INNOMATICS®
RESEARCH LABS

INNOVATION. AUTOMATION. ANALYTICS

PROJECT ON

CODE REFACTORING AND BUG FIXING

Anil Devalla

INNOMATICS
RESEARCH LABS

Problem and Use case domain understanding:-

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.

Problem solving :-

1. Working code after refactoring and bug fixing.
2. A report detailing all identified bugs along with the approach used to resolve each bug.

INNOMATICS RESEARCH LABS

Finding the Bugs Before Refactoring the Code :

Lack of HTML-Flask Connection:

- The initial code lacks a clear connection between the HTML webpage and the Flask server.
- There is no established linkage to facilitate data communication between the frontend and the backend.

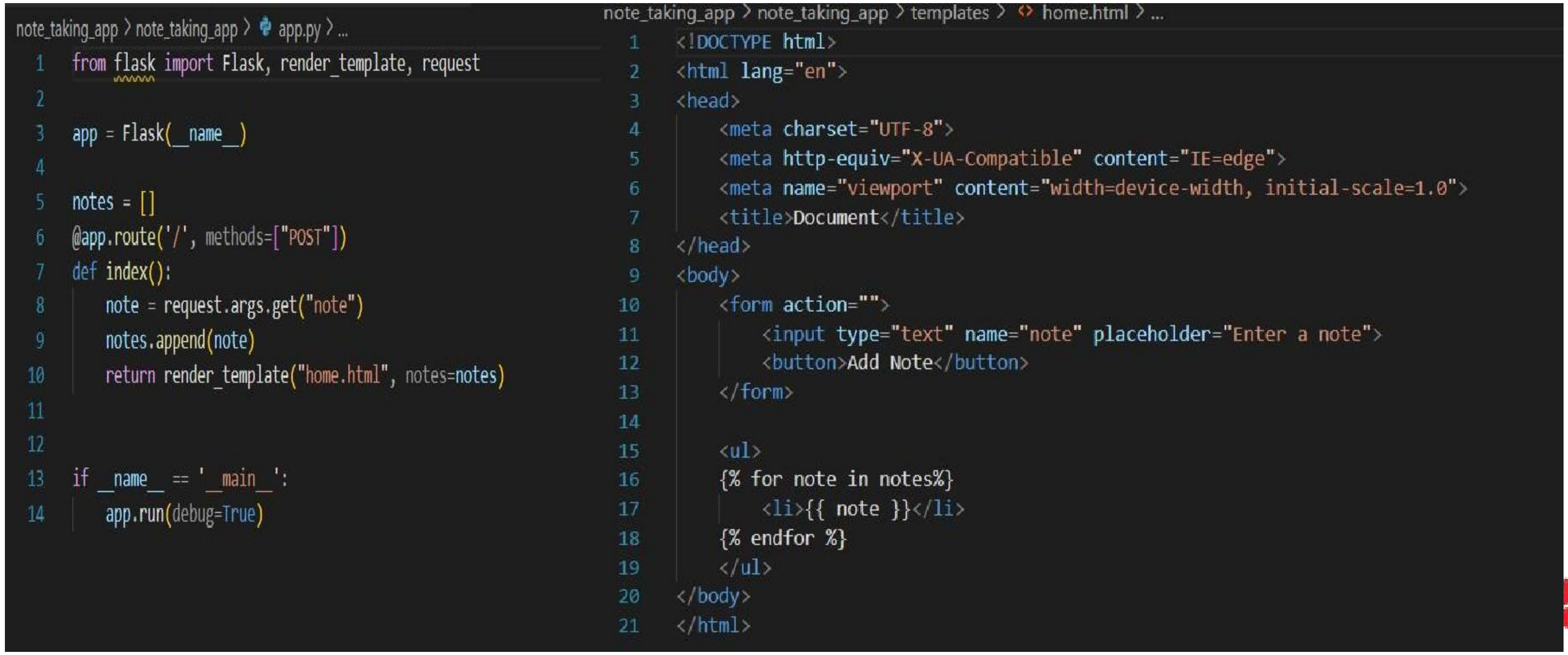
Inconsistent HTTP Method Specification:

- The method specified in the HTML form is 'POST' for submitting data, but the Flask server attempts to retrieve the data using request.args.get().
- This inconsistency in method specification can lead to an inability to properly receive and process the submitted data.

Data Retrieval Issues:

- The code employs request.args.get() to fetch data from the POST request, which might not accurately capture the submitted note on thewebpage.
- This mismatch in data retrieval methods may result in an unsuccessful retrieval of the submitted note.

