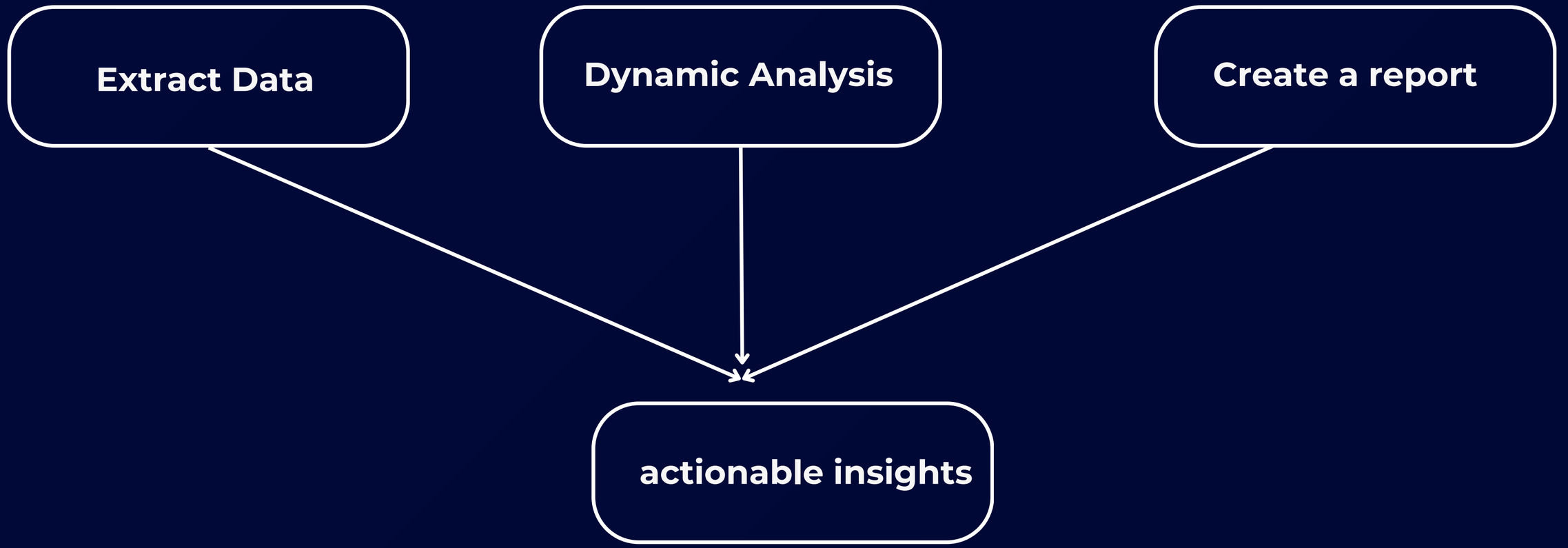
WEBSITE QUALITY ASSURANCE FEEDBACK

AICHA ETTRIKI

Introduction

The Webpage **Hybrid Testing Application** is a tool designed to :



Objectives

Identify Weaknesses

Quickly pinpoint areas that need improvement.

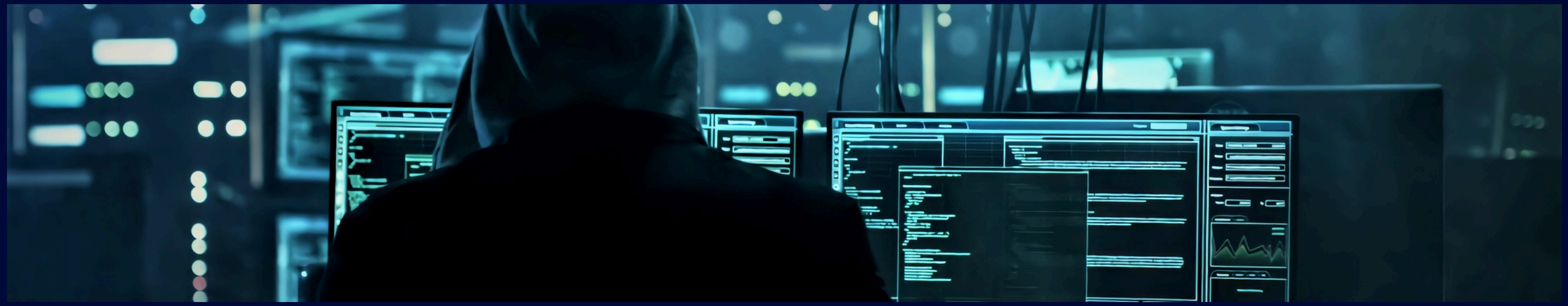
Enhance Web Quality

Provide recommendations for better user experience and performance.

Save Time

Automate manual testing processes for quicker iterations.

