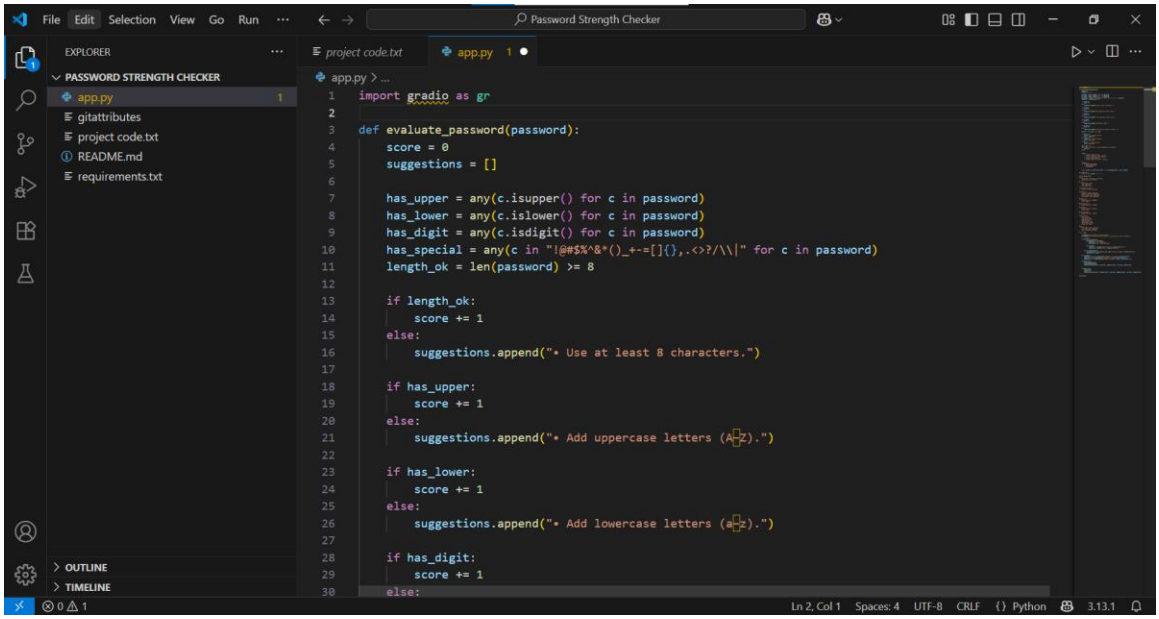
📄 Password Strength Checker – Project Documentation

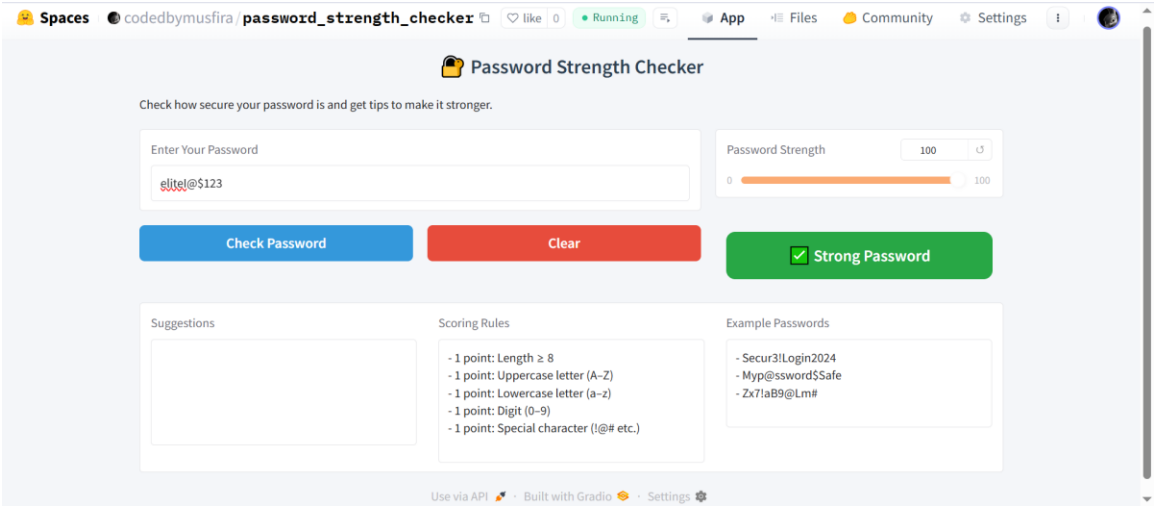
Date: July 26, 2025
Author: Musfira Ahmed
Project: Web App Development using Python + Gradio

---

1. Task Overview

The aim of this project was to build a Password Strength Checker Web App using Python and Gradio. The app evaluates passwords based on five security rules and gives feedback via a score, strength verdict, and improvement suggestions. The UI is styled using embedded CSS for a better user experience.





