COMP.SEC.300 Programming Project Report

Mehmet Onur Sahin

May, 2025

Contents

Chapter 1

Use of AI

During the development of this project, AI-based tools — specifically ChatGPT and Claude — were utilized in certain phases to enhance productivity and improve the quality of the deliverables. AI assistance was primarily employed for the following purposes:

- **Idea Generation:** To gather suggestions and conceptual guidance on implementation approaches, secure programming practices, and user interface improvements.
- **Manual Testing Support:** AI tools were consulted to assist in designing and interpreting manual testing scenarios, and identifying possible edge cases.
- **Technical Writing Support:** AI was used to aid in structuring and formalizing the content of this report, ensuring clarity, consistency, and academic tone while preserving the author's original findings and interpretations.

Most of the coding, implementation, and final decision-making responsibilities were carried out by me. AI tools served as supplementary resources.

Chapter 2

General Description

The goal of this project, titled *MatchScores*, is to provide a simple and secure web application that allows users to follow football match scores. The program has been developed using the Python Flask framework and is designed to be operated through a web browser interface hosted locally.

The web application runs on the user's local machine via the URL `http://127.0.0.1:8080`, which serves as the root endpoint of the system. From here, users can navigate to various parts of the application depending on their access level.

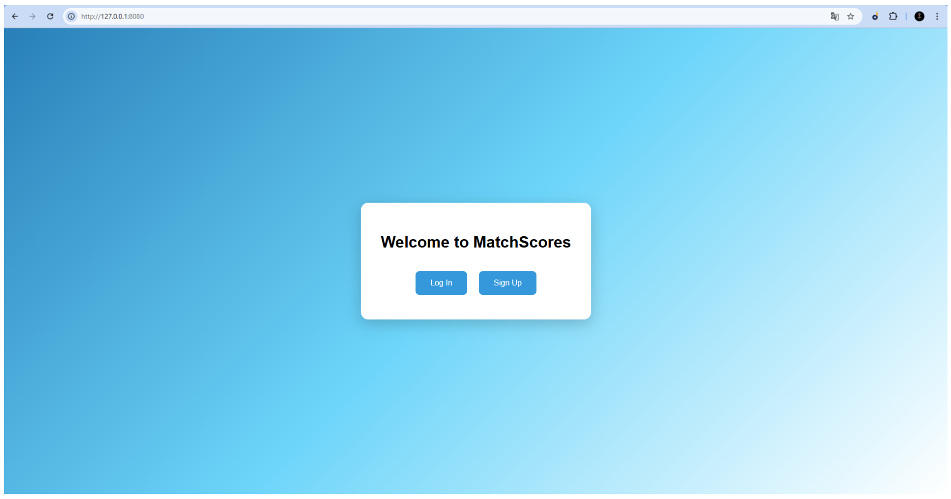


Figure 2.1: Root URL of the program

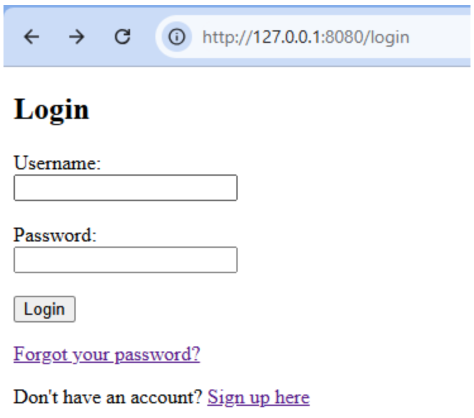
The application supports two types of user roles:

- **User:** Regular users can sign up, log in, and browse football match records. They are able to search for matches and view information about individual games.

- **Admin:** In addition to all user-level functionalities, administrators have access to content management features. They can create new football clubs, add new match records, and delete existing matches from the database. **Since I did not implement an admin panel page, giving admin role to a user is only possible by directly manipulating the database.**

The user interface of the application consists of several essential views:

- A **login page**, where users can authenticate themselves.
- A **signup page**, for new users to create an account.


http://127.0.0.1:8080/login

Login

Username:

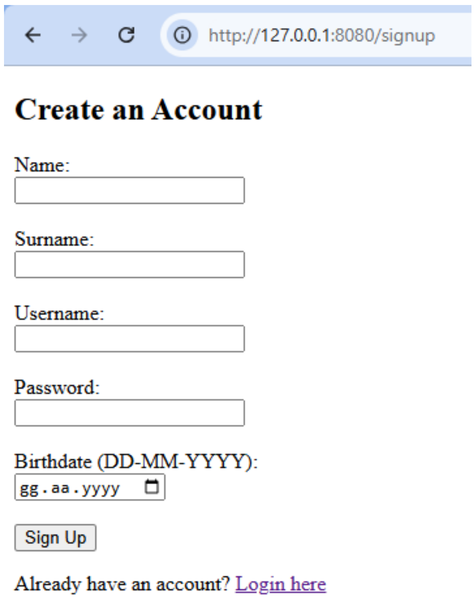
Password:

Login

Forgot your password?

