INNOMATICS RESEARCH LABS

INNOVATION. AUTOMATION. ANALYTICS

Code Refactoring and Bug Fixing Report

On

Note taking Application

Prepared By – Sakshi Yeshwant Nandardhane

InternID: IN1240908

INNOMATICS RESEARCH LABS

About Me

My name is Sakshi Nandardhane, and I am currently pursuing a Bachelor's degree in Computer Science Engineering, driven by a profound passion for data science and its real-world applications. As an intern at Innomatics Research Labs, I am immersed in projects centered on Data Science and Analysis, gaining invaluable practical experience to complement my theoretical knowledge. My aspiration is to harness the power of data-driven insights to catalyze informed decision-making and facilitate impactful transformations.

Link to Project Repo: https://github.com/SakshiYN/Note_Taking_App_IN1240908

LinkedIn Profile: https://www.linkedin.com/in/sakshi-nandardhane/

Agenda of Report:

1. **Objective of the Project**
2. **Initial Code Snippet**
3. **Identification of the Bugs**
4. **Bug Fixing and Code Refactoring**
5. **Additional CSS Styling**
6. **Modified Final Code**
7. **Final Outcome**
8. **Conclusion**

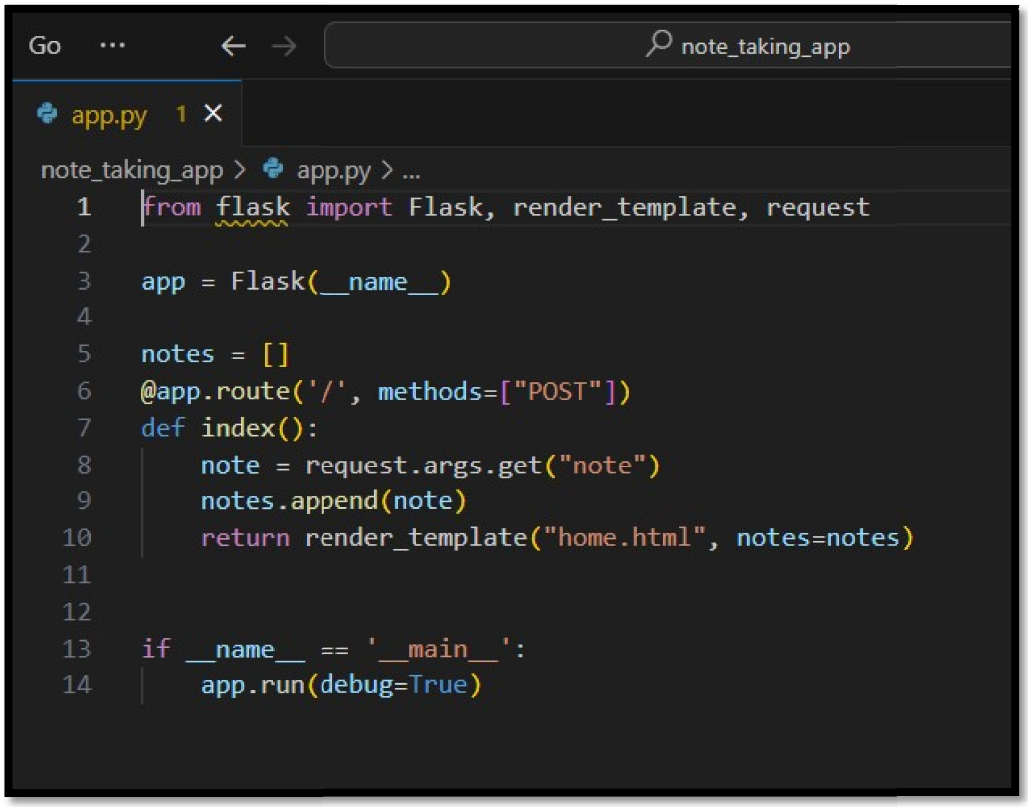
Technical Stack Used:

1. **Python**: Backend scripting language
2. **Flask**: Lightweight web framework.
3. **HTML**: Markup language for content structuring.
4. **CSS**: Styling language for visual presentation.

