | Enum: access | refresh |
| revoked | Boolean | Default false |

**OneTimeAction (Password Reset)**

| **Attribute** | **Type** | **Constraints / Notes** |
| --- | --- | --- |
| actionId | UUID | **PK** |
| userId | UUID | **FK → users.user\_id** |
| tokenHash | String | Hash of emailed reset token |
| expiresAt | Time | TTL expiry |

**AuthController**

| **Method** | **Parameters** | **Returns** | **Purpose** |
| --- | --- | --- | --- |
| register | email: String, password: String | void | Create user + password hash |
| login | email: String, password: String, otp: String | Tokens | Verify pwd + TOTP → issue tokens |
| enrollTotpStart | — | TotpStart | Create TOTP secret (QR/URI) |
| enrollTotpFinish | otp: String | MFAEnrollment | Verify first OTP & persist enrollment |
| logout | sessionId: UUID | void | Revoke session + its tokens |

**PasswordService**

| **Method** | **Parameters** | **Returns** | **Notes** |
| --- | --- | --- | --- |
| hash | password: String | String (PHC) | Produce Argon2id PHC string |
| verify | password: String, phcHash: String | boolean | Constant-time verify |

**MFAService**

| **Method** | **Parameters** | **Returns** | **Notes** |
| --- | --- | --- | --- |
| createSecret | — | byte[] | TOTP seed |
| verify | secret: byte[], code: String | boolean | TOTP check with small drift |

**SessionService**

| **Method** | **Parameters** | **Returns** | **Notes** |
| --- | --- | --- | --- |
| mint | userId: UUID | Session | Create session anchor |
| revoke | sessionId: UUID | void | End session |

**TokenService**

| **Method** | **Parameters** | **Returns** | **Notes** |
| --- | --- | --- | --- |
| mint | sessionId: UUID | Token | Short TTL |
| rotate | oldRefresh: String | Tokens | Atomically issue new pair & invalidate old refresh |
| revokeBySession | sessionId: UUID | void | Invalidate all tokens for a session |

**AuthRepository (Unified Persistence)**

| **Area** | **Method** | **Signature** | **Purpose** |
| --- | --- | --- | --- |
| Users | createUser | (u: User) → void | Insert user |
|  | getUserByEmail | (email: String) → User? | Lookup by email |
| Passwords | savePasswordHash | (userId: UUID, phcHash: String) → void | Upsert hash |
|  | getPasswordHash | (userId: UUID) → String? | Fetch PHC |
| MFA | saveMfa | (e: MFAEnrollment) → void | Insert enrollment |
|  | getMfa | (userId: UUID) → MFAEnrollment? | Get current TOTP |
| Sessions | createSession | (s: Session) → void | Insert session |
|  | revokeSession | (sessionId: UUID) → void | Mark session closed (if modeled) |
| Tokens | saveToken | (t: Token) → void | Insert token |
|  | revokeTokensBySession | (sessionId: UUID) → void | Bulk invalidate |
|  | isTokenRevoked | (tokenId: UUID) → boolean | Check status |
| Reset | createResetAction | (a: OneTimeAction) → void | Insert reset action |
|  | consumeResetAction | (tokenHash: String) → OneTimeAction? | Fetch+delete if valid/unexpired |

Tables:  
**users**

| **Column** | **Type** | **Constraints / Index** |
| --- | --- | --- |
| user\_id | UUID | **PK** |
| email | VARCHAR(320) | **UNIQUE**, idx\_users\_email |

**password\_records**

| **Column** | **Type** | **Constraints / Index** |
| --- | --- | --- |
| user\_id | UUID | **PK**, **FK → users(user\_id)** ON DELETE CASCADE |
| hash | TEXT | PHC string |

**mfa\_enrollments**

| **Column** | **Type** | **Constraints / Index** |
| --- | --- | --- |
| enrollment\_id | UUID | **PK** |
| user\_id | UUID | **FK → users(user\_id)** ON DELETE CASCADE, idx\_mfa\_user |
| secret | BYTEA | TOTP seed (encrypt at rest) |

**sessions**

| **Column** | **Type** | **Constraints / Index** |
| --- | --- | --- |
| session\_id | UUID | **PK** |
| user\_id | UUID | **FK → users(user\_id)** ON DELETE CASCADE, idx\_sessions\_user |

**tokens**

| **Column** | **Type** | **Constraints / Index** |
| --- | --- | --- |
| token\_id | UUID | **PK** |
| session\_id | UUID | **FK → sessions(session\_id)** ON DELETE CASCADE, idx\_tokens\_session |
| type | VARCHAR(7) | CHECK (type IN ('access','refresh')) |
| revoked | BOOLEAN | DEFAULT FALSE |

**one\_time\_actions**

| **Column** | **Type** | **Constraints / Index** |
| --- | --- | --- |
| action\_id | UUID | **PK** |
| user\_id | UUID | **FK → users(user\_id)** ON DELETE CASCADE, idx\_actions\_user\_expires |
| token\_hash | TEXT | Hash of emailed token |
| expires\_at | TIMESTAMP | Expiry |

# Design Patterns

- GRASP: *Controller* (AuthController) orchestrates use cases; *Information Expert* places logic in Password/MFA/Token/Session services.

- SOLID: *SRP* keeps modules cohesive; *DIP/OCP* let you swap hashing or token strategy without API changes.

- GoF: *Strategy* (hashing/token), *Repository* (AuthRepository).

- Security tactics: PHC (Argon2id), TOTP, short-lived access + rotating refresh, token revocation, TLS, least privilege.