Don't have an account? Sign up here

(a) Login Page


http://127.0.0.1:8080/signup

Create an Account

Name:

Surname:

Username:

Password:

Birthdate (DD-MM-YYYY):
gg.aa.yyyy

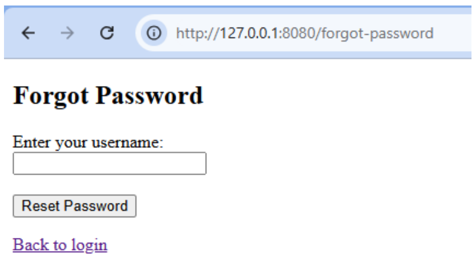
Sign Up

Already have an account? Login here

(b) Signup Page

Figure 2.2: User authentication interfaces: login and signup pages.

- A **forgot password page**, which allows users to request a password reset. This functionality generates a secure token that is sent to the user, enabling them to safely reset their password. **This feature will be further explained.**



http://127.0.0.1:8080/forgot-password

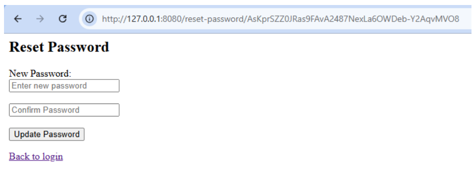
Forgot Password

Enter your username:

Reset Password

Back to login

(a) Forgot Password Page



http://127.0.0.1:8080/reset-password/AsKprSZZ0JRas9FAvA2487NexLa6OWDeb-Y2AqvMVO8

Reset Password

New Password:

Enter new password

Confirm Password

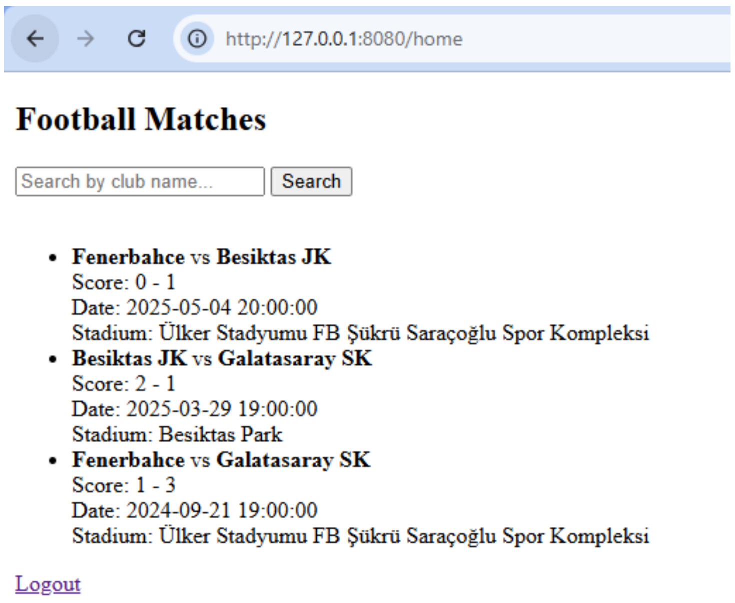
Update Password

Back to login

(b) Reset Password Page

Figure 2.3: Forgot and reset password pages.

- A **main dashboard**, where users can search for football matches and view matches.



http://127.0.0.1:8080/home

Football Matches

Search by club name... Search

- **Fenerbahce** vs **Besiktas JK**
  Score: 0 - 1
  Date: 2025-05-04 20:00:00
  Stadium: Ülker Stadyumu FB Şükrü Saraçoğlu Spor Kompleksi
- **Besiktas JK** vs **Galatasaray SK**
  Score: 2 - 1
  Date: 2025-03-29 19:00:00
  Stadium: Besiktas Park
- **Fenerbahce** vs **Galatasaray SK**
  Score: 1 - 3
  Date: 2024-09-21 19:00:00
  Stadium: Ülker Stadyumu FB Şükrü Saraçoğlu Spor Kompleksi

Logout

Figure 2.4: Homepage for a normal user

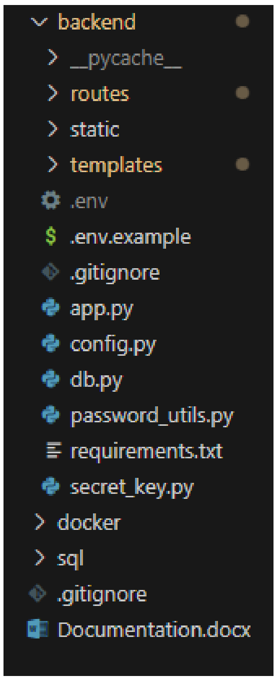
The application's design prioritizes clarity and ease of use. Most navigation elements are self-explanatory, with additional descriptions provided for actions that may not be immediately evident, such as the password reset token mechanism.