Features



Performance

Accessibility

Compatibility

Usability

Security

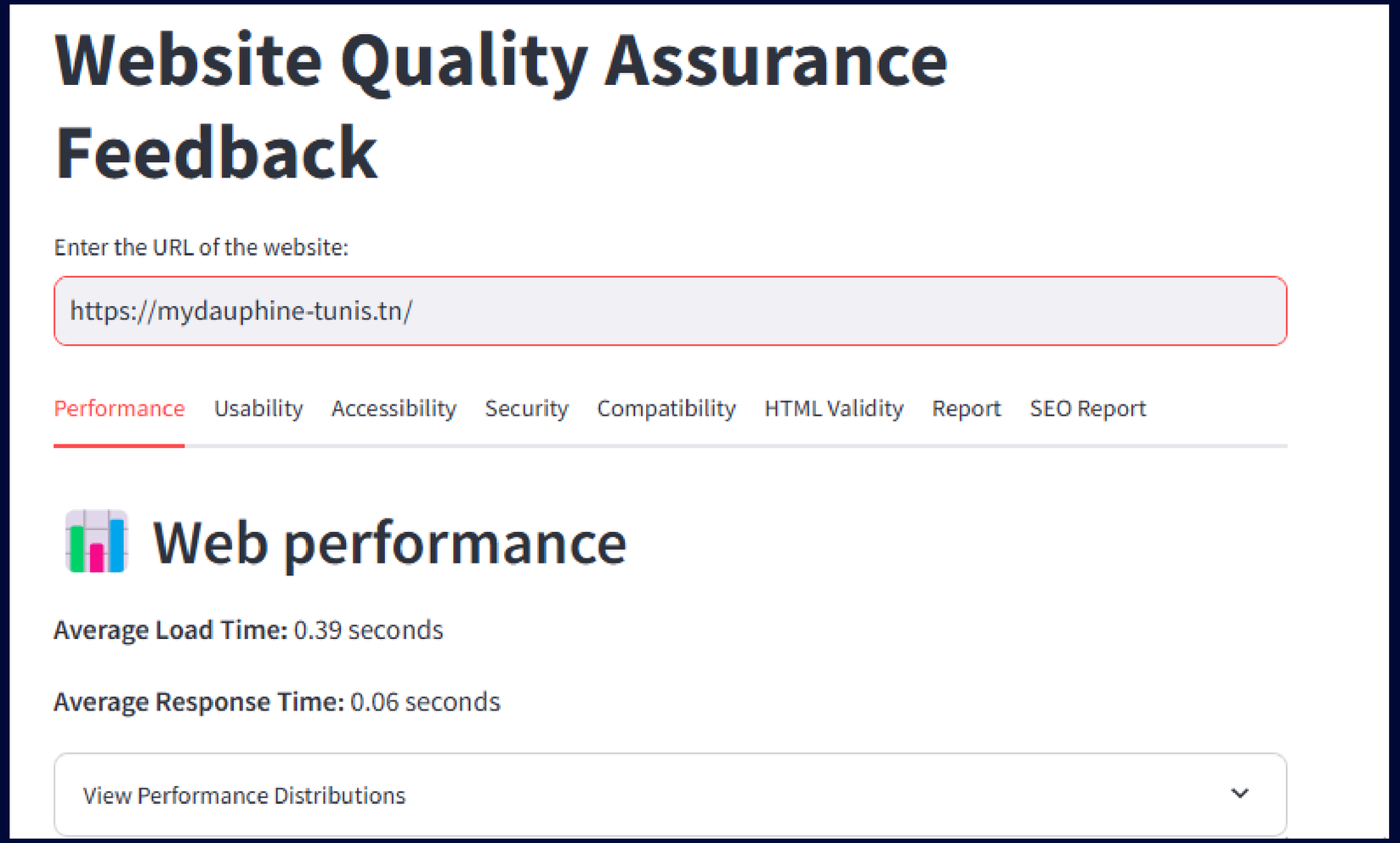
HTML Validity

Results Report

SEO Report

How it works

Performance



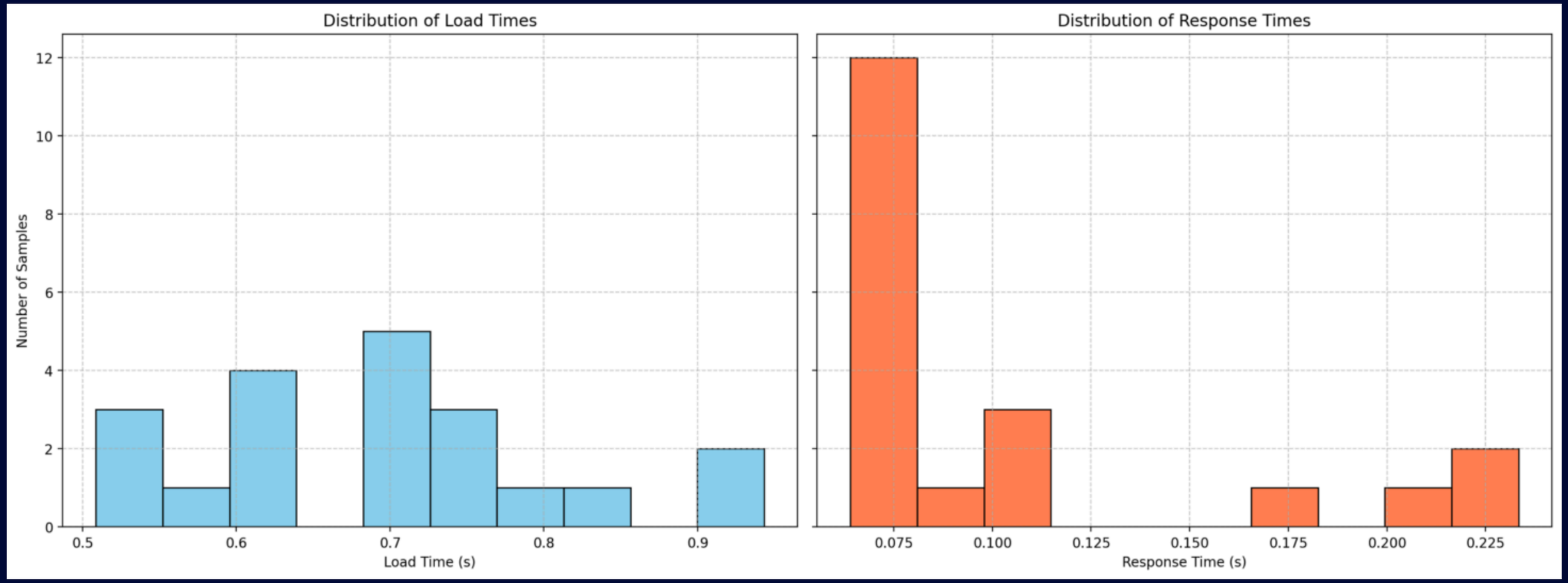
How it works

Performance

```
# Measure the performance 'num_samples' times
for _ in range(num_samples):
    start = timeit.default_timer()  # Start time
    response = requests.get(url)  # Perform GET request
    end = timeit.default_timer()  # End time
    load_times.append(end - start)  # Total time for request
    response_times.append(response.elapsed.total_seconds())  # Time from server

# Calculate averages
avg_load_time = sum(load_times) / num_samples
avg_response_time = sum(response_times) / num_samples
```

