Poznan University of Technology
Faculty of Computing and Telecommunications

Course: Application Security - Laboratories

PROJECT DESIGN DRAFT

Secure User Registration Module

Bank Dispute Resolution Platform

Student:
Mario Mastrulli
Student ID: ER-2050

1.Component Description

The purpose of this project is to design and implement a secure user registration module for a **Bank Dispute Resolution Platform**.
This platform is designed for a sensitive environment where individuals (specifically those flagged in banking blacklists or credit bureaus) must register to upload confidential documentation for legal disputes and financial rehabilitation.

Data Collection:
Due to the financial and legal nature of the domain, the system requires strict identity verification (KYC - Know Your Customer). The registration process collects:

- **Full Name:** Required to match the legal identity with the submitted documents.
- **Fiscal Code (Tax ID):** A critical unique identifier required to verify if the user is listed in the target blacklist database.
- **Email Address:** Used as the username and for official communications.
- **Phone Number:** Collected to enable future Two-Factor Authentication (2FA) mechanisms.
- **Password:** User credentials for access.

Security Assumptions:

- **HTTPS Enforcement:** Following the assignment guidelines (Note 4), this design operates on the assumption that the production environment will enforce HTTPS (TLS 1.2+) to encrypt the Fiscal Code and credentials in transit.
- **Environment:** The application database is assumed to be isolated from public internet access.

2. Component Requirements

Functional Requirements:

- **Registration Form:** A secure web interface accepting Full Name, Email Address, Phone Number, Password, and Fiscal Code.
- **Server-Side Input Validation:**
    - **Full Name:** Validation to ensure it contains only alphabetical characters and spaces.
    - **Email:** Syntax validation (RFC 5322).
    - **Phone Number:** Format validation (e.g., numeric, optional country code).
    - **Fiscal Code:** Regex validation to prevent injection and ensure format correctness.
    - **Password:** Complexity enforcement (minimum 8 characters, requiring mixed **case** and **numbers**).
- **Uniqueness Check:** The system must ensure that the Email, Phone Number, and Fiscal Code are not already registered.
- **Token Generation:** Generation of a unique, cryptographically secure activation token.
- **Activation Logic:** The account is created with a locked status **`(is_active = False)`.** It can only be unlocked via a verification link.

Non-Functional Requirements:

- **Confidentiality:** Passwords must never be stored in cleartext. The system will use **PBKDF2-SHA256** (standard in Flask/Werkzeug) or **Argon2** for hashing.
- **Integrity:** Activation tokens must be generated using a CSPRNG (Cryptographically Secure Pseudo-Random Number Generator).
- **Error Handling:** Generic error messages will be used during registration to prevent "User Enumeration" attacks.

3. Component Architecture

To ensure security and rapid prototyping, the following technology stack has been selected:

- **Backend:** Python 3 with **Flask Framework**.
  - *Reason:* Flask provides explicit control over request handling and includes the `werkzeug.security` library for robust password hashing.
- **Database: SQLite**.
  - *Reason:* Lightweight and serverless, ideal for the laboratory prototype while remaining SQL-compatible for future migration to PostgreSQL.
- **Frontend:** HTML5 + CSS (Bootstrap) for a responsive and clean user interface.

High-Level Architecture:
This diagram illustrates the flow of data from the client to the database.
`[Client Browser] <--(HTTPS)--> [Flask App (Logic/Validation)] <--(SQL)--> [SQLite DB]`

4. Database Structure

The database consists of a primary table users. The schema is designed to enforce data integrity and security, reflecting the data collected during registration.

Field Name	**Data Type**	**Constraints**	**Description**
id	INTEGER	PK, Auto-increment	Internal unique identifier.
full_name	VARCHAR(100)	NOT NULL	User's legal First and Last Name.
email	VARCHAR(255)	UNIQUE, NOT NULL	User's login address.
phone_number	VARCHAR(20)	UNIQUE, NOT NULL	Contact for security alerts/OTP.
password_hash	VARCHAR(255)	NOT NULL	Hashed password string.
fiscal_code	VARCHAR(16)	UNIQUE, NOT NULL	**Sensitive:** Used for blacklist checks.
activation_token	VARCHAR(64)	UNIQUE, NULLABLE	Random string for activation.
is_active	BOOLEAN	DEFAULT 0 (False)	Account status flag.
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Creation timestamp for auditing.