2. Password Evaluation Logic

2.1 evaluate_password() Function

This function performs the core analysis by applying five checks:

```
def evaluate_password(password):
    score = 0
    suggestions = []

    has_upper = any(c.isupper() for c in password)
    has_lower = any(c.islower() for c in password)
    has_digit = any(c.isdigit() for c in password)
    has_special = any(c in "!@#$%^&*()_+-=[]{},.<>?/\\|" for c in
password)
```

```
length_ok = len(password) >= 8
```

Each valid criterion adds 1 point to the score:

```
if length_ok:
    score += 1
else:
    suggestions.append("◆ Use at least 8 characters.")

if has_upper:
    score += 1
else:
    suggestions.append("◆ Add uppercase letters (A–Z).")

if has_lower:
    score += 1
else:
    suggestions.append("◆ Add lowercase letters (a–z).")

if has_digit:
    score += 1
```

```
else:
    suggestions.append("🔶 Add digits (0–9).")

if has_special:
    score += 1
else:
    suggestions.append("🔶 Add special characters (!@# etc.).")
```

Password strength is calculated:

```
percent = int((score / 5) * 100)

if score == 5:
    verdict = "💪 Strong Password"
    color = "#28a745"
elif score >= 3:
    verdict = "😐 Medium Password"
    color = "#ffc107"
else:
    verdict = "⚠️ Weak Password"
    color = "#dc3545"
```

A styled HTML verdict box is returned with extra info:

```
result_html = f"""
<div class='verdict-box' style='background-color:{color};'>
    {verdict}
</div>
"""


rules = (
    "- 1 point: Length ≥ 8\n"
    "- 1 point: Uppercase letter (A–Z)\n"
    "- 1 point: Lowercase letter (a–z)\n"
    "- 1 point: Digit (0–9)\n"
    "- 1 point: Special character (!@# etc.)"
)

examples = (
    "- Secur3!Login2024\n"
    "- Myp@ssword$Safe\n"
    "- Zx7!aB9@Lm#"
)
```

```
    return percent, gr.HTML(result_html), "\n".join(suggestions), rules, examples
```

3. Clear Functionality

This allows the user to input a new password without refreshing the app.

The clear_all() function resets all fields:

```
def clear_all():
    return "", 0, gr.HTML(""), "", "", ""
```

4. Gradio Interface Setup

4.1 Layout and Styling with Gradio Blocks

```
with gr.Blocks(css="""
body {
    font-family: 'Segoe UI', sans-serif;
    background-color: #f4f6f8;
}
...
.verdict-box {
```

```
    color: white;
    font-size: 18px;
    font-weight: bold;
    padding: 14px;
    text-align: center;
    border-radius: 10px;
    margin-top: 12px;
}
""") as app:
```

This CSS improves the visual design of the app, including button hover effects and a better font and layout.

4.2 Building the User Interface

```
    gr.Markdown("## 🔐 Password Strength Checker")
    gr.Markdown("Enter your password below to get a full strength check, improvement tips, and example strong passwords.")

    with gr.Group():
        password_input = gr.Textbox(
```

```
        label="🔑 Enter Your Password",
        type="text",  # Optional: change to "password" for hidden input
        placeholder="e.g., MySecureP@ss123"
    )
```

4.3 Buttons and Output Fields

```
with gr.Row():
        check_btn = gr.Button("✅ Check Password")
        clear_btn = gr.Button("❌ Clear")


    strength_slider = gr.Slider(0, 100, label="Password Strength (%)", interactive=False)
    result_box = gr.HTML()
    suggestions_box = gr.Textbox(label="🔧 Suggestions to Improve", lines=4, interactive=False)
    rules_box = gr.Textbox(label="📋 Scoring Rules", lines=5, interactive=False)
    examples_box = gr.Textbox(label="💡 Example Strong Passwords", lines=3, interactive=False)
```

4.4 Button Events & Function Mapping

```
    check_btn.click(
        fn=evaluate_password,
        inputs=password_input,
        outputs=[strength_slider, result_box, suggestions_box,
rules_box, examples_box]
    )
    clear_btn.click(
        fn=clear_all,
        inputs=None,
        outputs=[password_input, strength_slider, result_box,
suggestions_box, rules_box, examples_box]
    )
app.launch()
```

This connects the button clicks to the core functions, enabling full interaction.

5. Challenges & Solutions

5.1 Plain UI

- **Issue:** Basic UI lacked styling.

- **Solution:** Custom CSS was embedded to style input fields, buttons, verdict boxes, and layout.

5.2 No Reset Option

- **Issue:** App lacked a reset mechanism.
- **Solution:** clear_all() was implemented to reset all fields.

5.3 Visible Password Input

- **Issue:** Password input was in plain text.
- **Solution:** For production, change type="text" to type="password" to mask input.

