Registration Module Documentation (Draft)

Kamil Grzymkowski (151908)

2026-01-11

Contents

1 Kamil Grzymkowski (151908), Application Security

- **Course**: Application Security
- **Level**: Graduate/Master's
- **Focus Areas**:
  - Secure user registration
  - Password security with Argon2 hashing
  - Email verification with secure tokens
  - Input validation (frontend and backend)
  - Cryptographic token generation and storage
  - Cross-platform validation via WebAssembly

1.1 Exercise Description

This exercise demonstrates a secure user registration module with email verification, implementing modern web security practices. The registration module is responsible for:

- Secure user registration with client-side and server-side validation
- Password strength validation with detailed scoring (0–7 scale)
- Cryptographically secure verification token generation
- Email verification workflow before account activation
- Password hashing using Argon2 algorithm
- Defense against common attacks (enumeration, injection, brute force)
- Shared validation logic between frontend (WASM) and backend (native Rust)

2 Component Description

2.1 Purpose

The registration module provides a secure mechanism for creating user accounts and verifying ownership of email addresses before granting access to the application.

Its main goals are:

1. Ensure only valid, well-formed data is accepted through dual validation (frontend and backend).
2. Prevent user enumeration and information leakage through generic error responses.
3. Protect passwords using Argon2 hashing with random salts.
4. Store verification tokens as SHA256 hashes, never in plaintext.
5. Enforce email verification before the account can be used.
6. Automatically clean up expired tokens and unverified accounts.

2.2 Responsibilities

- Accept registration requests with username, email, and password.
- Validate input on both frontend (via WASM) and backend (native Rust).
- Hash passwords using Argon2 with random salts before storage.
- Generate 32-byte cryptographically secure verification tokens.
- Store only SHA256 hashes of tokens in the database.
- Send verification emails via SMTP (MailHog in development).
- Verify email tokens and mark users as verified.

- Clean up expired tokens and unverified users automatically.
- Provide translated error messages via the translator crate.

2.3 Data Collected

The registration module stores the minimal data required for account creation and email verification.

User Login Table (`user_login`)

- `user_id`: INTEGER, primary key, auto-increment
- `username`: TEXT, unique, not null
- `email`: TEXT, unique, not null
- `password`: TEXT (Argon2 hash, nullable for password reset state)
- `email_verified`: INTEGER (boolean), default 0
- `email_verified_at`: INTEGER (timestamp), nullable
- `password_reset`: INTEGER (boolean), default 0

User Data Table (`user_data`)

- `user_id`: INTEGER, primary key, foreign key to `user_login` with CASCADE
- `counter`: INTEGER, default 0 (application-specific data)

Email Verification Tokens Table (`email_verification_tokens`)

- `user_id`: INTEGER, primary key, foreign key to `user_login` with CASCADE
- `token_hash`: TEXT (SHA256 hash of the token)
- `expires_at`: INTEGER (Unix timestamp)
- `created_at`: INTEGER (Unix timestamp)

2.4 Security Assumptions

- All traffic is served over HTTPS in production.
- Database is encrypted using SQLCipher with a 32-byte key stored in system keyring or environment variable.
- Email delivery is handled via SMTP (MailHog for development, production SMTP for deployment).
- Verification tokens are sent only to the user's email address.
- Passwords are never logged, stored in plaintext, or returned to the client.
- Frontend validation is for user experience; backend validation is authoritative.
- Expired tokens and unverified users are cleaned up automatically every hour.

3 Component Requirements

3.1 Functional Requirements

3.1.1 FR-1: User Registration

ID: FR-1

Description: The system must allow a new user to register with username, email, and password.

Details:

- The frontend provides a registration form with real-time validation.
- Username must be 3–20 characters, printable UTF-8 only.
- Email must be valid format.
- Password must be 8–64 characters with uppercase, lowercase, digit, and special character.
- Returns specific error codes for duplicate username/email and validation failures.

3.1.2 FR-2: Email Verification

ID: FR-2

Description: The system must require email verification before the account is active.

Details:

- On successful registration, a secure verification token is generated.
- Token hash is stored in database (plaintext never stored).
- Verification email sent via SMTP with verification link.
- Token has configurable expiry.
- Verification endpoint validates token and marks user as verified.
- Token is deleted after successful verification.