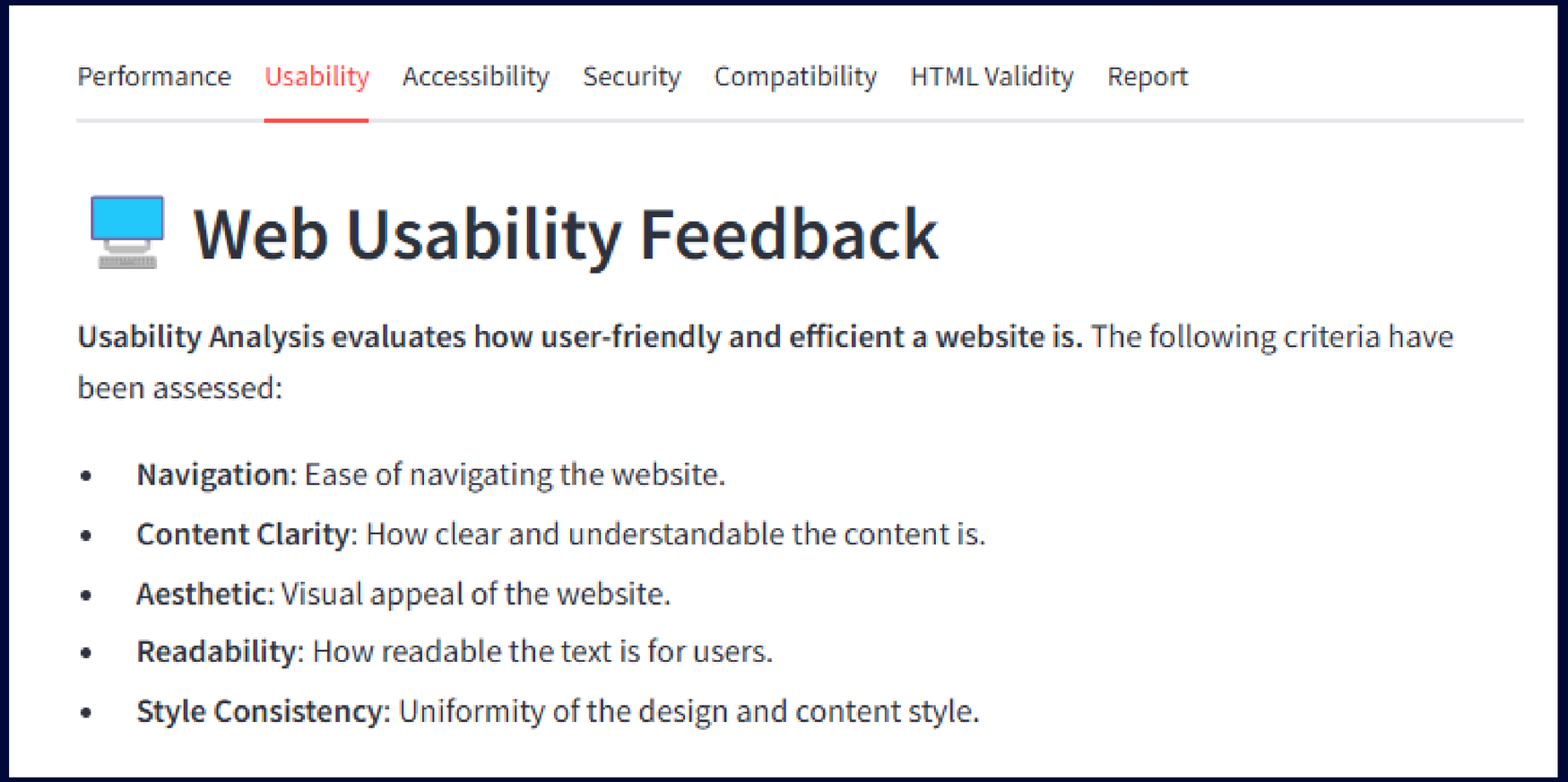
How it works

Performance



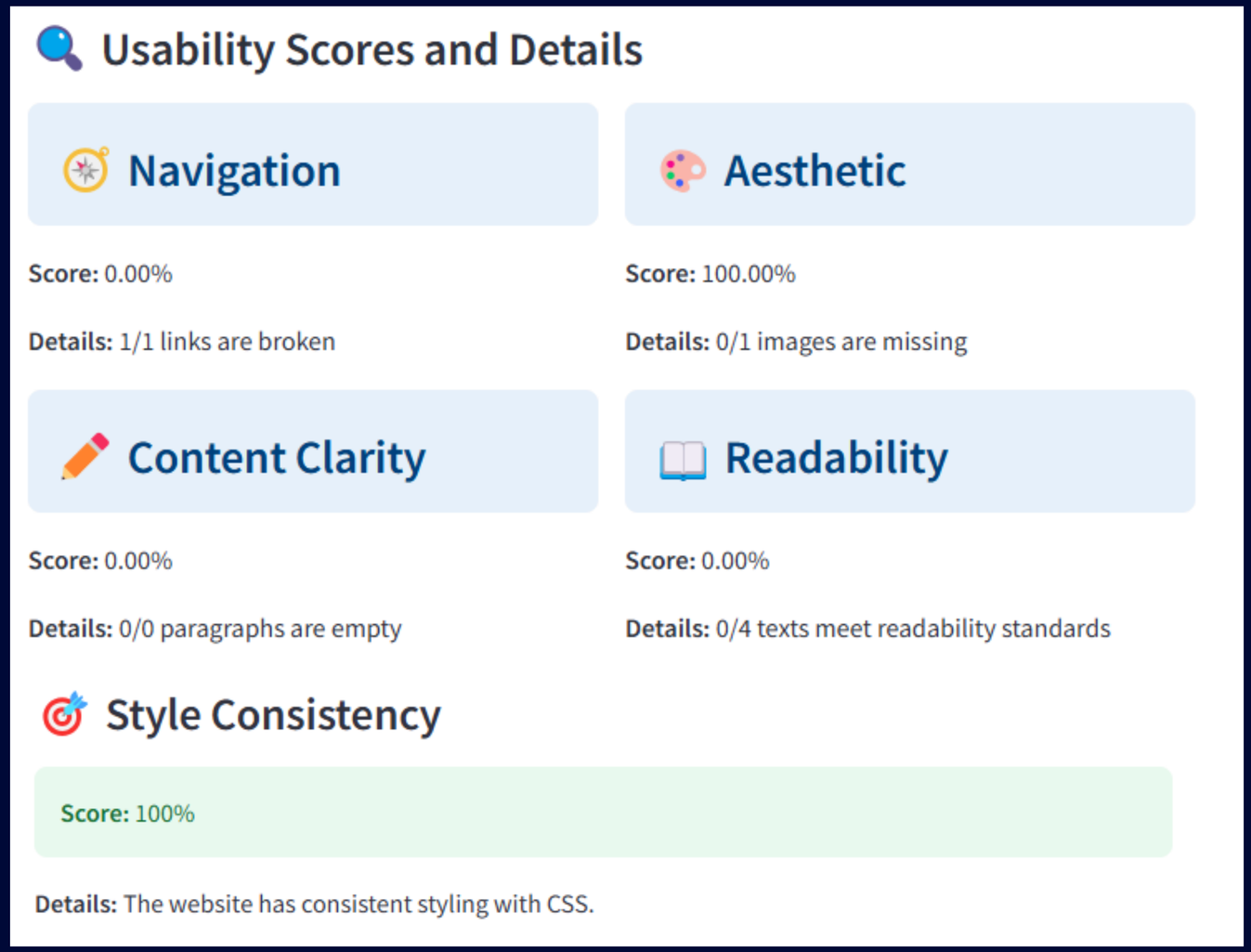
How it works

Usability



How it works

Usability



How it works

Usability

Broken links

Evaluates the accessibility of links on the site.

Content Clarity

Measures the clarity of textual content.

Aesthetic

Assesses the presence and quality of images.

Readability

Evaluates text readability in terms of font size and color contrast.

Style Consistency

Checks for consistent styling across the site using CSS.

How it works

Usability

Broken links

```
# 1. Evaluate navigation (check for broken links)
links = soup.find_all('a')
total_links = len(links)
broken_links = [link for link in links if link.get('href') and 'http' not in link['href']]
broken_links_count = len(broken_links)
navigation_score = (1 - broken_links_count / total_links) * 100 if total_links else 0
feedback['navigation']['score'] = navigation_score
feedback['navigation']['description'] = f'{broken_links_count}/{total_links} links are broken'
```

How it works

Usability

Content Clarity

```
# 2. Evaluate content clarity (empty paragraphs)
paragraphs = soup.find_all('p')
total_paragraphs = len(paragraphs)
empty_paragraphs = [p for p in paragraphs if not p.text.strip()]
empty_paragraphs_count = len(empty_paragraphs)
content_clarity_score = (1 - empty_paragraphs_count / total_paragraphs) * 100 if total_paragraphs else 0
feedback['content_clarity']['score'] = content_clarity_score
feedback['content_clarity']['description'] = f'{empty_paragraphs_count}/{total_paragraphs} paragraphs are empty'
```

How it works

Usability

Aesthetic

```
# 3. Evaluate aesthetics (missing images)
images = soup.find_all('img')
total_images = len(images)
missing_images = [img for img in images if 'src' not in img.attrs or not img['src']]
missing_images_count = len(missing_images)
aesthetic_score = (1 - missing_images_count / total_images) * 100 if total_images else 0
feedback['aesthetic']['score'] = aesthetic_score
feedback['aesthetic']['description'] = f'{missing_images_count}/{total_images} images are missing'
```