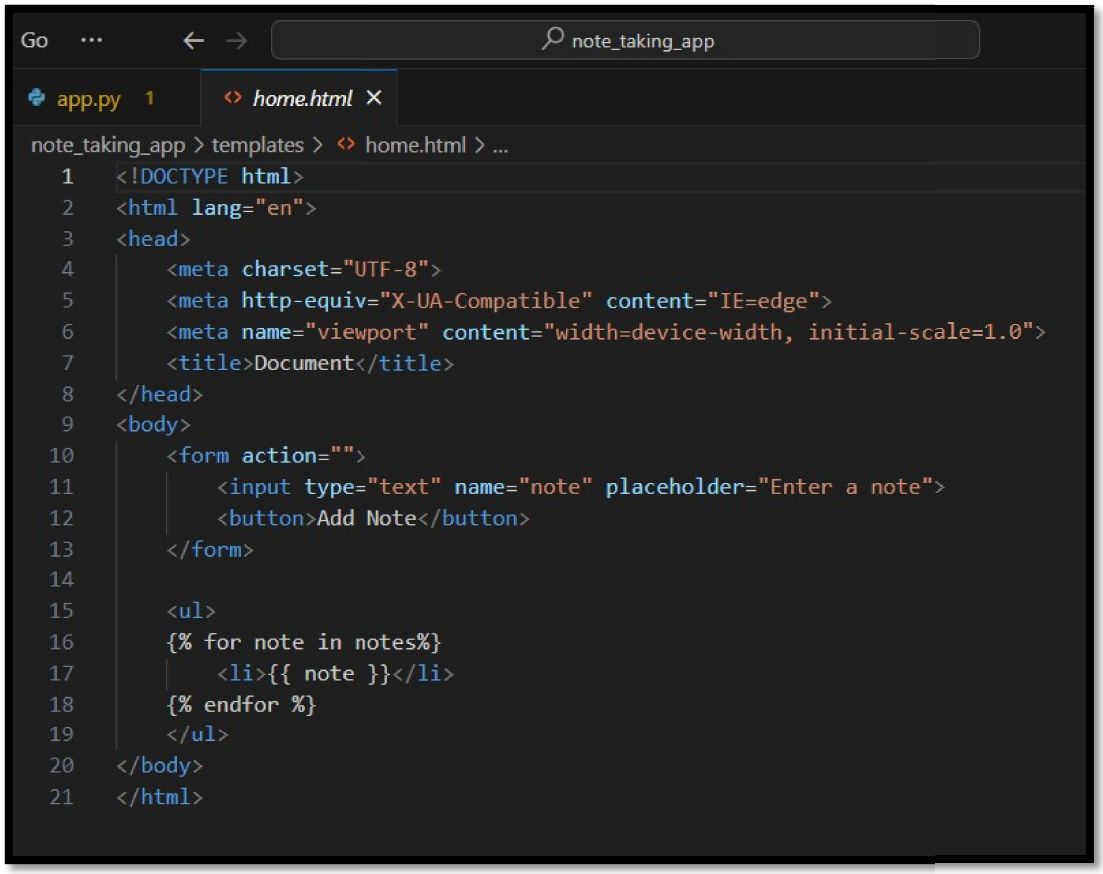
1. Objective of the Project

The objective of this project is to develop a Note Taking Application using Python, Flask, and HTML. The aim is to provide users with a simple yet effective platform to conveniently jot down notes. Despite the initial challenges stemming from the team's limited experience in backend development, our mission is to re-factor the existing codebase and rectify any issues to ensure the seamless functionality of the application. By addressing identified bugs and optimizing the code, our goal is to deliver an intuitive and user-friendly Note Taking Application that meets the needs of our users. Through this endeavor, we seek to enhance our skills in backend development .