3.1.3 FR-3: Input Validation

ID: FR-3

Description: All registration inputs must be validated on both frontend and backend.

Details:

- Shared validation logic compiled to WASM for frontend use.
- Username: 3–20 characters, printable UTF-8.
- Email: Valid format.
- Password: 8–64 characters with complexity requirements.
- Password strength score calculated with visual indicator.
- Validation errors translated to user-friendly messages.

3.1.4 FR-4: Error Handling

ID: FR-4

Description: The system must provide clear, translated error messages.

Details:

- Error responses use typed error codes.
- Validation errors include field-specific error codes.
- All error codes translated with localization support.
- Server errors logged internally; clients receive generic error.

3.1.5 FR-5: Token Lifecycle

ID: FR-5

Description: Verification tokens must be single-use and time-limited.

Details:

- Each user can have only one active verification token.
- Tokens expire after configurable duration.
- Expired tokens are automatically cleaned up.
- Unverified users with expired tokens are deleted.
- Token is deleted upon successful verification.

3.2 Non-Functional Requirements

3.2.1 NFR-1: Security

ID: NFR-1

Description: All security best practices must be followed.

Details:

- Passwords hashed with memory-hard algorithm.
- Verification tokens stored as hashes only.
- Database encrypted at rest.
- HTTPS enforced in production.
- Input validated on both frontend and backend.

3.2.2 NFR-2: Reliability

ID: NFR-2

Description: The module must be fault-tolerant.

Details:

- On email sending failure, user record is rolled back.
- Foreign keys ensure referential integrity.
- Automatic cleanup of expired tokens.

3.2.3 NFR-3: Usability

ID: NFR-3

Description: The module must provide clear user feedback.

Details:

- Real-time field validation on frontend.
- Password strength indicator.
- Translated error messages for all validation failures.
- Clear success/error states on all pages.

4 Component Architecture

4.1 Technology Stack

Frontend

- Vue 3 with TypeScript
- Vuetify for UI components
- Pinia for state management
- WASM modules for validation and translation (shared with backend)

Backend

- Rust with Axum web framework
- SQLite with SQLCipher encryption
- Argon2 for password hashing
- SMTP for email sending