How it works

Usability

Readability

```
# 4. Evaluate readability (text length)
texts = soup.find_all(['p', 'span', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
readable_texts = [t for t in texts if len(t.text.strip()) > 50]  # Minimum threshold for text length
readability_score = len(readable_texts) / len(texts) * 100 if texts else 0
feedback['readability']['score'] = readability_score
feedback['readability']['description'] = f'{len(readable_texts)}/{len(texts)} texts meet readability standards'
```

How it works

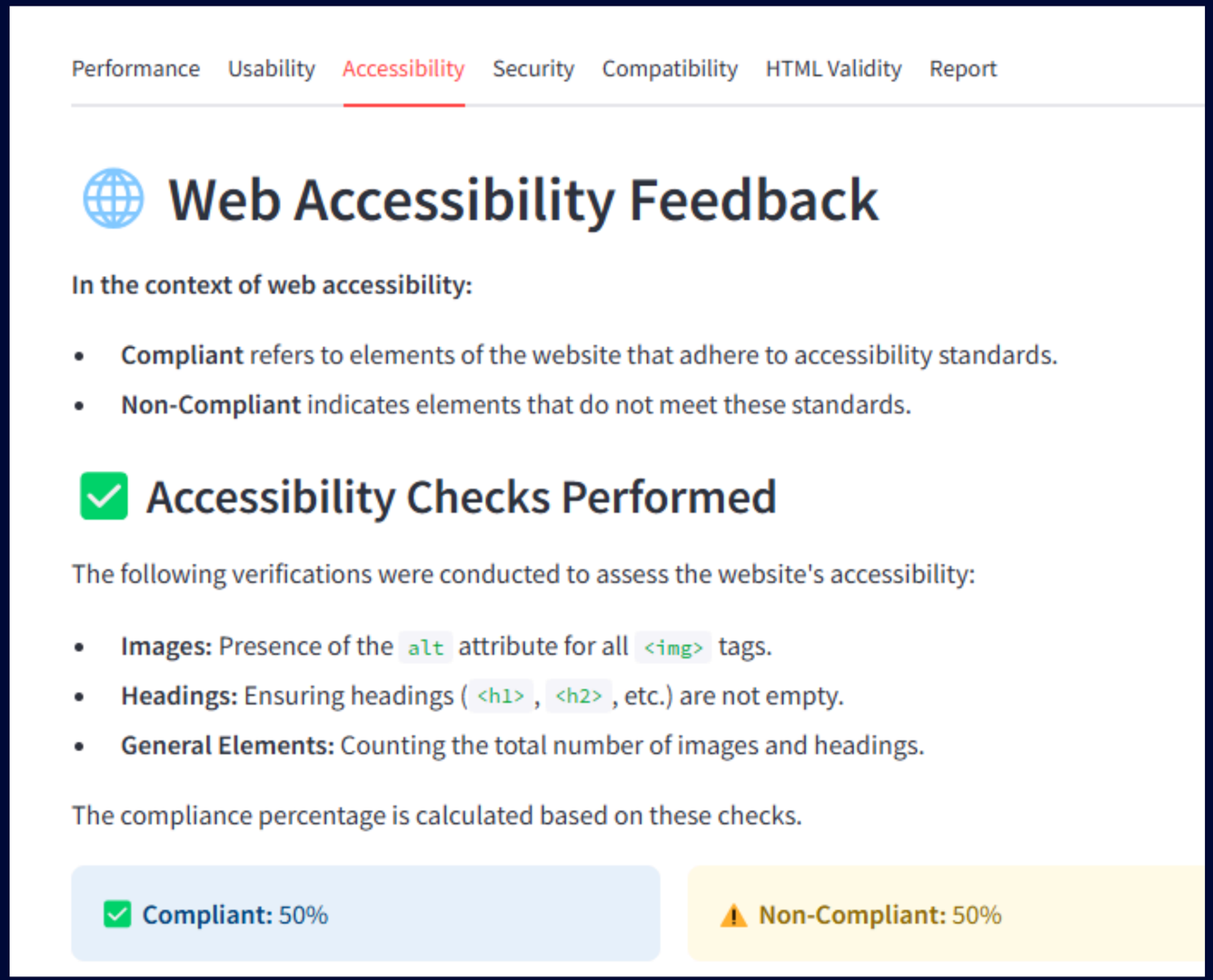
Usability

Style Consistency


```
# 5. Check for consistent styling (CSS presence)
css_links = soup.find_all('link', {'rel': 'stylesheet'})
inline_styles = soup.find_all(style=True)
if css_links or inline_styles:
    feedback['style_consistency']['score'] = 100
    feedback['style_consistency']['description'] = 'The website has consistent styling with CSS.'
else:
    feedback['style_consistency']['score'] = 0
    feedback['style_consistency']['description'] = 'No consistent styling detected (missing CSS).'
```

