Application Security – Laboratories – LAB03

User registration process (Part A)

Gaillard Théo
`theo.gaillard@student.put.poznan.pl`
Quivron Emile
`emile.quivron@student.put.poznan.pl`

Supervisor: Michał Apolinarski, Ph.D.
Politechnika Poznańska

November 23, 2025

Component Short Description

The user registration module is a secure software component responsible for creating new user accounts within the system. Its primary purpose is to allow new users to provide personal and authentication information, validate that information securely on the server side, and store the finalized account data in the application database.

Collected data

The component collects the following data from the user during registration:

- Email address – used as the unique identifier for the account and for sending activation tokens.
- Password – chosen by the user and processed using strong hashing before storage.
- Optional user metadata – such as username or full name, depending on system requirements.

Security assumptions and objectives

- The server is assumed to operate over a secure HTTPS channel to ensure confidentiality during transmission.
- All data submitted by users is considered untrusted and must be validated server-side to prevent injection attacks, malformed input, or exploitation attempts.
- Passwords must never be stored in plain text; they are protected using a secure one-way hashing function with salting.
- A unique activation token is generated for each new registration to confirm ownership of the provided email address.
- The module supports error handling and defensive measures throughout the process, including input validation, token verification, storage integrity constraints, and optional security hardening (rate limiting, advanced password policies, and event logging).

Component Requirements

Functional Requirements

FR-01 The system must provide a user registration form for entering required data.

FR-02 The server must validate all submitted fields for correctness, integrity, and constraints.

FR-03 The system must verify that the email address is not already registered.

FR-04 The system must hash passwords securely before storing them.

FR-05 The system must generate a unique account activation token after registration.

FR-06 The system must send an activation link containing the token to the user's email address.

FR-07 The system must activate the account by validating the token upon request.

FR-08 The system must handle invalid or expired tokens with clear error messages.

FR-09 The module must store user data (email, hashed password, token status, timestamps).

FR-10 The system should log validation or activation errors.

Optional Functional Requirements (Bonus Features)

OFR-01 Display a password strength meter during user input.

OFR-02 Enforce an advanced password policy (length, complexity, blacklist, etc.).

OFR-03 Perform email verification through DNS MX lookup or SMTP handshake.

OFR-04 Apply email domain restrictions via whitelist or blacklist.

OFR-05 Support registration using invitation tokens.

OFR-06 Implement CAPTCHA or rate limiting to prevent automated or abusive submissions.

OFR-07 Improve activation token security (short lifetime, association with device/IP, etc.).

OFR-08 Log successful and failed registration attempts as security events.

Non-Functional Requirements

Security Requirements

NFR-S01 Passwords must be stored using secure, modern cryptographic hashingitemize.

NFR-S02 Sensitive information (passwords, tokens) must not be logged.

NFR-S03 All communication between client and server should use HTTPS.

Usability Requirements

NFR-U01 Error and validation messages from the frontend must be clear and user-friendly.

NFR-U02 Registration should require no more than one user interaction step before activation.

Performance Requirements

NFR-P01 : Registration requests should be processed with low latency.

NFR-P02 : The system should scale to the required number of concurrent registrations.

Reliability Requirements

NFR-R01 : Duplicate account creation with the same email must be prevented.

NFR-R02 : Activation tokens must have a configurable expiration period.

NFR-R03 : Token validation must fail safely in cases of expiration or tampering.

Maintainability Requirements

NFR-M01 : The module must be structured with separate layers for validation, logic, and data access.

NFR-M02 : Configuration values must be stored in an accessible JSON configuration file.

Tech Stack

Frontend

- HTML5 and CSS3: For building the registration and activation pages.
- JavaScript: Handles client-side input validation (email format, password strength meter, etc.).
- Optional: Bootstrap or TailwindCSS for quick UI styling and responsive layouts.

Backend

- Python 3: Core application logic and registration workflows run here.
- Flask: Web framework for routing (e.g., `POST /register` and `GET /activate/<token>`), input validation, and returning HTML or JSON responses.

Security & Cryptography

- `bcrypt`: For secure password hashing with salt.
- `secrets` (Python standard library): For generating secure random activation tokens.