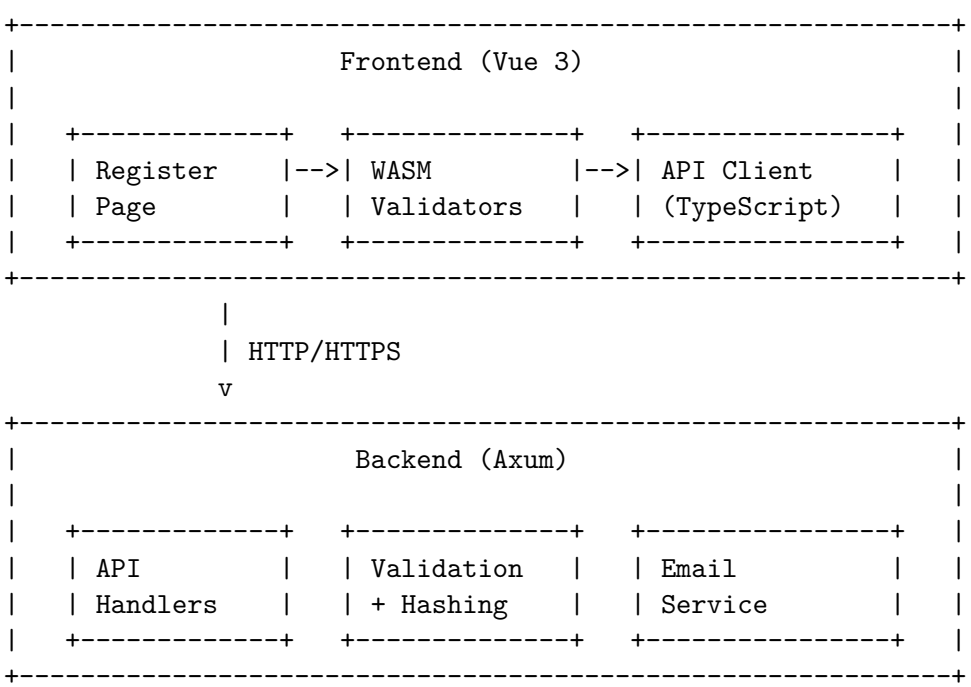
Shared Components

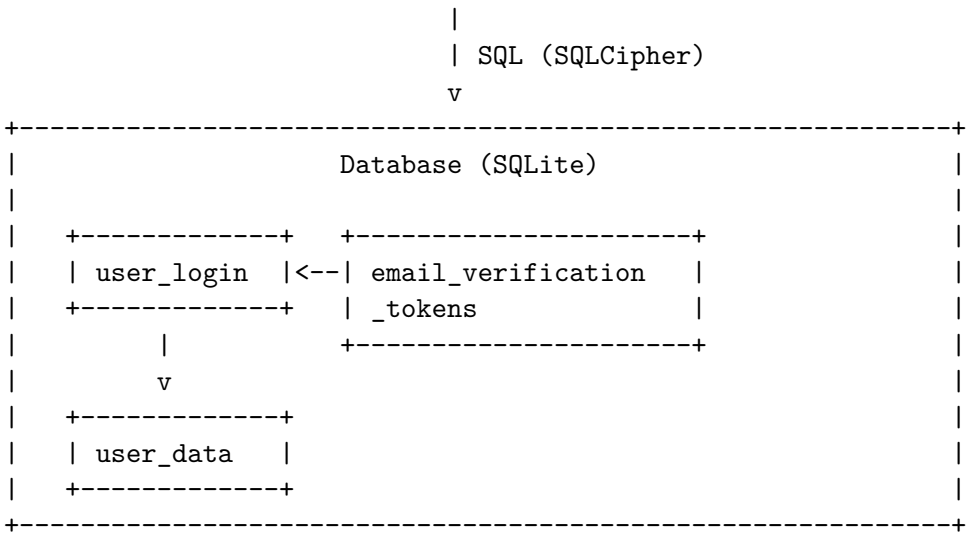
- `api-types`: Request/response type definitions
- `field-validator`: Input validation logic (compiled to native and WASM)
- `translator`: Error message translation

Database

- SQLite with SQLCipher encryption
- Foreign keys with CASCADE delete

4.2 High-Level Architecture Diagram





5 Database Structure

5.1 User Login Table (`user_login`)

Column	Type	Constraints	Notes
`user_id`	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique user identifier
`username`	TEXT	UNIQUE, NOT NULL	Indexed for lookup
`email`	TEXT	UNIQUE, NOT NULL	Stored normalized (lowercase)
`password`	TEXT	NOT NULL	Argon2 hash in PHC format
`email_verified`	INTEGER	NOT NULL, DEFAULT 0	Boolean: 0=false, 1=true
`email_verified_at`	INTEGER	NULLABLE	Unix timestamp when verified

Constraints and notes:

- Unique constraints on `username` and `email` prevent duplicates.
- `email_verified` must be true before user can log in.

5.2 User Data Table (`user_data`)

Column	Type	Constraints	Notes
`user_id`	INTEGER	PRIMARY KEY, FOREIGN KEY → `user_login.user_id`	CASCADE on delete
`counter`	INTEGER	NOT NULL, DEFAULT 0	Application-specific data

Notes:

- One-to-one relationship with `user_login`.
- Created automatically during registration.
- Cascading delete ensures cleanup when user is removed.

5.3 Email Verification Tokens Table (`email_verification_tokens`)

Column	Type	Constraints	Notes
`user_id`	INTEGER	PRIMARY KEY, FOREIGN KEY → `user_login.user_id`	CASCADE on delete
`token_hash`	TEXT	NOT NULL	SHA256 hash of verification token
`expires_at`	INTEGER	NOT NULL	Unix timestamp for expiry
`created_at`	INTEGER	NOT NULL	Unix timestamp when created

Constraints and guarantees:

- Primary key on `user_id` ensures one token per user.
- Only SHA256 hash is stored; plaintext token is sent via email only.
- Token is valid only if `expires_at > current_time`.
- Token is deleted after successful verification.
- Cascading delete removes token when user is deleted.

