Project: Web-Based AI Tool for Generating Educational Content
Problem Statement:
Develop a web-based AI tool using Python, HTML, JavaScript, and OpenAI's API that dynamically generates educational content based on a given course title. The tool should allow users to enter a course title and receive the following AI-generated content:
1. Course Objective
2. Sample Syllabus
3. Three Measurable Learning Outcomes (aligned with Bloom's Taxonomy)
4. Assessment Methods
5. Recommended Readings

The project requires capturing user input, making requests to the OpenAI API, displaying the generated content, and ensuring error handling. Privacy of user data and alignment with educational standards, like Bloom's Taxonomy, are essential.

Project Components:
1. Frontend:
   - HTML: For structuring the user interface (UI).
   - CSS: For styling the form and results display.
   - JavaScript: For handling user input, making API requests (using AJAX or Fetch API), and dynamically updating the page with generated content.

2. Backend*
   - Flask (Python): To handle user input, interact with OpenAI's API, and serve the generated content.
   - OpenAI API: To generate educational content based on the user's input.

3. AI-Powered Content Generation:
   - OpenAI API (GPT-4): Generate the course objectives, syllabus, learning outcomes, assessments, and recommended readings.

4. Error Handling:
   - Ensure the system gracefully handles errors, like invalid input or failed API requests, by displaying appropriate error messages.

5. Data Privacy:
   - The application should avoid storing any personal data and inform users about data privacy practices (e.g., not storing or processing personal information locally).

Step-by-Step Implementation:
1. Frontend - HTML and CSS (Simple UI)
Create a basic web page where users can input a course title:

```html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>AI-Powered Educational Content Generator</title>
   <link rel="stylesheet" href="style.css">
</head>
<body>
   <div class="container">
      <h1>Course Content Generator</h1>
      <form id="course-form">
         <label for="course-title">Enter Course Title:</label>
         <input type="text" id="course-title" name="course-title" required>
         <button type="submit">Generate Content</button>
      </form>
      <div id="output-container">
         <!-- The generated content will be displayed here -->
      </div>
   </div>
   <script src="script.js"></script>
</body>
</html>
```

2. Backend - Flask (Python)
Set up a Flask app to handle the form submission and OpenAI API calls:
```python
from flask import Flask, render_template, request, jsonify
import openai
app = Flask(__name__)
# Replace with your OpenAI API key
openai.api_key = "your-openai-api-key"
@app.route('/')
def index():
   return render_template('index.html')
@app.route('/generate', methods=['POST'])
def generate_content():
   course_title = request.json.get('course_title')
   try:
      response = openai.Completion.create(
         model="text-davinci-003",
         prompt=f"Create educational content for a course titled '{course_title}'. "
```

```
                "The content should include a course objective, sample syllabus, "
                "three measurable learning outcomes (aligned with Bloom's Taxonomy), "
                "assessment methods, and recommended readings.",
            max_tokens=500
        )
        generated_content = response.choices[0].text.strip()
        return jsonify({'content': generated_content}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    app.run(debug=True)
```

3. JavaScript - Fetch API for Dynamic Content Loading
Use JavaScript to handle form submission and update the UI with the generated content without refreshing the page:

```javascript
document.getElementById('course-form').addEventListener('submit', async function(event) {
    event.preventDefault();
    const courseTitle = document.getElementById('course-title').value;
    const outputContainer = document.getElementById('output-container');
    try {
        const response = await fetch('/generate', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ course_title: courseTitle }),
        });
        const result = await response.json();
        if (response.ok) {
            outputContainer.innerHTML = `<h2>Generated Course
Content:</h2><pre>${result.content}</pre>`;
        } else {
            outputContainer.innerHTML = `<p>Error: ${result.error}</p>`;
        }
    } catch (error) {
        outputContainer.innerHTML = `<p>An error occurred: ${error.message}</p>`;
    }
});
```

4. CSS - Styling for Better UI
Add a simple CSS file (`style.css`) to make the interface more user-friendly:

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}
.container {
    max-width: 600px;
    margin: 50px auto;
    padding: 20px;
    background-color: white;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}
h1 {
    text-align: center;
    color: #333;
}
label {
    display: block;
    margin: 15px 0 5px;
    font-weight: bold;
}
input[type="text"] {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
button {
    display: inline-block;
    padding: 10px 15px;
    color: white;
    background-color: #28a745;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}
button:hover {
    background-color: #218838;
}
#output-container {
    margin-top: 30px;
}
pre {
    background-color: #f8f8f8;
    padding: 10px;
    border-radius: 5px;
    white-space: pre-wrap;
}
```

5. Bloom's Taxonomy Alignment
When prompting OpenAI, specifically ask for learning outcomes to follow Bloom's Taxonomy by using action verbs like:
- Remembering: Define, List, Recall
- Understanding: Explain, Describe, Identify
- Applying: Use, Implement, Demonstrate
- Analyzing: Compare, Contrast, Differentiate
- Evaluating: Judge, Critique, Recommend
- Creating: Design, Develop, Formulate

You can enhance the prompt with specific instructions to ensure Bloom's Taxonomy alignment:

```python
prompt=f"Create educational content for a course titled '{course_title}'. "
       "The content should include a course objective, sample syllabus, "
       "three measurable learning outcomes (aligned with Bloom's Taxonomy: create, analyze,
evaluate), "
       "assessment methods, and recommended readings."
```

6. Testing and Error Handling
Test the application by entering different course titles. Make sure:
- The AI generates clear, well-structured educational content.
- The UI updates without refreshing.
- Errors (e.g., invalid inputs or API issues) are handled gracefully, showing user-friendly error messages.

7. Documentation
Ensure that your documentation covers:
- Setup Instructions: How to clone the project, install dependencies, and run it.
- API Keys: Instructions for obtaining and setting up the OpenAI API key.
- Usage: How to use the tool to generate educational content.
- Testing: How to test different scenarios, including error cases.

Conclusion:

This project demonstrates how to integrate AI with a user-friendly web interface, enabling dynamic generation of educational content. By leveraging Flask for backend processing and OpenAI for content generation, the tool helps instructors or educators quickly generate comprehensive course materials. The project also emphasizes best practices in error handling, data privacy, and Bloom's Taxonomy alignment, making it a practical tool for educational settings.