Code Refactoring and Bug Fixing

Scenario:
A team of enthusiastic data scientists embarked on a mission to develop a Note Taking Application using Python, Flask, and HTML. However, their lack of experience in backend development has led to challenges in making the application fully functional. Recognizing your proficiency in backend development, you have been tasked with fixing the broken code and ensuring the application works seamlessly.

Task:
Refactor the existing codebase and ensure the proper functioning of the Note Taking Application. Document all identified bugs during the debugging process. Remember, the task is not about recreating the app from scratch. Your goal is to fix the already existing codebase and make the application work as intended.

More Details:
The application's home route contains a text field and a button. Users can add a note, and all the notes should be displayed as an unordered list below the text field on the same page.

Solution

Bug Report and Resolution Approach

Bug 1: Incorrect HTTP Method Handling
Issue: The original code only allowed POST requests to the root route ('/').
Impact: Users couldn't view the page initially, as browsers typically send GET requests when accessing a URL directly.
Resolution: We expanded the route to handle both GET and POST requests by changing the methods parameter to ["GET", "POST"].

Bug 2: Improper Form Data Retrieval
Issue: The code was using request.args.get() to retrieve form data, which is typically used for query parameters in GET requests.
Impact: This would fail to retrieve data submitted via POST requests, resulting in no notes being added.
Resolution: We replaced request.args.get() with request.form.get(), which correctly retrieves data from POST request forms.

Bug 3: Unconditional Note Addition
Issue: The original code attempted to add a note on every request, including GET requests.
Impact: This could lead to adding empty or None values to the notes list, cluttering it with useless entries.

Resolution: We added a conditional check if request.method == "POST": to ensure notes are only added when a POST request is made (i.e., when the form is submitted).

Approach to Resolution

Analyze the Code: We carefully reviewed the original code to understand its intended functionality.
Identify Inconsistencies: We noted the mismatch between the HTTP method handling and form data retrieval method.
Apply Best Practices: We updated the code to follow Flask best practices for handling form submissions.

Conclusion:
By implementing these changes, we've created a more robust and functional Flask application that correctly handles both GET and POST requests, properly retrieves form data, and only adds notes when intended.