2. Initial Code Snippet



The initial code snippet provided the foundational structure for our Note Taking Application. In the Python backend, a Flask application was established to handle the routing and logic of the application. Within the HTML front end, a basic form was implemented, featuring a text input field for users to input their notes and a submit button to add them.



However, despite this initial setup, the application encountered functionality issues, particularly in the display of added notes. The HTML template lacked the necessary logic to render the notes.

3. Identification of the Bugs

- **Bug 1**: Incorrect request method defined in the Flask route.
- **Bug 2**: Incorrect Flask request object used to obtain data.
- **Bug 3**: The index function's request method is incorrect, and there is no "if note" condition.
- **Bug 4**: Method for the form tag is missing.
- **Bug 5**: The Form Action attribute is Empty
- **Bug 6**: The Submit type Attribute is missing.

4.Bug Fixing and Code Refactoring

- **Bug 1**: **Incorrect request method defined in the Flask route.**

Original Issue:

The bug in the original code lies in restricting the route method to only "POST", which prevents clients from accessing the route with a "GET" request. The route handler for the home page (/) is restricted to only accept "POST" requests. This means that clients cannot access the route with a "GET" request to read data from the server.

```
5   notes = []
6   @app.route('/', methods=["POST"])
7   def index():
8       note = request.args.get("note")
9       notes.append(note)
10      return render_template("home.html", notes=notes)
11
```

Solution:

To fix this bug, we need to allow the route to accept both "GET" and "POST" requests. This enables clients to read and write data to the server as needed. We can achieve this by updating the `methods` parameter in the `@app.route()` decorator to include both "GET" and "POST" methods.

```
5   notes = []
6   @app.route('/', methods=["GET","POST"])
7   def index():
8       note = request.form.get("note")
9       if note:
10        notes.append(note)
11      return render_template("home.html", notes=notes)
```

- **Bug 2: Incorrect Flask request object used to obtain data**

  Original Issue:

  In the original code, the method `request.args.get("note")` is used to retrieve the note. However, this method only handles GET requests and not POST requests, potentially leading to issues with retrieving data from the form.

```
7   def index():
8       note = request.args.get("note")
9       notes.append(note)
10      return render_template("home.html", notes=notes)
```

  Solution:

  The note retrieval code should be updated to `request.form.get("note")` to handle POST requests as well. This ensures that the note can be retrieved correctly regardless of whether the request is a GET or POST request. By using `request.form.get("note")`, the code properly handles both GET and POST requests, ensuring that the note is retrieved correctly from the form data. This resolves the bug and ensures the proper functioning of the application.

```
7   def index():
8       note = request.form.get("note")
9       if note:
10          notes.append(note)
11      return render_template("home.html", notes=notes)
```

- **Bug 3: The index function's request method is incorrect, and there is no "if note" condition**

Original Issue:

In the original code, the `index()` function only handles POST requests, and there is no check to ensure that only non-empty notes are added to the list.

```
7   def index():
8       note = request.args.get("note")
9       notes.append(note)
10      return render_template("home.html", notes=notes)
```

Solution:

1. Update the index() function to handle both GET and POST requests.
2. Add a condition to check if the note is not empty before appending it to the list of notes.

By updating the `index()` function to handle both GET and POST requests, and adding the `if note:` condition to check for empty notes before appending them to the list, the code ensures that only non-empty notes are added to the list and displayed in the application. This resolves the bug and ensures the proper functioning of the application.

```
7   def index():
8       note = request.form.get("note")
9       if note:
10          notes.append(note)
11      return render_template("home.html", notes=notes)
```

➢ **Bug 4**: Method for the form tag is missing.

