Task 3

Password Complexity Checker Web Application

report

Introduction

In the digital age, where online accounts govern everything from banking to social interaction, password security is critical. Despite this, many users rely on weak, easily guessable passwords, making them vulnerable to cyberattacks such as brute-force and credential stuffing. The Password Complexity Checker Web Application is designed to help users create stronger passwords through real-time feedback during input. Unlike traditional systems that reject weak passwords only after submission, this tool instantly informs users whether their password meets key criteria—such as minimum length, presence of uppercase letters, digits, and special characters. Built using Python and Flask with a modern Bootstrap interface, and enhanced by JavaScript for dynamic validation, the application combines usability with robust backend checks. It not only enforces secure practices but also educates users on password construction, making it an effective tool for both learning and implementation in secure web systems.

Summary

The Password Complexity Checker functions as a login interface that evaluates the strength of the user's password input in real time. Upon entering a username and password, the application assesses whether the password meets standard complexity requirements such as:

- A minimum of 8 characters
- At least one uppercase letter
- At least one lowercase letter
- At least one number
- At least one special character

These rules are enforced both on the **client-side** via JavaScript and on the **server-side** using Python regular expressions. This dual-layered validation ensures that even if client-side JavaScript is disabled, the backend will still enforce password standards.

The application includes a simple Flask backend that handles POST requests and uses the flash() method to provide feedback to the user. Upon a valid submission, a success message is displayed;

otherwise, the user is prompted with detailed messages indicating what rules were not met. The frontend uses Bootstrap 5 to create a responsive and clean login interface. JavaScript enhances the UX by checking password strength dynamically and displaying each validation rule with either a ✅ or ❌, giving users instant insight into their password's quality.

The entire codebase is self-contained within a single Python file, using render_template_string() instead of external HTML files, making it easier to deploy and share for demo or educational use. The application is accessible on http://127.0.0.1:5000 and can be launched with a single terminal command.

Overall, this tool demonstrates a practical and interactive way to guide users in creating secure passwords, while also being a great example of full-stack development using Flask, HTML, CSS, and JavaScript.

🔷 Working Principle

The working principle of the Password Complexity Checker revolves around **real-time validation** and **user feedback** integrated into a web-based form.

Client-Side Validation:
At the core of the real-time checking functionality is a JavaScript event listener attached to the password input field. As the user types, the script dynamically evaluates the password against five criteria using regular expressions (regex). These checks determine whether the password:

1. Has at least 8 characters
2. Contains at least one uppercase letter ([A-Z])
3. Contains at least one lowercase letter ([a-z])
4. Includes at least one digit (\d)
5. Includes one special character ([!@#$%^&*(),.?":{}|<>])

Each condition returns visual feedback directly below the input field. If a criterion is met, it is shown with a green checkmark (✅); otherwise, a red cross (❌) is shown. This ensures users are immediately aware of what they need to adjust, improving password quality at the source.

Server-Side Validation:

Once the user submits the form, the input is validated again on the backend using the same criteria via Python regex. This is crucial for maintaining integrity and avoiding bypassing the checks through developer tools or disabled JavaScript.

The backend uses Flask’s request object to fetch the input, checks it with check_password_strength() function, and uses Flask's flash() method to communicate messages back to the user. If the password passes all checks, the user receives a success message.

Bootstrap for Styling:
The interface utilizes Bootstrap 5 CDN for styling. It ensures responsiveness across devices and provides a clean, intuitive look for the login panel. The form is embedded within a Bootstrap card component, centered and padded for better UX. Error and success messages are styled using Bootstrap’s alert and list-group classes, giving professional visual cues.

Security and Usability:

The application doesn’t store any credentials but is built to simulate a secure login system. It emphasizes security education, especially regarding password construction, without overwhelming users. It also prevents weak password submissions by enforcing checks even if JavaScript is tampered with.

This architecture showcases best practices in web development and security: layered validation, user-centric feedback, and clean UX.

CODE:

```
from flask import Flask, request, render_template_string, flash, redirect, url_for


app = Flask(__name__)

app.secret_key = 'supersecretkey'


HTML_TEMPLATE = """

<!DOCTYPE html>
```

```
}

<html>

<head>

   <title>Password Complexity Checker</title>

   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/boo
tstrap.min.css" rel="stylesheet">

   <style>

      body {

         background-color: #f5f7fa;

      }

      .card {

         margin-top: 80px;

         box-shadow: 0 4px 8px rgba(0,0,0,0.1);

      }

      .strength {

         margin-top: 10px;

      }

      .valid {

         color: green;

      }

      .invalid {

         color: red;

      }
```

```
  </style>
</head>
<body>
  <div class="container">
    <div class="row justify-content-center">
      <div class="col-md-6">
        <div class="card p-4">
          <h3 class="text-center mb-4">🔐 Password Complexity
Checker</h3>
          <form method="POST" novalidate>
            <div class="mb-3">
              <label class="form-label">Username:</label>
              <input type="text" name="username" class="form-
control" required>
            </div>
            <div class="mb-3">
              <label class="form-label">Password:</label>
              <input type="password" id="password"
name="password" class="form-control" required>
              <div id="feedback" class="form-text
strength"></div>
            </div>
            <button type="submit" class="btn btn-success w-
100">Login</button>
```