How it works

Accesibility



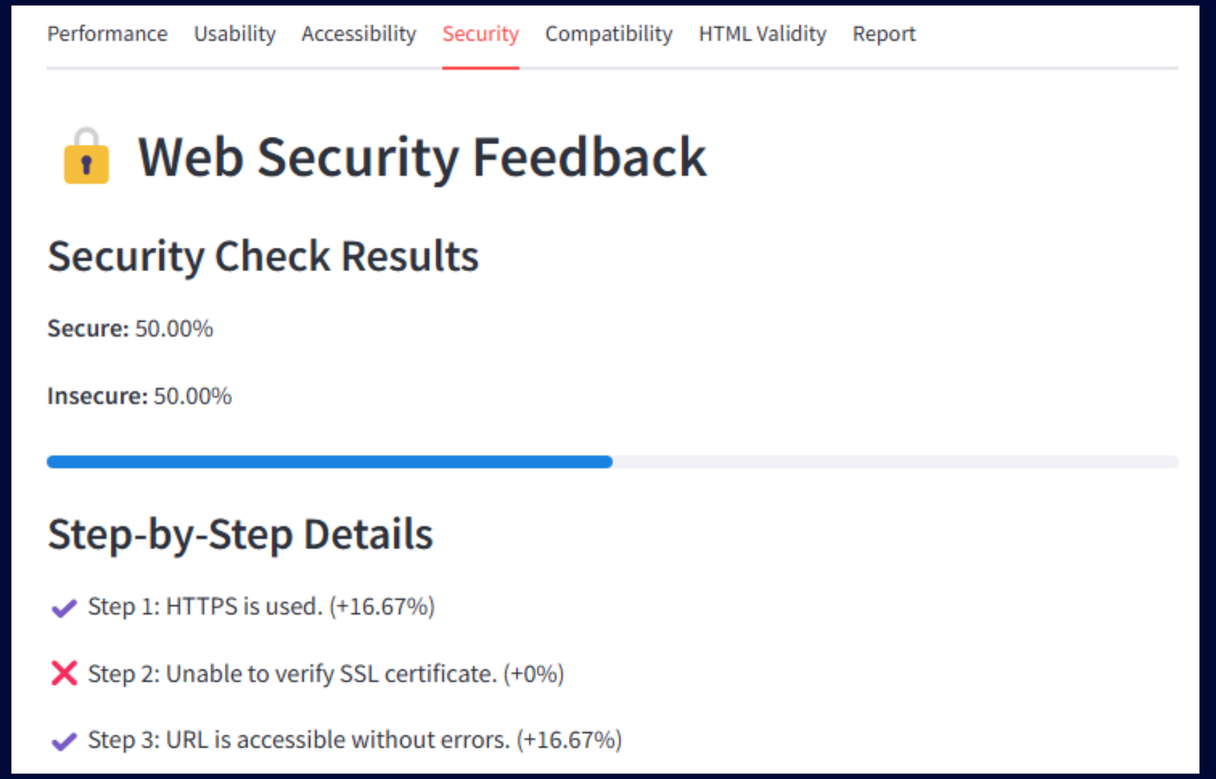
How it works

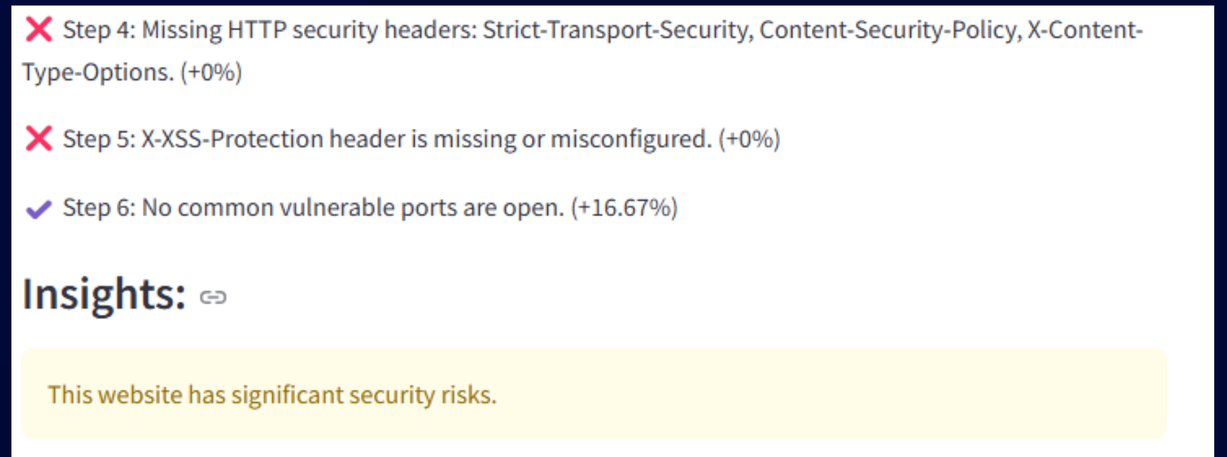
Accesibility

```
# --- Check for images with 'alt' attributes ---
images = soup.find_all('img')
compliant_images = sum(1 for img in images if img.get('alt'))  # Images with 'alt'
non_compliant_images = len(images) - compliant_images

# --- Check for empty headings ---
headings = soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
empty_headings = sum(1 for h in headings if not h.text.strip())  # Headings without visible text

# --- Check for the 'lang' attribute in <html> ---
html_tag = soup.find('html')
lang_attribute = html_tag.get('lang') if html_tag else None
lang_compliant = bool(lang_attribute)
```

How it works

Security





How it works

Security

```
# Step 1: Check if HTTPS is used
if url.startswith("https://"):
    feedback['secure'] += contribution_per_check
    details.append(f"✔ Step 1: HTTPS is used. (+{contribution_per_check:.2f}%)")
else:
    feedback['insecure'] += contribution_per_check
    details.append(f"❌ Step 1: HTTPS is not used. (+0%)")
```

How it works

Security

```
# Step 2: Verify SSL certificate validity
try:
    context = ssl.create_default_context()
    with context.wrap_socket(socket.socket(socket.AF_INET), server_hostname=url) as connection:
        connection.settimeout(5)
        connection.connect((url, 443))
        certificate = connection.getpeercert()
        if certificate:
            expiry_date = datetime.strptime(certificate['notAfter'], '%b %d %H:%M:%S %Y %Z')
            if expiry_date > datetime.now():
                feedback['secure'] += contribution_per_check
                details.append(f"✔ Step 2: SSL certificate is valid and not expired. (+{contribution_per_check:.2f}%)")
            else:
                feedback['insecure'] += contribution_per_check
                details.append("❌ Step 2: SSL certificate has expired. (+0%)")
        else:
            feedback['insecure'] += contribution_per_check
            details.append("❌ Step 2: No SSL certificate found. (+0%)")
except:
```

How it works

Security

```
# Step 3: Verify if the URL is accessible without errors
try:
    response = requests.get(url, timeout=5)
    if response.status_code == 200:
        feedback['secure'] += contribution_per_check
        details.append(f"✔ Step 3: URL is accessible without errors. (+{contribution_per_check:.2f}%)")
    else:
        feedback['insecure'] += contribution_per_check
        details.append(f"❌ Step 3: URL returned status code {response.status_code}. (+0%)")
except:
    feedback['insecure'] += contribution_per_check
    details.append("❌ Step 3: URL is not accessible. (+0%)")
```

How it works

Security

```
# Step 4: Check for HTTP security headers
try:
    response = requests.get(url, timeout=5)
    headers = response.headers
    required_headers = ['Strict-Transport-Security', 'Content-Security-Policy', 'X-Frame-Options', 'X-Content-Type-Options']
    missing_headers = [header for header in required_headers if header not in headers]
    if not missing_headers:
        feedback['secure'] += contribution_per_check
        details.append(f"✔ Step 4: All required HTTP security headers are present. (+{contribution_per_check:.2f}%)")
    else:
        feedback['insecure'] += contribution_per_check
        details.append(f"❌ Step 4: Missing HTTP security headers: {', '.join(missing_headers)}. (+0%)")
except:
    feedback['insecure'] += contribution_per_check
    details.append("❌ Step 4: Unable to check HTTP security headers. (+0%)")
```

How it works

Security

```
# Step 5: Check for XSS protection
try:
    response = requests.get(url, timeout=5)
    headers = response.headers
    if 'X-XSS-Protection' in headers and headers['X-XSS-Protection'] == '1; mode=block':
        feedback['secure'] += contribution_per_check
        details.append(f"✔ Step 5: X-XSS-Protection header is correctly configured. (+{contribution_per_check:.2f}%)")
    else:
        feedback['insecure'] += contribution_per_check
        details.append("❌ Step 5: X-XSS-Protection header is missing or misconfigured. (+0%)")
except:
    feedback['insecure'] += contribution_per_check
    details.append("❌ Step 5: Unable to verify X-XSS-Protection header. (+0%)")
```