Original Issue:

In the original code, the method attribute is missing its value, which could lead to unexpected behavior or defaulting to a GET request.

```
 9   <body>
10       <form action="">
11           <input type="text" name="note" placeholder="Enter a note">
12           <button>Add Note</button>
13       </form>
```

Solution:

Add "POST" as the attribute value for the method attribute in the form tag. By specifying the method as "POST", the form data will be sent to the server using a POST request, ensuring that it is properly handled by the backend code. This resolves the bug and ensures that form submissions work as intended.

```
56   <body>
57       <div class="container"></div>
58       <h1>Note Taking App</h1>
59       <form action="/" method = "post">
60           <input type="text" name="note" placeholder="Enter a note">
61           <button type = "submit">Add Note</button>
62       </form>
```

➢ **Bug 5**: The Form Action attribute is Empty

Original Issue:

In the original code, the form action attribute is empty (`action=""`), which could lead to potential issues with form submission.

```
 9   <body>
10       <form action="">
11           <input type="text" name="note" placeholder="Enter a note">
12           <button>Add Note</button>
13       </form>
```

Solution:

Include a valid URL or route path in the form action attribute to specify where the form data should be submitted. By specifying a valid URL or route path (in this case, `"/"`) in the form action attribute, the form data will be submitted to the specified location when the form is submitted. Additionally, including `method="POST"` ensures that the form data is submitted using the POST method. This resolves the bug and ensures that form submissions work as intended

.

```
59       <form action="/" method = "post">
60           <input type="text" name="note" placeholder="Enter a note">
61           <button type = "submit">Add Note</button>
```

➢ **Bug 6**: The Submit type Attribute is missing.

Original Issue:

In the original code, the button element lacks the `type="submit"` attribute, which can lead to inconsistent behavior across different browsers.

```
 9   <body>
10       <form action="">
11           <input type="text" name="note" placeholder="Enter a note">
12           <button>Add Note</button>
13       </form>
```

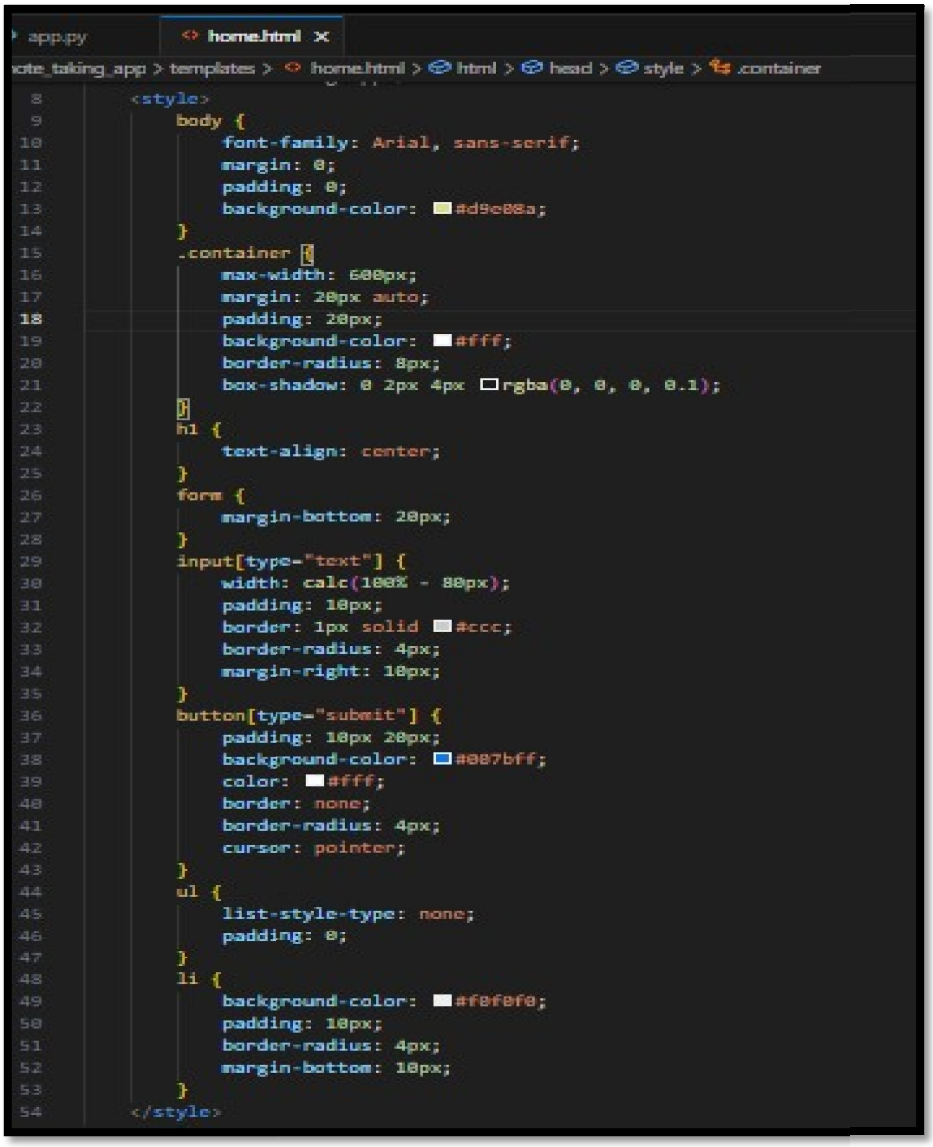
Solution:

Add the `type="submit"` attribute to the button element to explicitly specify that it is a submit button. By adding the `type="submit"` attribute, the button element is explicitly designated as a submit button. This enhances accessibility and ensures consistent behavior across different browsers, especially in scenarios where JavaScript is not enabled.

```
59       <form action="/" method = "post">
60           <input type="text" name="note" placeholder="Enter a note">
61           <button type = "submit">Add Note</button>
```

5. Additional CSS Styling

The HTML code for the Note Taking App has been enriched with some CSS styling, enhancing its visual appeal and usability. Utilizing a cohesive color scheme and font-family, the design ensures readability and consistency. The layout is optimized for various screen sizes with centered alignment. Form elements like input fields and buttons are styled for clarity, while note display sections feature organized and visually distinct presentations. These CSS enhancements elevate user experience, making the app functional and attractive.



6. Modified Final Code

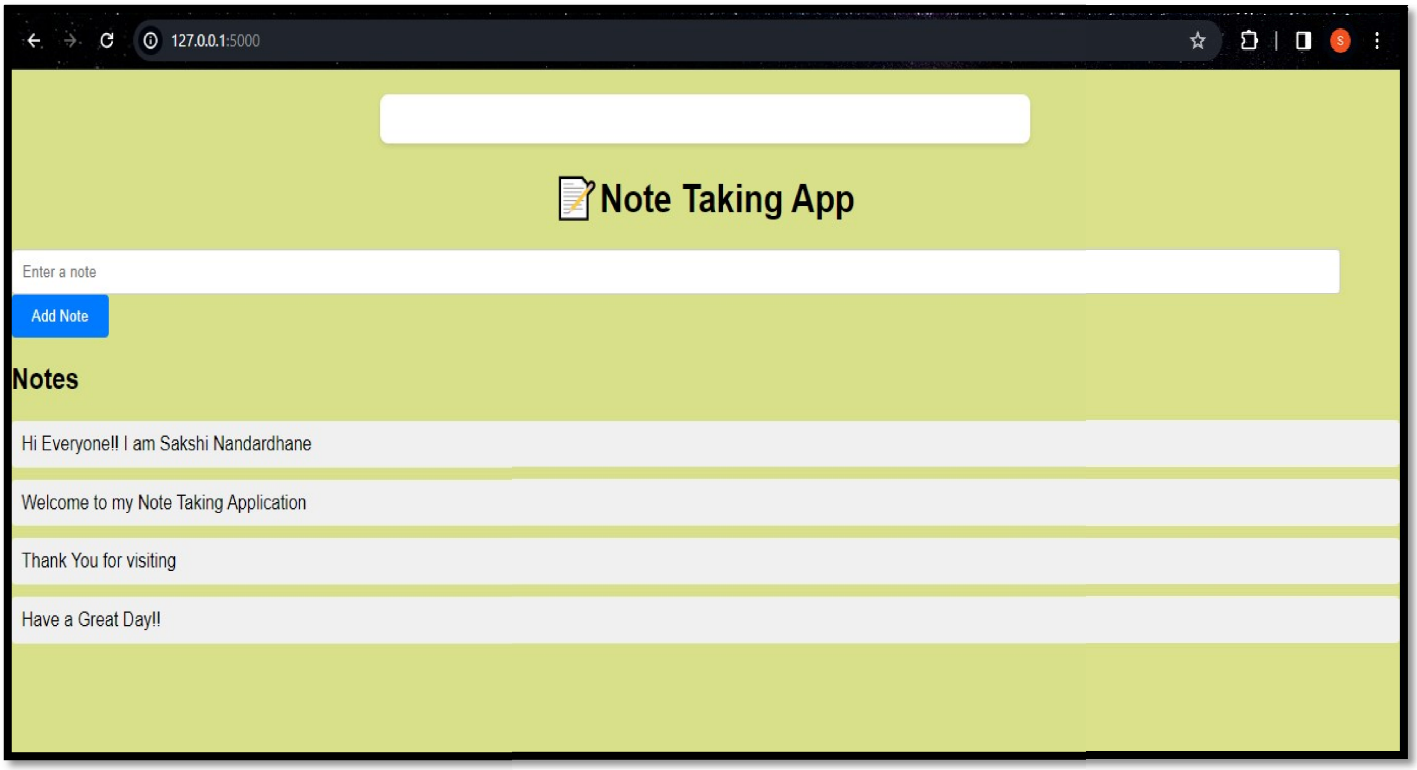
app.py

```python
from flask import Flask, render_template, request

app = Flask(__name__)

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


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

home.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Note Taking App</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #d9e08a;
        }
        .container {
            max-width: 600px;
            margin: 20px auto;
            padding: 20px;
            background-color: #fff;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
        }
        h1 {
            text-align: center;
        }
        form {
            margin-bottom: 20px;
        }
        input[type="text"] {