Chapter 3

Structure of the Program

The *MatchScores* project is implemented as a web application using the Python Flask framework. The program follows a modular and organized structure that separates different concerns such as routing, database management, templates, and static resources. This approach improves code readability, maintainability, and security.

Here is a screenshot of the structure of the program:



```
backend
  > __pycache__
  > routes
  > static
  > templates
    .env
    .env.example
    .gitignore
    app.py
    config.py
    db.py
    password_utils.py
    requirements.txt
    secret_key.py
> docker
> sql
  .gitignore
  Documentation.docx
```

Figure 3.1: Structure of the program

- **backend/app.py:** This is the main application file that initializes the

Flask app, sets up configuration parameters, and defines the routing logic for the application. It also registers different blueprints responsible for separating user and admin functionalities.

- **sql/:** Contains the CREATE commands for the database tables using MySQL.
- **backend/routes/:** Contains separate Python files for organizing the application's routes based on user roles and functionalities:
- **backend/templates/:** Contains HTML template files rendered by Flask.. These templates define the structure and layout of the web pages for different parts of the application, such as:
  - `login.html`
  - `signup.html`
  - `home.html`
  - `forgot_password.html`
- **backend/static/:** Stores static resources (CSS files in case of this project).
- **backend/config.py:** Contains configuration settings for the application, including secret keys, database URIs, and other environment-specific variables.
- **backend/password_utils.py:** Includes the required functions for storing and comparing password inputs and saving them to the database securely.
- **docker/:** Contains Dockerfile for backend to ensure a containerized workspace.

Chapter 4

Secure Programming Solutions

In the development of the *MatchScores* web application, several secure programming principles were applied in accordance with the **OWASP Top 10 (2021)** and **SANS CWE Top 25 (2023)** guidelines. Below is a description of the identified risks, the countermeasures implemented, and the locations within the codebase where these protections are enforced.

4.1 Summary Table

Here is the summary table of what secure programming principles were applied in MatchScores project:

Security Risk (OWASP/SANS)	**Implemented Solution**	**Location in Code**
Identification and Authentication Failures (A7)	Passwords are hashed with a salted hash using `generate_salt()` and `hash_password()` before storage. No plaintext passwords are stored.	`signup.py`, `login.py`
Broken Access Control (A1)	Admin-only actions (like adding clubs) are restricted by checking `session['is_admin']` before processing. Unauthorized users are redirected.	`home.py (add_club)`
Injection (A3) / SQL Injection (CWE-89)	All database operations use parameterized queries (%s placeholders) to safely insert user inputs into SQL statements, preventing SQL injection.	All database `execute()` calls
Cryptographic Failures (A2)	Secure password reset tokens are generated with `secrets.token_urlsafe()` and stored with expiration timestamps to prevent reuse or guessing.	`forgot_password.py`
Improper Input Validation (CWE-20)	User inputs like birthdates, club IDs, and passwords are validated for correctness and constraints (e.g., age check ≥ 18, non-empty fields) before use or storage.	`signup.py`, `home.py`
Cryptographic Failures (A2)	Passwords are never stored in plaintext. Reset tokens are secure and time-limited. Credentials like salts and hashes are properly handled.	`signup.py`, `login.py`, `forgot_password.py`
Inadequate Logging and Monitoring (A9)	Login and signup failures provide generic error messages to avoid information leakage (like whether a username exists).	`login.py`, `forgot_password.py`

Table 4.1: Secure Programming Solutions Implemented

4.1.1 Detailed Implementation

Password Hashing and Salting

Why: To protect against password theft and rainbow table attacks.
How:

- A unique salt is generated using `generate_salt()`.
- The password is hashed using `hash_password(password, salt)`.
- Both hash and salt are stored securely in the database.

Example:

```python
salt = generate_salt()
hashed_password = hash_password(password, salt)
```

Validated during login:

```python
hashed_input_password = hash_password(password,
   stored_salt)
if hashed_input_password != stored_hash:
```

Mitigates **OWASP A7** and **SANS CWE-521**.