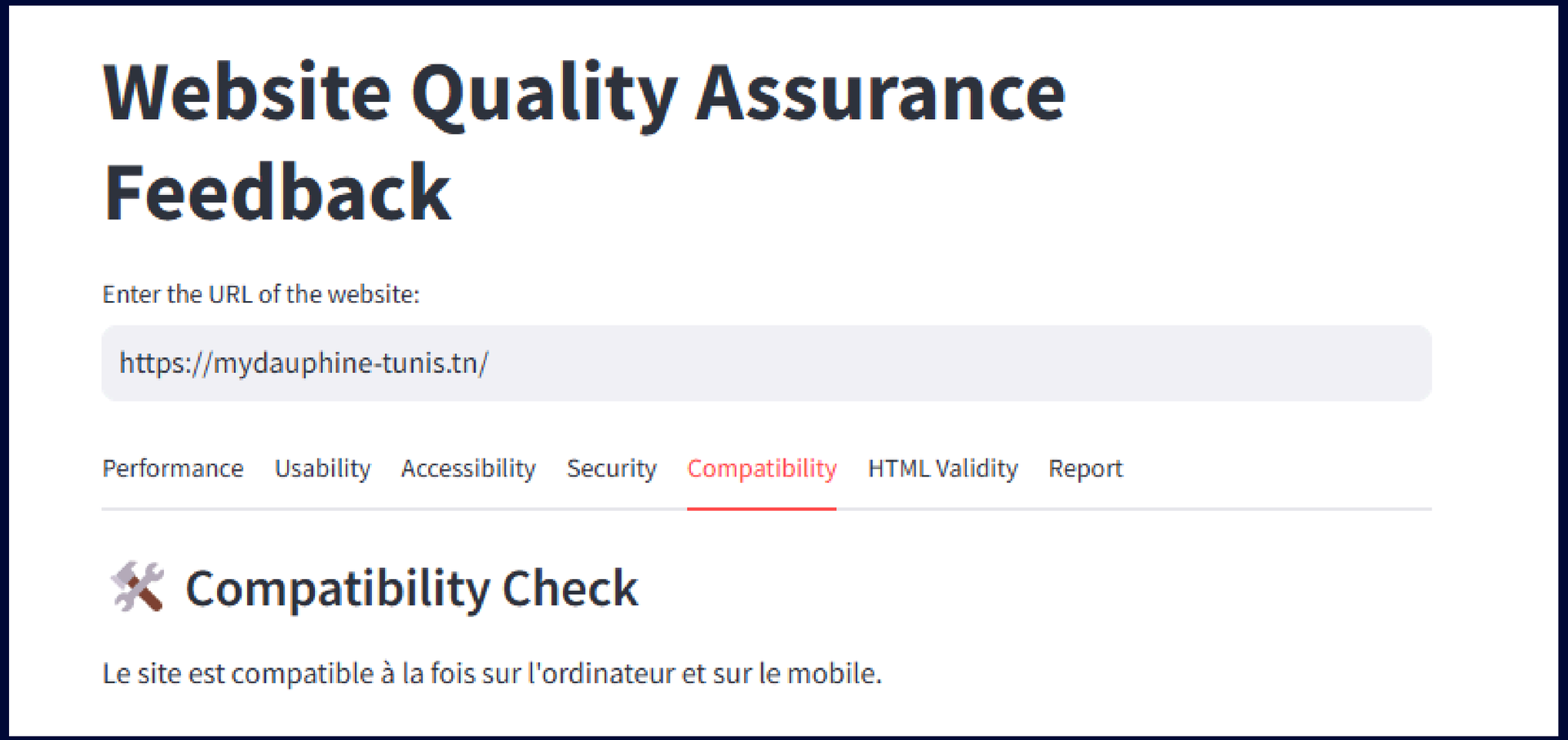
How it works

Security

```
    # Step 6: Check for Open Ports
    try:
        hostname = url.replace("https://", "").replace("http://", "").split('/')[0]
        open_ports = []
        for port in [21, 22, 23, 25, 110, 143]:  # Common vulnerable ports
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
                sock.settimeout(2)
                if sock.connect_ex((hostname, port)) == 0:
                    open_ports.append(port)
        if not open_ports:
            feedback['secure'] += contribution_per_check
            details.append(f"✔ Step 6: No common vulnerable ports are open. (+{contribution_per_check:.2f}%)")
        else:
            feedback['insecure'] += contribution_per_check
            details.append(f"❌ Step 6: Open vulnerable ports found: {', '.join(map(str, open_ports))}. (+0%)")
    except:
        feedback['insecure'] += contribution_per_check
        details.append("❌ Step 6: Unable to check for open ports. (+0%)")
```

How it works

Compatibility



How it works

Compatibility

For Desktop

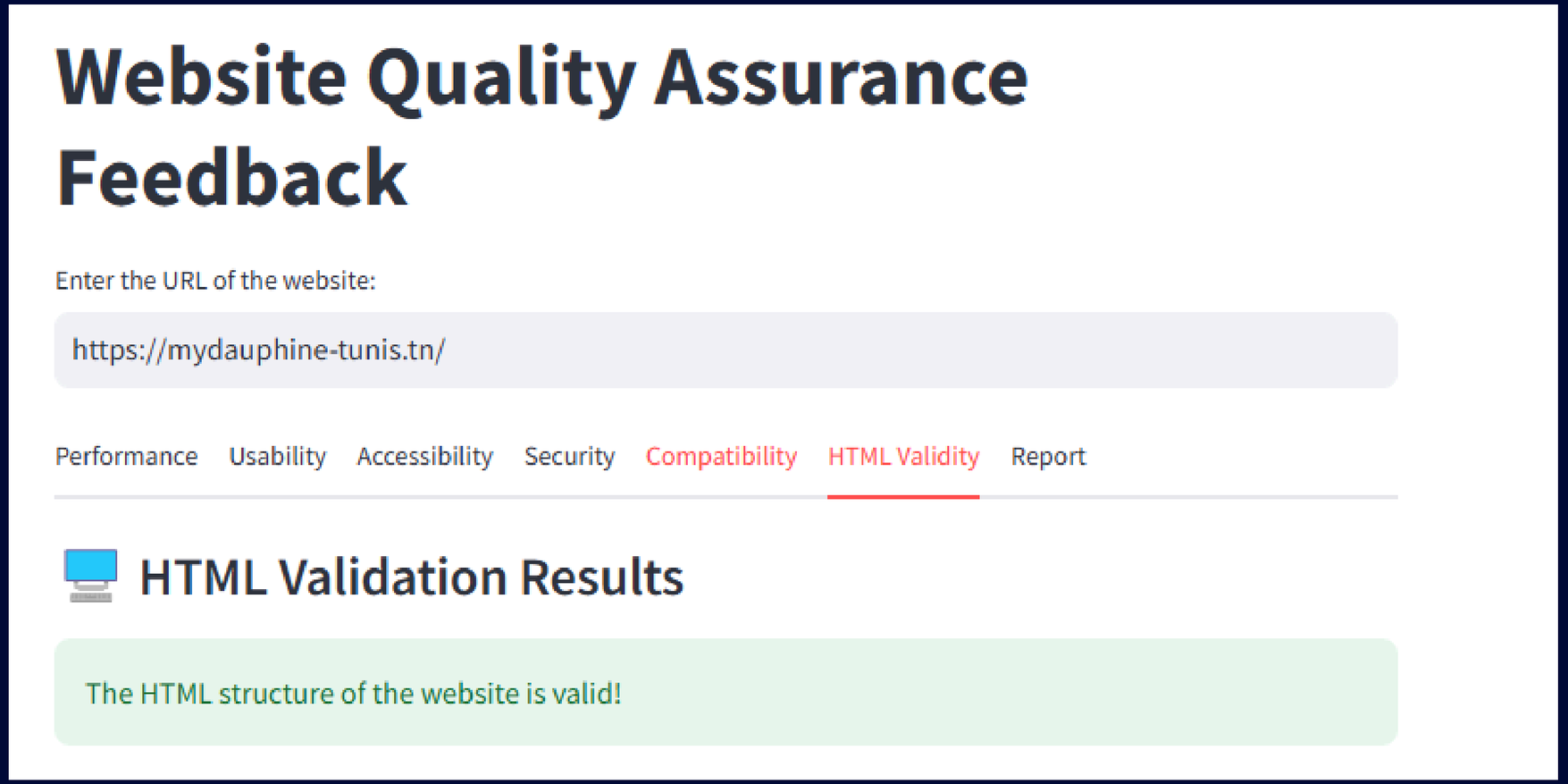
1. Sends an HTTP request to the provided URL to test site accessibility.
2. Checks for the presence of a <meta name="viewport">
3. Looks for a main container (a , <main>, or <body>)