Html and python files before problem solving:-



note_taking_app > note_taking_app > app.py > ...

```python
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)
```

note_taking_app > note_taking_app > templates > 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>Document</title>
</head>
<body>
    <form action="">
        <input type="text" name="note" placeholder="Enter a note">
        <button>Add Note</button>
    </form>

    <ul>
    {% for note in notes%}
        <li>{{ note }}</li>
    {% endfor %}
    </ul>
</body>
</html>
```

INNOMATICS RESEARCH LABS

Steps to Solve Code Refactoring and Bug Fixing:

Project Setup:

Create a folder in Visual Studio Code for the project.

Establish a virtual environment using the command: python -m venv .name_of_virtual_environment.

Activate the virtual environment in the Visual Studio Code terminal: .\name_of_virtual_environment\Scripts\activate. Install Flask within the virtual environment: pip install flask.

File Structure:

Organise the project with an HTML file (e.g., home.html) for the user interface.

Create a Python file (e.g., app.py) for the server-side code.

Import necessary modules and set up the Flask application.

HTML Styling:

In home.html, enhance the visual appeal by adding a light green background to the body using the HTML style tag. Modify the form in home.html to use the POST method for connection and set the action as '/' to align with server routes. Improve the aesthetics by adding a light yellow background colour for the placeholder with black text using HTML styles. **CSS Styling:**

Create a style.css file for additional styling.

Add styles to create a colourful and visually appealing user interface.