Database

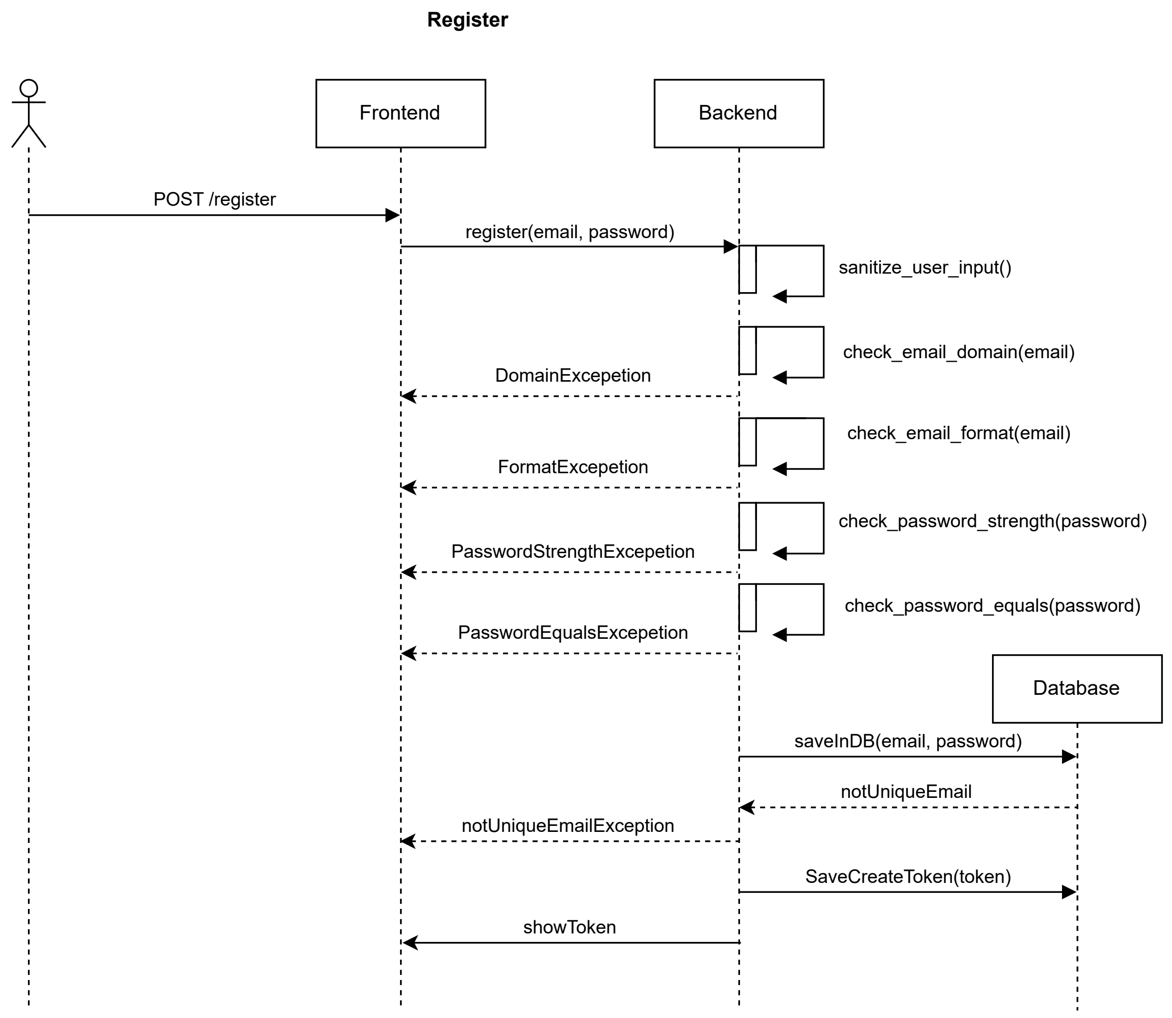
- SQLite3: local file-based relational database accessed using Python's built-in `sqlite3` module.
- Example tables and columns:
  - `users`: `id`, `email`, `password_hash`, `created_at`, `activated` (boolean)
  - `activation_tokens`: `token`, `user_id`, `expires_at`
- Constraints should ensure unique emails, token expiration rules, and referential integrity.

Web Server

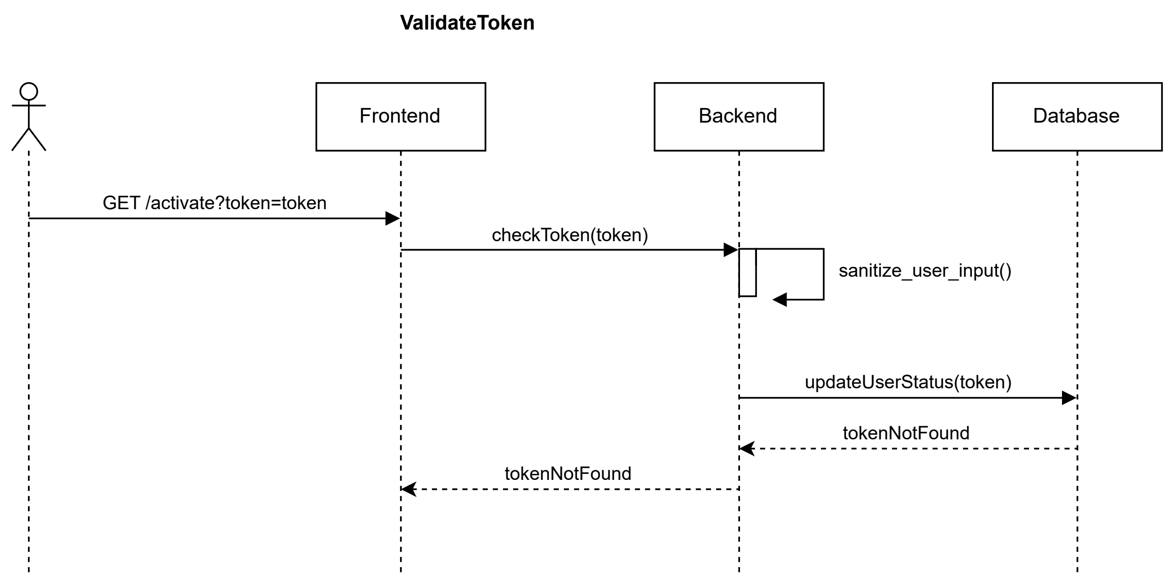
- `nginx` as reverse proxy to the Python application (via Gunicorn); serves static files and handles TLS termination.

UML Diagram

User Registration Process


Register
Frontend
Backend
POST /register
register(email, password)
sanitize_user_input()
check_email_domain(email)
DomainExcepetion
check_email_format(email)
FormatExcepetion
check_password_strength(password)
PasswordStrengthExcepetion
check_password_equals(password)
PasswordEqualsExcepetion
Database
saveInDB(email, password)
notUniqueEmail
notUniqueEmailException
SaveCreateToken(token)
showToken

Token Validation Process


ValidateToken
Frontend
Backend
Database
GET /activate?token=token
checkToken(token)
sanitize_user_input()
updateUserStatus(token)
tokenNotFound
tokenNotFound