For Mobile

1. Sends an HTTP request to the provided URL to test site accessibility.
2. Checks for the presence of a <meta name="viewport">
3. Checks the main container’s style to ensure the maximum width fits smaller screens (480 pixels).

How it works

HTML Validity



How it works

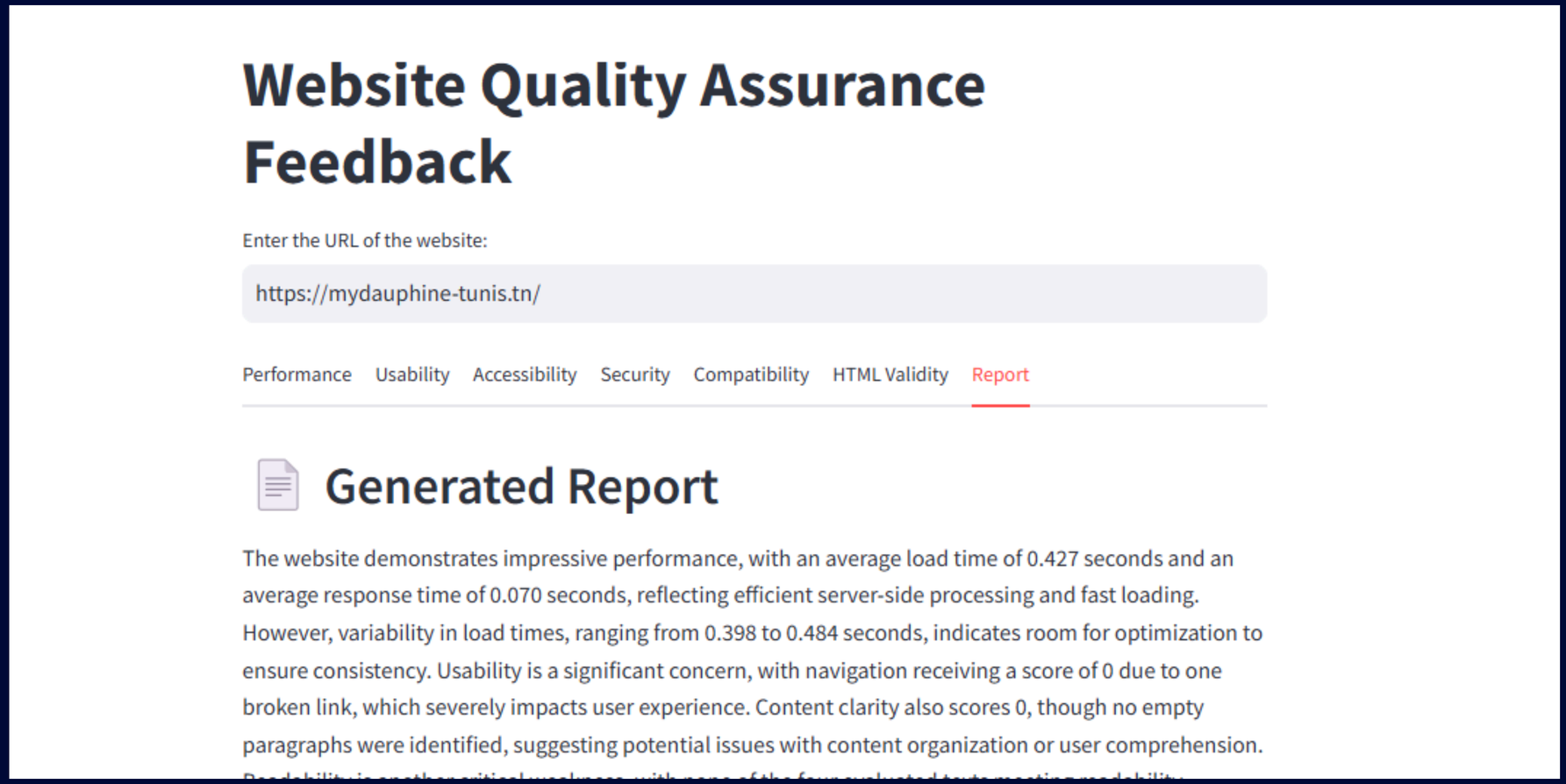
HTML Validity

- It uses the requests library to send the request and BeautifulSoup to parse the HTML content.
- Find all tags that are within a <pre> tag (which preserves the HTML structure) but contain other tags, indicating an issue.

```python
def check_html_validity(url):
    feedback = {"valid": True, "errors": []}
    try:
        # Make an HTTP request
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')

        # Check if there are improperly closed tags
        errors = soup.find_all(lambda tag: tag.parent.name == "pre" and len(tag.find_all()) > 0)
        if errors:
            feedback["valid"] = False
            feedback["errors"] = [str(error) for error in errors]
    except Exception as e:
        feedback["valid"] = False
        feedback["errors"].append(f"Error during HTML validation: {e}")

    return feedback
```