5. UML Sequence Diagrams

The following diagrams illustrate the sequence of operations for the registration and activation flows.

Figure 1: Registration Flow

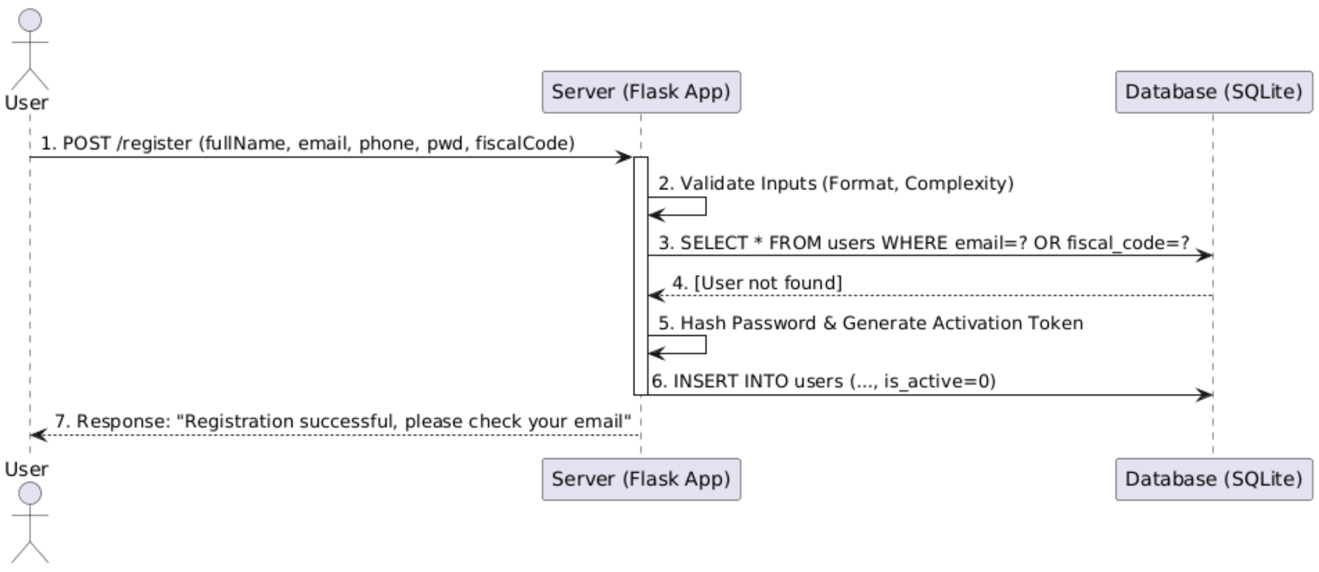
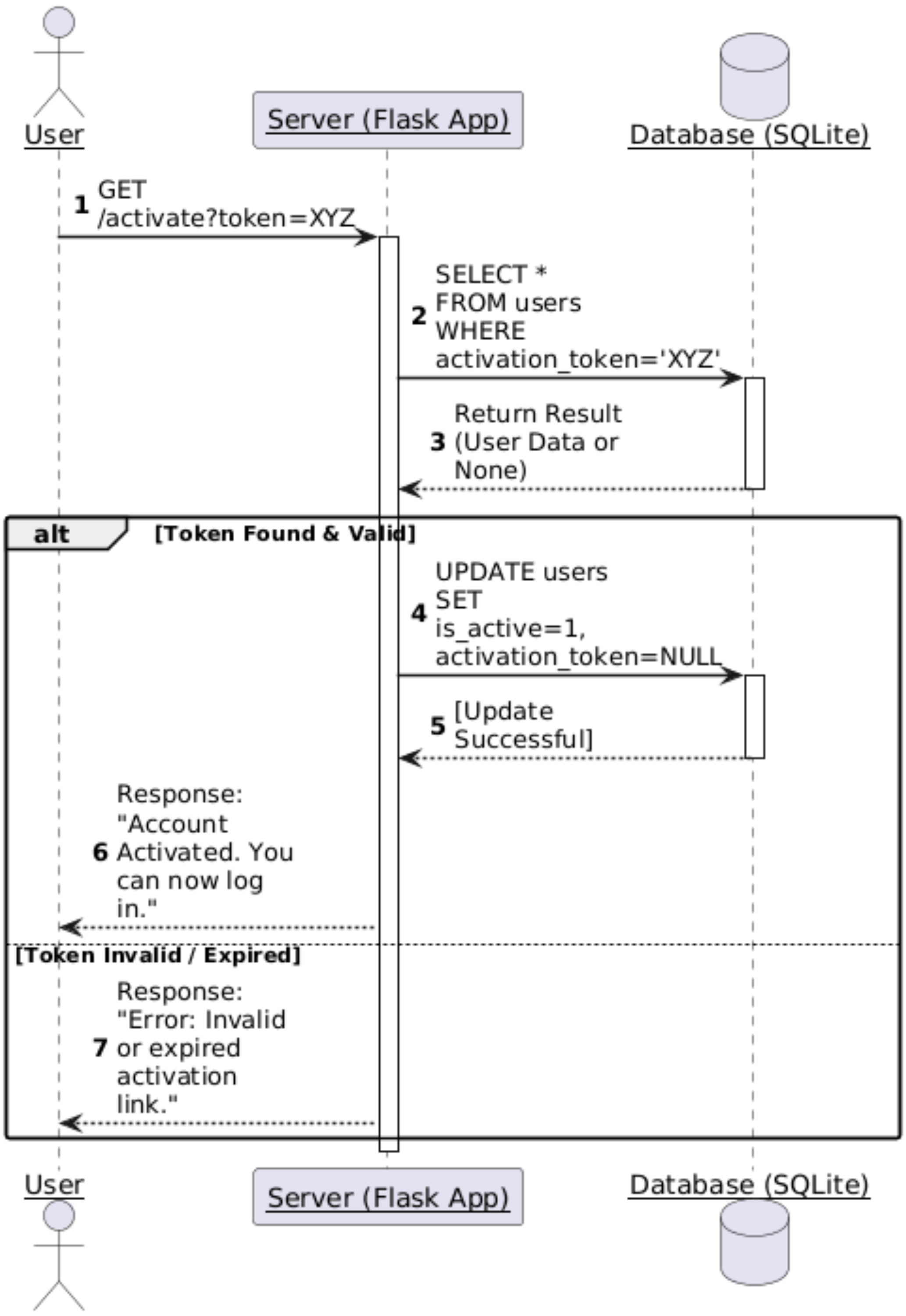