6. What I Did

- Designed a scoring algorithm for password security.
- Created a Gradio interface with input/output components.
- Implemented real-time suggestions for missing password criteria.
- Styled the app using custom CSS.
- Added reset functionality and HTML verdict display.
- Included strong password examples and rule breakdown.

7. Learning Outcomes

- Developed an end-to-end web app using **Gradio** and **Python**.

- Learned to integrate **HTML/CSS** into Python-based UI.
- Practiced real-world **input validation logic**.
- Built UI logic using **Gradio components** and layout tools.
- Understood deployment potential using **Hugging Face Spaces**.

8. Requirements & Installation

Requirements (requirements.txt):

python>=3.8

gradio>=4.0

Install Dependencies

pip install gradio

9. Optional Deployment Platforms

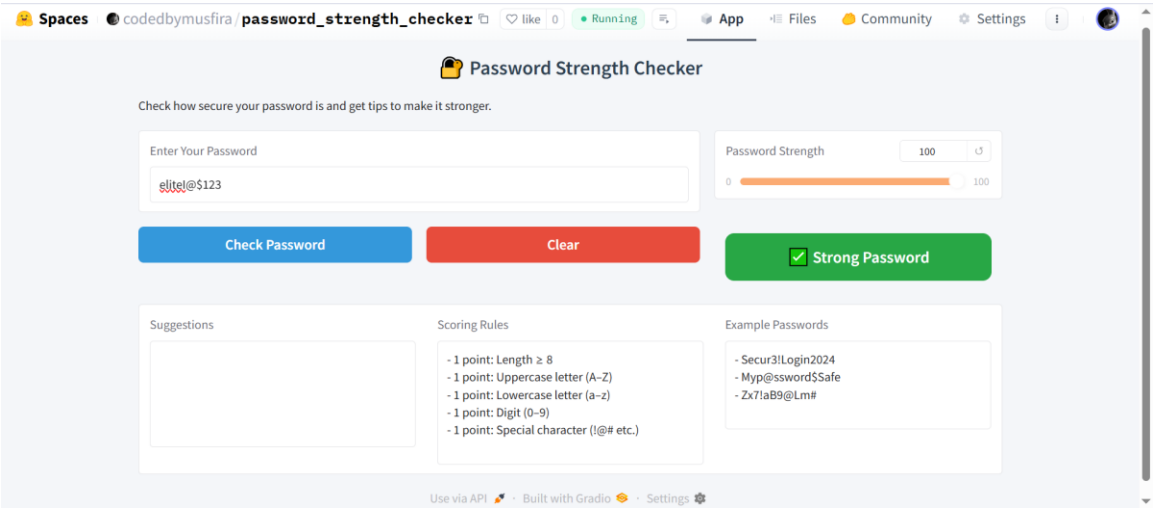
You can deploy this app to any of the following platforms:

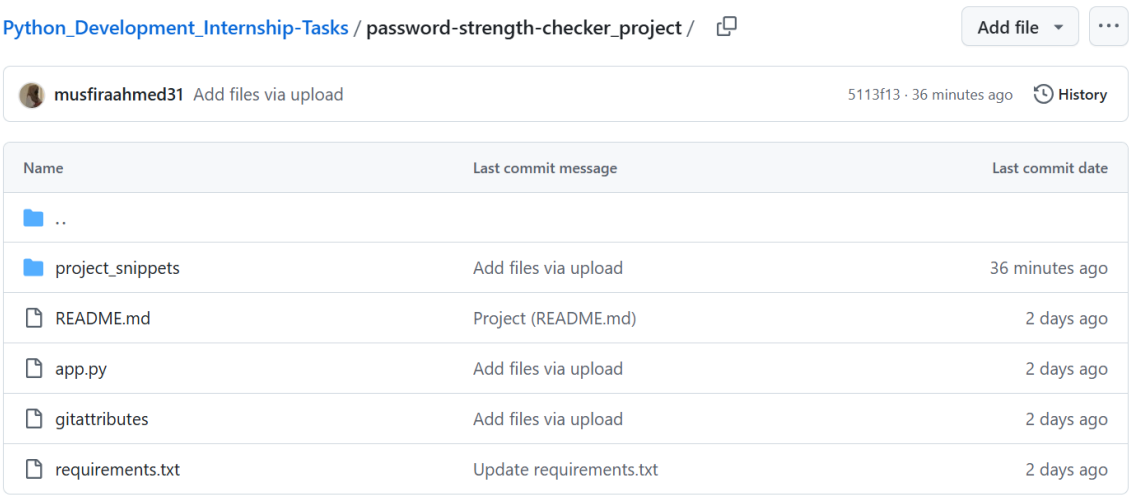
- Hugging Face Spaces
- Streamlit Community Cloud
- Render

To run the app locally:

python app.py

10. Screenshots of Code Running





- App Launched in Browser
- Strong Password Test Result

- Weak Password Test Result
- Suggestions Displayed