            width: calc(100% - 80px);
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 4px;
            margin-right: 10px;
        }
        button[type="submit"] {
            padding: 10px 20px;
            background-color: #007bff;
            color: #fff;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
        ul {
            list-style-type: none;
            padding: 0;
        }
        li {
            background-color: #f0f0f0;
            padding: 10px;
            border-radius: 4px;
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
    <div class="container"></div>
    <h1>Note Taking App</h1>
    <form action="/" method = "post">
        <input type="text" name="note" placeholder="Enter a note">
        <button type = "submit">Add Note</button>
    </form>
<h2> Notes </h2>
    <ul>
    {% for note in notes%}
        <li>{{ note }}</li>
    {% endfor %}
    </ul>
</body>
</html>
```

7.Final Outcome



8.Conclusion

Through thorough code refactoring and bug fixing, significant improvements have been made to the Note Taking App, enhancing both its functionality and user experience. This project exemplifies the importance of meticulous code review and optimization to address existing issues and ensure the smooth operation of the application. By identifying and rectifying various bugs and deficiencies, we have succeeded in refining the application and delivering a more polished and reliable product to users.

Overview of Fixes:

- Refactored code to handle both GET and POST requests.
- Corrected the method used to retrieve data from Flask's request object.
- Ensured proper handling of form submission by specifying the POST method in the form action attribute.
- Added the 'type="submit"' attribute to the button element for consistent behavior.
- Enhanced CSS styling for improved visual appeal and user experience.