User
Server (Flask App)
Database (SQLite)
1. POST /register (fullName, email, phone, pwd, fiscalCode)
2. Validate Inputs (Format, Complexity)
3. SELECT * FROM users WHERE email=? OR fiscal_code=?
4. [User not found]
5. Hash Password & Generate Activation Token
6. INSERT INTO users (..., is_active=0)
7. Response: "Registration successful, please check your email"
User
Server (Flask App)
Database (SQLite)

Figure 1

Description of Figure 1:

1. The User submits the registration form (Full Name, Email, Phone, Password, Fiscal Code).
2. The Server validates the input format and complexity.
3. The Server checks the Database for existing Email or Fiscal Code to prevent duplicates.
4. If the user is unique, the Server hashes the password and generates a secure activation token.
5. The Server saves the new user record with `is_active=0` (inactive).
6. The Server displays a success message to the user.

Figure 2: Activation Flow



User
Server (Flask App)
Database (SQLite)
1 GET /activate?token=XYZ
2 SELECT * FROM users WHERE activation_token='XYZ'
3 Return Result (User Data or None)
alt
[Token Found & Valid]
4 UPDATE users SET is_active=1, activation_token=NULL
5 [Update Successful]
6 Response: "Account Activated. You can now log in."
[Token Invalid / Expired]
7 Response: "Error: Invalid or expired activation link."
User
Server (Flask App)
Database (SQLite)

Description of Figure 2:
This diagram illustrates the account activation process, including the handling of both successful and unsuccessful scenarios using a UML `alt` (alternative) fragment.

1. The User initiates the process by accessing the activation URL with the provided Token.
2. The Server queries the Database to validate the Token.
3. **The `alt` block shows the conditional logic:**
    - **If the Token is found and valid**, the Server updates the user's record, setting `is_active=1` and nullifying the token (`activation_token=NULL`) to prevent reuse. It then confirms the successful activation.
    - **If the Token is invalid or expired**, the Server sends back an appropriate error message to the user.

This flow ensures that only legitimate activation attempts result in an active account.