Bug Analysis and Refactoring of Note Taking Application

Intern Name: Nikhil Somnath Borade

Internship: Data Science with GenAI

Organization: Innomatics Research Labs

Task: Debugging & Refactoring a Note Taking Application

---

1. Introduction

This document explains the bugs identified in a Flask-based Note Taking Application and the approach used to fix them. The application initially contained multiple issues related to routing, form handling, and template rendering, which prevented it from functioning correctly. Each bug was carefully analyzed and resolved to make the application fully operational.

---

2. Overview of the Application

The application contains:

- A single **Home Route**
- A **text input field** to enter notes
- A **button** to add notes
- An **unordered list** to display all notes on the same page

---

3. Bugs Identified and Fixes Applied

Bug 1: Home Route Accepted Only POST Requests

Issue:
The home route was configured to accept only POST requests. When the user accessed the application in a browser, a GET request was sent, causing the application to fail.

Root Cause:
Browsers send GET requests by default when opening a URL.

Fix Applied:
Updated the route to accept both GET and POST methods.

Solution:

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

Bug 2: Incorrect Method Used to Fetch Form Data

Issue:
Form data was accessed using request.args, which is meant for query parameters.

Root Cause:
The form submits data using POST, not GET.

Fix Applied:
Replaced request.args.get() with request.form.get().

Solution:

```
note = request.form.get('note')
```

Bug 3: Form Did Not Specify HTTP Method

Issue:
The HTML form did not specify a method, so it defaulted to GET.

Root Cause:
Missing method="POST" attribute in the form tag.

Fix Applied:
Explicitly added POST method to the form.

Solution:

```
<form method="POST">
```

Bug 4: Empty Notes Could Be Added

Issue:
The application allowed empty or whitespace-only notes to be added.

Root Cause:
Lack of input validation.

Fix Applied:
Used string validation with strip() to ensure meaningful input.

Solution:

```
if note and note.strip(): notes.append(note.strip())
```

Bug 5: Duplicate Notes on Page Refresh

Issue:
Refreshing the page after submitting a note caused the same note to be added again.

Root Cause:
Form submission was not redirected, leading to repeated POST requests.

Fix Applied:
Implemented the **POST–Redirect–GET** pattern.

Solution:

```
return redirect(url_for('index'))
```

Bug 6: TemplateNotFound Error

Issue:
The application crashed with TemplateNotFound: home.html.

Root Cause:
The HTML template file was not placed inside Flask's required templates directory.

Fix Applied:
Moved home.html into the templates folder.

Correct Structure:

```
note-app/ ├── app.py └── templates/ └── home.html
```

---

4. Final Outcome

After fixing all the identified bugs:

- The application loads correctly

- Notes can be added successfully
- Notes are displayed dynamically as an unordered list
- Duplicate and empty submissions are prevented
- Code is clean, readable, and maintainable

---

5. Conclusion

This task helped in understanding real-world debugging scenarios in Flask applications. By identifying routing issues, fixing form handling, and applying best practices, the application was successfully refactored into a working and stable version.

---

6. Learning Outcomes

- Debugging Flask routing issues
- Proper handling of GET and POST requests
- Input validation techniques
- Importance of correct project structure
- Writing clean and maintainable code