Access Control

Why: Prevent unauthorized users from accessing admin features.
How:

- Check `session['is_admin']` before allowing access to restricted routes.
- Deny access or redirect if not authorized.

Example:

```python
if not session.get('is_admin'):
    flash("You do not have permission to add a club.", "
       error")
    return redirect(url_for('home.home'))
```

Mitigates **OWASP A1**.

Parameterized Queries

Why: Prevent SQL Injection attacks.
How:

- Use %s placeholders and a separate tuple of parameters for every `execute()` call.
- Never concatenate user input directly into SQL queries.

Example:

```
cursor.execute("SELECT * FROM users WHERE username = %s",
    (username,))
```

Mitigates **OWASP A3** and **SANS CWE-89**.

Password Reset Tokens

Why: Prevent predictable or reusable reset tokens.
How:

- Use `secrets.token_urlsafe(32)` for strong, URL-safe tokens.
- Set an expiry timestamp.

Example:

```
token = secrets.token_urlsafe(32)
expiry = datetime.utcnow() + timedelta(minutes=30)
```

Mitigates **OWASP A2**.

Input Validation

Why: Avoid malformed, dangerous, or policy-violating inputs.
How:

- Check birthdate format and ensure users are at least 18 years old.
- Ensure required form fields are present and valid before use.

Example:

```python
birth_date = datetime.strptime(birth_date_str, "%Y-%m-%d")
if age < 18:
    flash("You must be at least 18 years old to register.", "error")
```

Mitigates **SANS CWE-20**.

Error Messages and Generic Responses

Why: Prevent attackers from enumerating usernames or inferring system state.

How:

- Use the same response whether or not a username exists in the password reset flow.
- Provide non-specific messages on login failure.

Example:

```python
flash("If an account with that username exists, a reset link has been sent.")
```

Mitigates **OWASP A9**.

Chapter 5

Testing and Results

5.1 Testing Approach

To ensure the security and reliability of the *MatchScores* web application, multiple levels of testing were performed:

- **Manual functional testing** via the browser interface to verify correct functionality of user signup, login, password reset, match searching, and club management features.
- **Manual security testing**, focused on common vulnerabilities according to the OWASP Top 10 checklist.
- **Input validation testing**, including attempts to register with invalid data such as weak passwords, future birthdates, and duplicate usernames.
- **Permission control testing**, ensuring that only admins could access club creation and deletion functionalities.

5.2 Testing Results

The following summarizes the testing outcomes:

Test Case	**Result**
User signup with valid credentials	Successfully created an account, stored password securely with salted hash.

Signup with duplicate username	Correctly prevented and displayed an error message.
Signup with underage birthdate or future date	Properly blocked registration for users under 18 years old.
Signup with weak password	Denied signup for passwords that do not satisfy: at least 8 characters long, contain a number, a letter, and a special character
Login with incorrect password	Denied login and displayed appropriate error message.
Forgot password functionality	Generated a secure token and displayed reset link, with token expiring in 30 minutes.
Attempt SQL injection in search and forms	Parameters correctly sanitized by parameterized queries, no SQL injection possible.

5.3 Issues and Fixes

During testing, the following issues were identified:

- **Reset token exposure:** Password reset tokens were displayed in flash messages rather than being emailed. Due to the scope of this local prototype, this was accepted as a limitation.
- **Input length limitations:** No maximum length was enforced on form fields, which could allow potential denial-of-service attempts via large inputs. This can be done easily in future use.

5.4 Unimplemented Features

Some features were not implemented within the project timeline:

- Email integration for password reset tokens.
- Automated unit tests and CI/CD integration.

5.5 Suggestions for Improvement

Future improvements could include:

- Adding automated unit and integration tests.
- Integrating a DevSecOps pipeline with automated security scanning.
- Deploying the application on a secure server environment.

Chapter 6

References

1. OWASP Foundation. (2021). *OWASP Top Ten Web Application Security Risks 2021.* Retrieved from `https://owasp.org/www-project-top-ten/`

2. MITRE. (2023). *CWE Top 25 Most Dangerous Software Weaknesses.* Retrieved from `https://cwe.mitre.org/top25/archive/2023/2023_cwe_top25.html`

3. Flask Documentation. (2025). *Flask Web Framework Documentation (Version 3.0).* Retrieved from `https://flask.palletsprojects.com/`

4. Python Software Foundation. (2025). *secrets library — Generate secure random numbers for managing secrets.* Retrieved from `https://docs.python.org/3/library/secrets.html`

5. Docker Documentation. (2025). *Docker Overview.* Retrieved from `https://docs.docker.com/get-started/`