6 UML Sequence Diagrams

6.1 Registration Flow Sequence Diagram

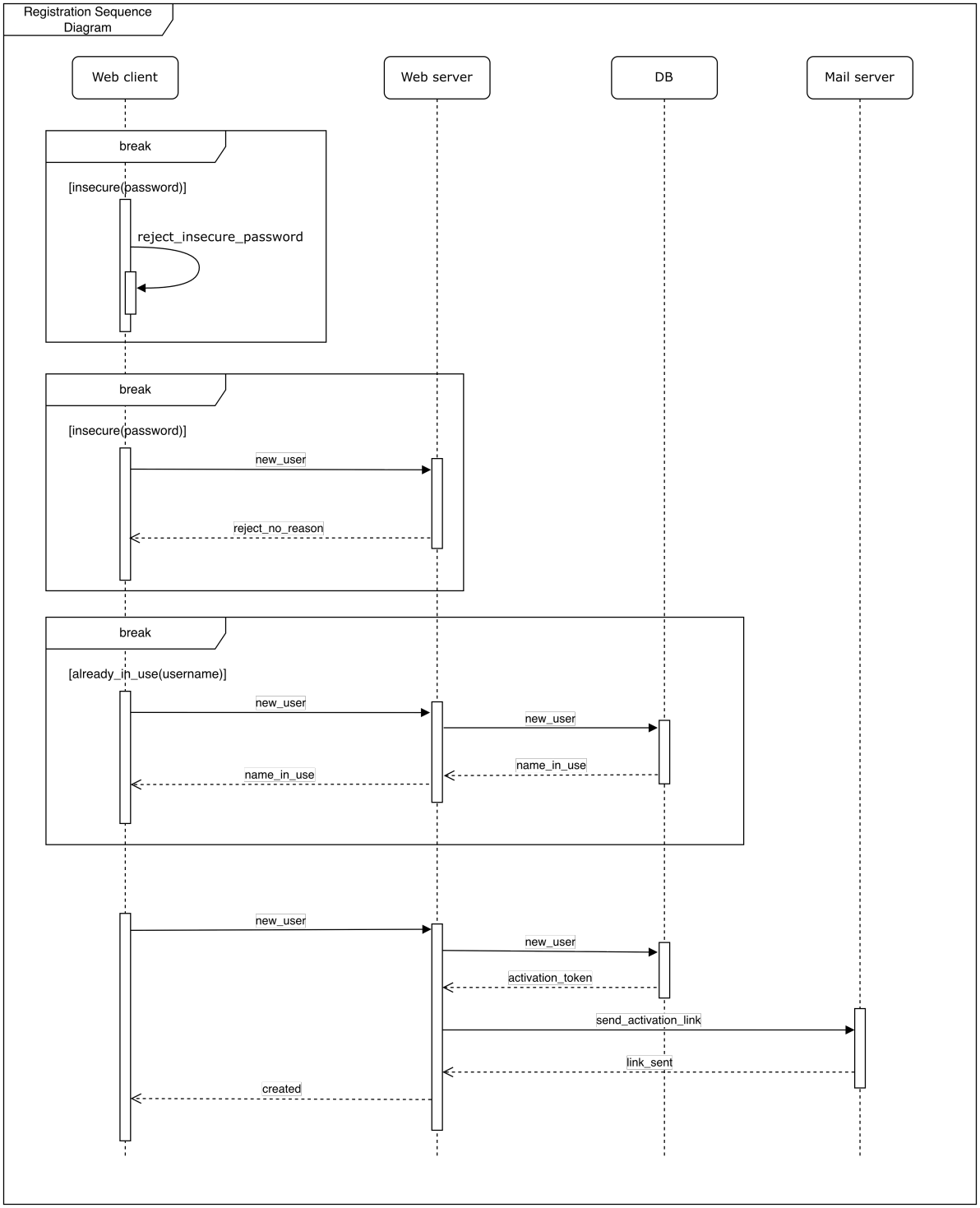


Figure 1: Registration Flow Sequence Diagram

6.2 Verification / Activation Flow Sequence Diagram

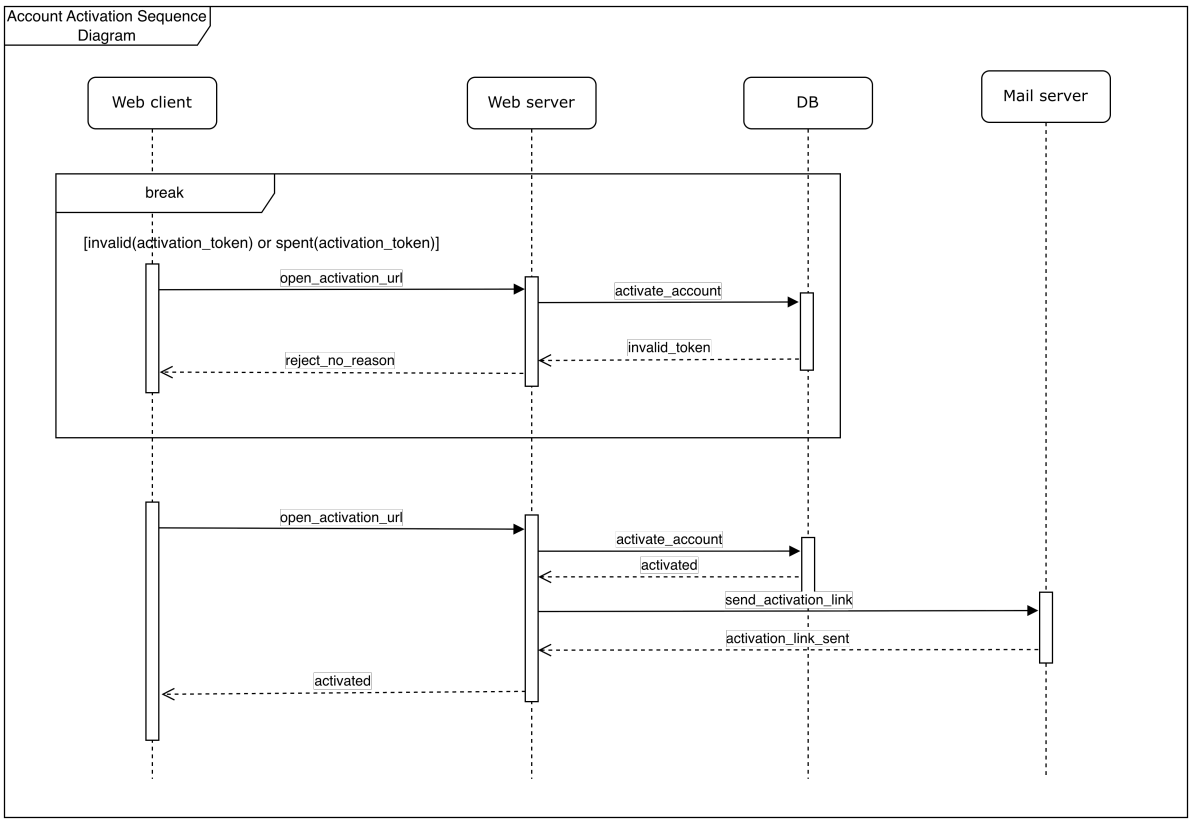


Figure 2: Verification Flow Sequence Diagram

7 Security Assumptions

This section describes the security measures planned for the registration module.

7.1 Password Security

- Passwords hashed using memory-hard algorithm (Argon2).
- Random salt generated per password.
- Passwords never stored in plaintext, logged, or returned to clients.

7.2 Token Security

- Verification tokens generated using cryptographically secure random data.
- Only hashes of tokens stored in database.
- Plaintext tokens sent only via email.
- Tokens have configurable expiry time.
- One active token per user.

7.3 Database Security

- Database encrypted at rest (SQLCipher).
- Encryption key stored securely (system keyring or environment variable).
- Foreign keys with CASCADE delete ensure data consistency.

7.4 Input Validation

- Dual validation on frontend and backend.
- Shared validation logic ensures consistency.
- Backend validation is authoritative.

7.5 Transport Security

- HTTPS required in production for all credential transmission.
- Development environment may use HTTP for convenience.

7.6 Future Enhancements

Potential security enhancements for Registration_3 scope:

- Rate limiting on registration endpoint.
- CAPTCHA integration for bot mitigation.
- Email domain whitelist/blacklist.
- Invite token system for controlled registration.