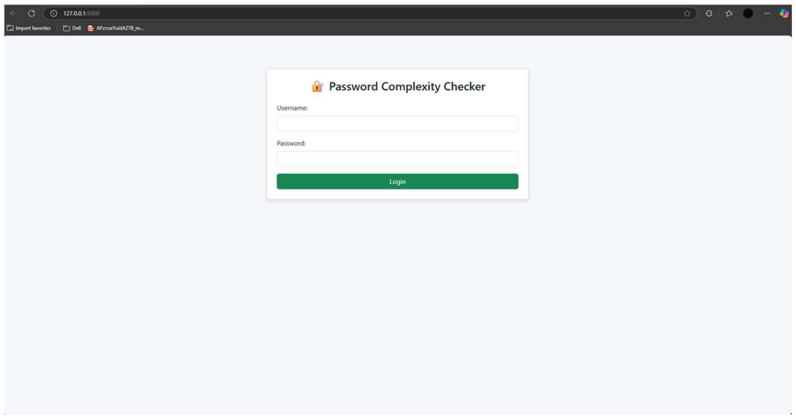
```
                </form>


                {% with messages =
get_flashed_messages(with_categories=true) %}
                  {% if messages %}
                    <ul class="mt-3 list-group">
                      {% for category, message in messages %}
                        <li class="list-group-item list-group-item-{{
'success' if category == 'success' else 'danger' }}">
                          {{ message }}
                        </li>
                      {% endfor %}
                    </ul>
                  {% endif %}
                {% endwith %}
              </div>
            </div>
          </div>
  </div>


<script>
document.getElementById("password").addEventListener("input"
, function() {
```

```
let pwd = this.value;
let feedback = [];

if (pwd.length < 8) {
   feedback.push("❌ Minimum 8 characters");
} else {
   feedback.push("✅ Minimum 8 characters");
}

if (/[A-Z]/.test(pwd)) {
   feedback.push("✅ At least one uppercase letter");
} else {
   feedback.push("❌ At least one uppercase letter");
}

if (/[a-z]/.test(pwd)) {
   feedback.push("✅ At least one lowercase letter");
} else {
   feedback.push("❌ At least one lowercase letter");
}

if (/[0-9]/.test(pwd)) {
    feedback.push("✅ At least one number");
  } else {
    feedback.push("❌ At least one number");
  }

  if (/[!@#$%^&*(),.?\":{}|<>]/.test(pwd)) {
    feedback.push("✅ At least one special character");
  } else {
    feedback.push("❌ At least one special character");
  }

  document.getElementById("feedback").innerHTML =
feedback.join("<br>");
});
</script>
</body>
</html>
"""

@app.route("/", methods=["GET", "POST"])
def login():
  if request.method == "POST":
    username = request.form.get("username")
```

```
        password = request.form.get("password")

        errors = check_password_strength(password)

        if errors:
            for error in errors:
                flash(error, "error")
        else:
            flash(f"Welcome, {username}! ✅ Password is strong.", 
"success")
            return redirect(url_for("login"))

    return render_template_string(HTML_TEMPLATE)

def check_password_strength(password):
    import re
    errors = []

    if len(password) < 8:
        errors.append("Password must be at least 8 characters long.")
    if not re.search(r"[A-Z]", password):
        errors.append("Password must contain at least one 
uppercase letter.")
```

```
    if not re.search(r"[a-z]", password):
        errors.append("Password must contain at least one
lowercase letter.")
    if not re.search(r"\d", password):
        errors.append("Password must contain at least one digit.")
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        errors.append("Password must contain at least one special
character.")

    return errors

if __name__ == "__main__":
    app.run(debug=True)
```

OUTPUT

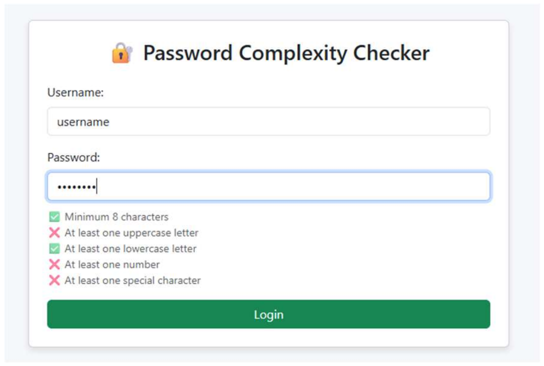


Password Complexity Checker

Username:

Password:

Webpage



Password Complexity Checker

Username:

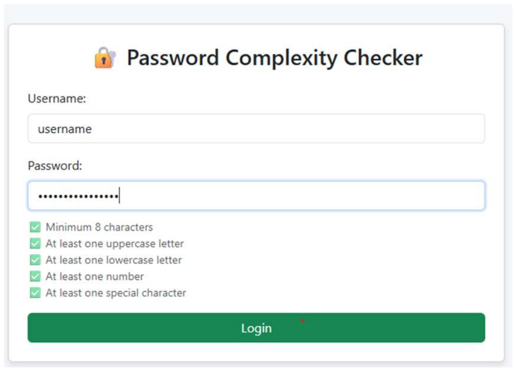
username

Password:

••••••••

✓ Minimum 8 characters
✗ At least one uppercase letter
✓ At least one lowercase letter
✗ At least one number
✗ At least one special character

Output 1



Password Complexity Checker

Username:

username

Password:

••••••••••••••••

✓ Minimum 8 characters
✓ At least one uppercase letter
✓ At least one lowercase letter
✓ At least one number
✓ At least one special character

Output 2

Conclusion

The Password Complexity Checker Web Application offers a practical solution to enhance password security through real-time feedback and backend validation. It educates users by visually showing password strength as they type, encouraging stronger choices. Built with Flask, Bootstrap, and JavaScript, it balances usability with robust security practices. Though it currently lacks credential storage, it serves as a solid foundation for future enhancements like password hashing and multi-factor authentication. This tool not only helps users understand secure password practices but also provides developers with a template for building secure, interactive web forms that prioritize both functionality and user experience.