Debugging and Refactoring of a Flask-Based Note Taking Application

- Nitisha Naigaonkar

---

1. Introduction

This report documents the debugging and refactoring of a simple Flask-based note taking application. The application consists of a single home route that allows users to enter notes and display them on the same page. While the intended functionality is straightforward, the original implementation contained multiple issues that prevented the application from running correctly. This report analyzes the given code, identifies the bugs, explains the debugging process, and presents the final working solution.

---

2. Problem Statement

The objective of the application is to:

- Display a home page containing a text input field and a submit button
- Allow users to enter notes
- Display all submitted notes as an unordered list on the same page

However, when the application was executed and accessed via the browser, it resulted in a **405 Method Not Allowed** error. As a result, the home page could not be loaded and the application was unusable. The task was to refactor the code so that the application functions correctly and to document the bugs encountered during the process.

---

3. Analysis of the Original Code

3.1 Original Backend (app.py)

The following backend code was provided as part of the original application

App :

```
from flask import Flask, render_template, request

app = Flask(__name__)

notes = []
@app.route('/', methods=["POST"])
def index():
    note = request.args.get("note")
    notes.append(note)
    return render_template("home.html", notes=notes)

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

3.2 Original Frontend (home.html)

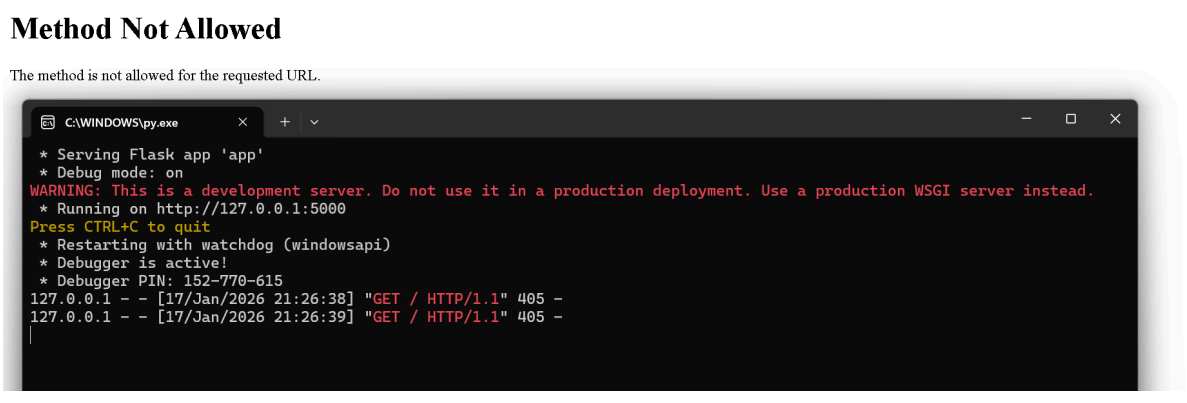
The corresponding HTML file was provided as follows

home:

```
<form action="">
    <input type="text" name="note" placeholder="Enter a
note">
    <button>Add Note</button>
</form>
```

---

4. Bugs Identified During Debugging



4.1 HTTP Method Mismatch

Observation:
The Flask route for the home page was configured to accept only `POST` requests.

```
@app.route('/', methods=["POST"])
```

Issue:
When a user opens a webpage, the browser sends a `GET` request by default. Since the route did not allow `GET` requests, Flask returned a **405 Method Not Allowed** error.

Impact:
The application failed to load the home page.

4.2 Incorrect Method for Reading Form Data

Observation:
The application attempted to access user input using:

```
note = request.args.get("note")
```

Issue:
`request.args` is used to retrieve query parameters from `GET` requests, whereas form submissions using `POST` store data in `request.form`.

Impact:
Once POST requests were enabled, the application would still fail to capture user input correctly.

4.3 HTML Form Default Behavior

Observation:
The HTML form did not specify a request method:

```
<form action="">
```

Issue:
By default, HTML forms submit data using the `GET` method. This behavior conflicted with the backend route, which only accepted `POST`.

Impact:
This mismatch directly contributed to the 405 error.

4.4 Missing Input Validation

Observation:
User input was appended directly to the notes list without validation.

```
notes.append(note)
```

Impact:
Empty notes could be submitted and rendered as empty list items on the page.

---

5. Debugging Requirements and Approach

Based on the issues identified, the following steps were required to fix the application:

1. Allow the home route to handle both `GET` and `POST` requests
2. Align frontend form submission with backend request handling
3. Correctly retrieve form data from POST requests
4. Prevent empty inputs from being stored
5. Keep the solution simple and readable

---

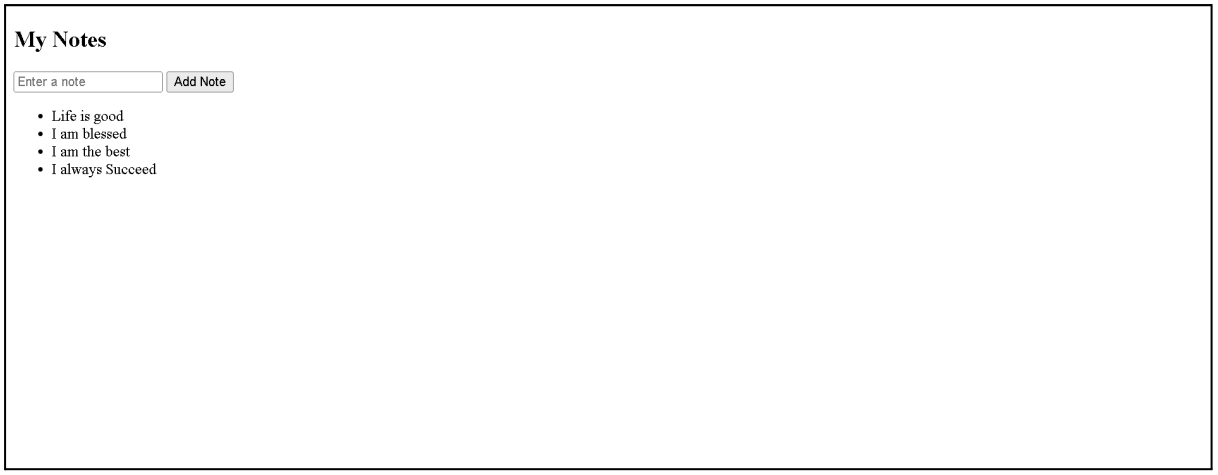
6. Application Flow After Refactoring

After refactoring, the application follows this flow:

1. The user accesses the home page `(GET /)`
2. Flask renders the page along with any existing notes
3. The user enters a note and submits the form
4. A `POST` request is sent to the server
5. The server validates and stores the note in memory
6. The updated list of notes is rendered on the same page

---

7. Final Result



After applying the fixes:

- The application loads successfully without errors
- Notes can be added and displayed correctly
- Empty inputs are ignored
- The codebase is clean, readable, and easy to understand
- All functional requirements of the task are met

The final implementation focuses on correctness and clarity rather than adding unnecessary features.

---

8. Known Limitation and Future Scope

The application uses an in-memory Python list to store notes. As a result, all notes are lost when the server restarts. This limitation was intentionally accepted to keep the application lightweight and focused on debugging concepts.

In the future, persistence can be added using a lightweight database such as SQLite, along with basic UI improvements.