How it works

HTML Validity



How it works

Result Report

Report

A dictionary containing quality assurance information

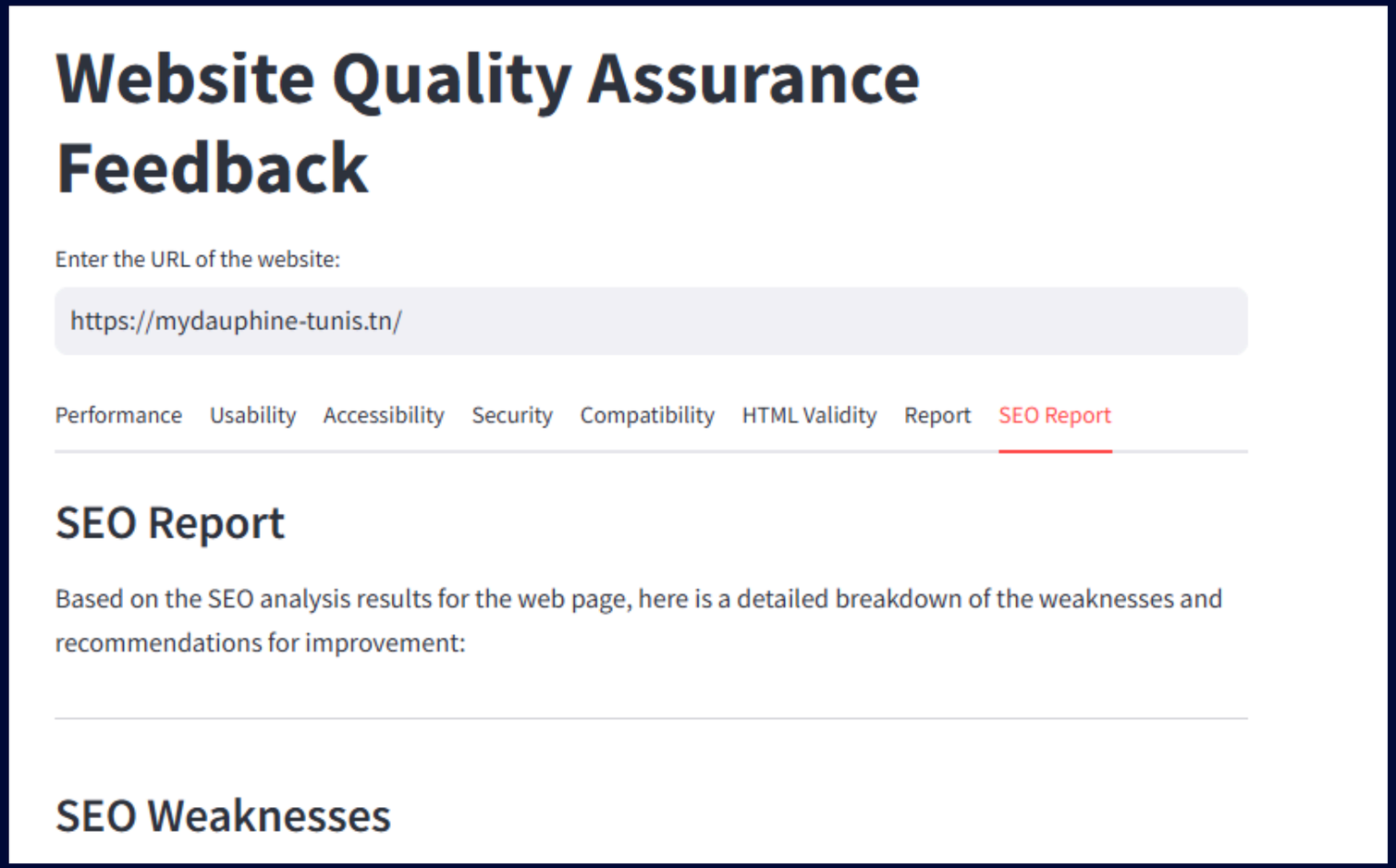
Passed to the model

Chatgpt-4o-latest

Generation of a report along with possible solutions.

How it works

Result Report



How it works

Result Report

SEO Report

Information about the Website

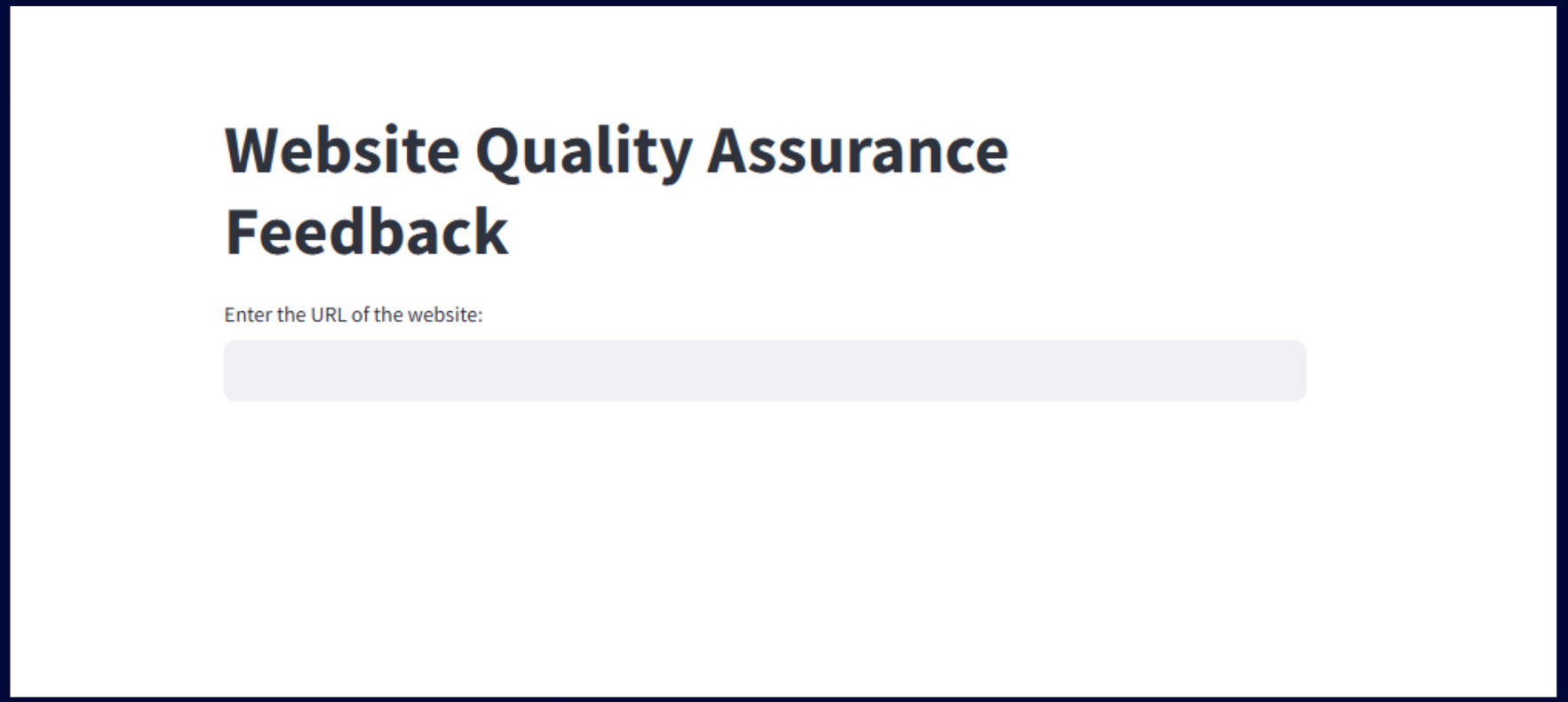
Passed to the model

Chatgpt-4o-latest

- SEO Weaknesses
- **Recommendations for Improvement**
- **Prioritized Action Plan**

How it works

Demo



Future Enhancements

01

Performance Improvements:
Optimizing the speed, efficiency, or scalability of a system.

02

New Features:
Adding functionalities that increase usability or expand capabilities.

03

User Experience (UX) Enhancements:
Improving the design, navigation, or accessibility.

04

More features using LLM:
Like detecting the topic of the website and analyse the feedbacks of the users.

Conclusion

The Website Quality Assurance Feedback project aimed to provide a comprehensive analysis of various aspects of website performance, usability, accessibility, security, and compatibility. Through automated testing and feedback generation, the project aimed to help developers and website owners understand the strengths and weaknesses of their websites, and make informed decisions to improve user experience and website quality.


THANK YOU