Flask Server Code:

In app.py, set the route method to methods=['POST', 'GET'] to handle both POST and GET requests. Implement the server-side logic:

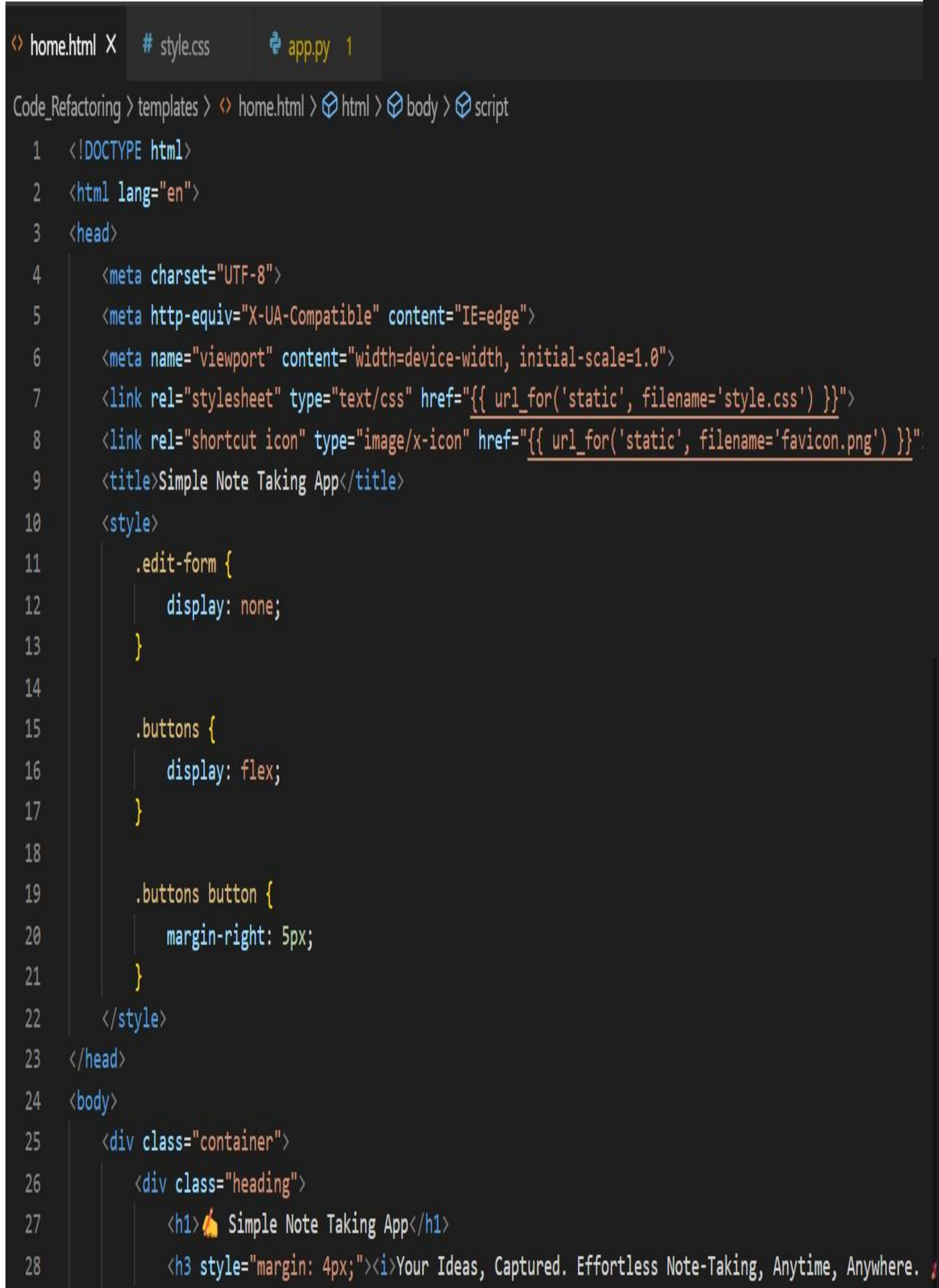
Python
```
if request.method == 'POST':
   note =
   request.form.get("note")if
   note:
      notes.append(note)
```
Ensure proper rendering of notes in the HTML template using render_template with home.html.

Enhancements with Style.css:

Link the style.css file to home.html for additional styling improvements.

Utilize CSS rules to enhance the visual presentation of the web application.

Home.html file After problem solving :-



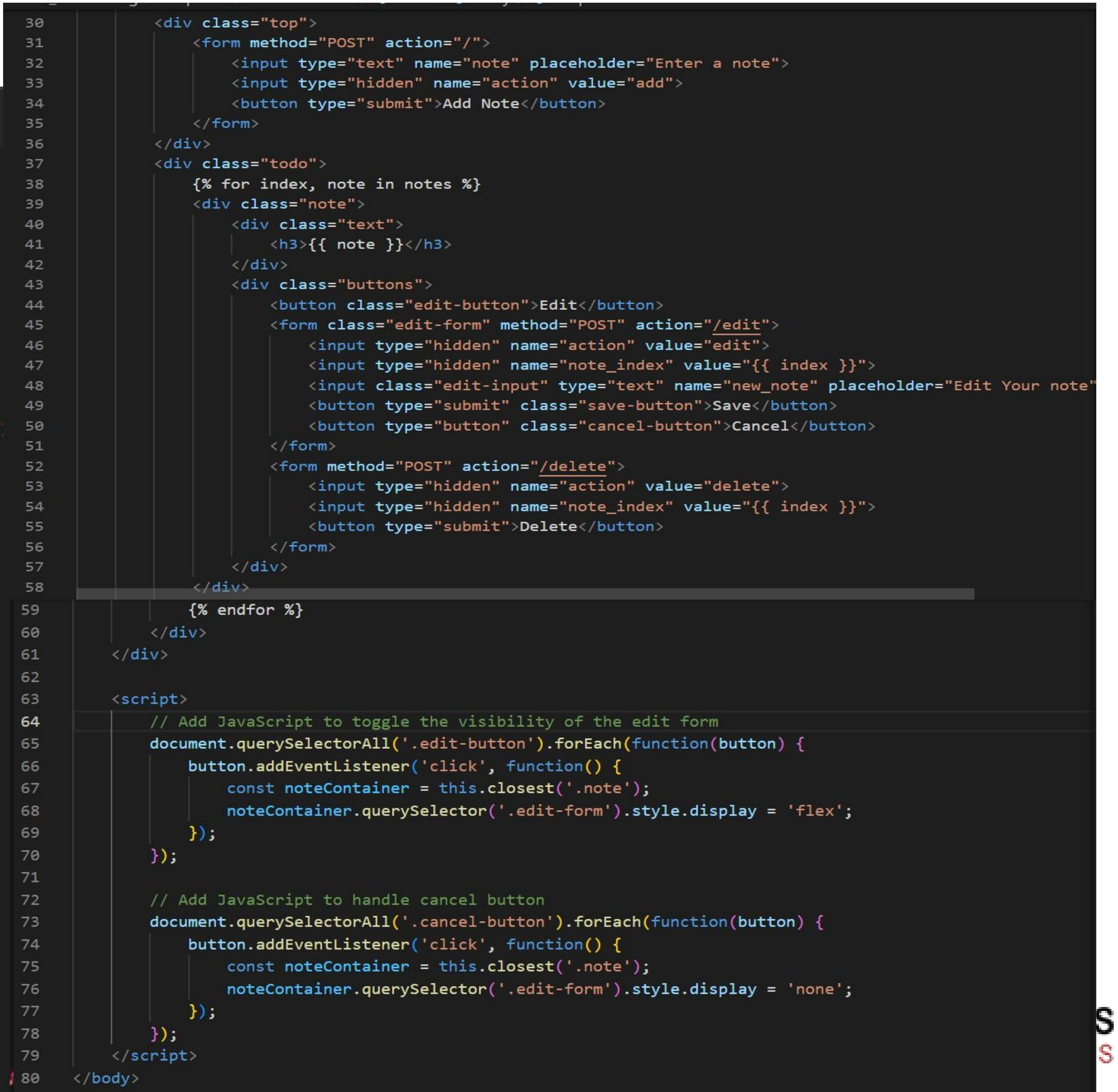
home.html X   # style.css   app.py 1

Code_Refactoring > templates > home.html > html > body > script

```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">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
    <link rel="shortcut icon" type="image/x-icon" href="{{ url_for('static', filename='favicon.png') }}"
    <title>Simple Note Taking App</title>
    <style>
        .edit-form {
            display: none;
        }

        .buttons {
            display: flex;
        }

        .buttons button {
            margin-right: 5px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="heading">
            <h1>✍ Simple Note Taking App</h1>
            <h3 style="margin: 4px;"><i>Your Ideas, Captured. Effortless Note-Taking, Anytime, Anywhere.
```



```html
        <div class="top">
            <form method="POST" action="/">
                <input type="text" name="note" placeholder="Enter a note">
                <input type="hidden" name="action" value="add">
                <button type="submit">Add Note</button>
            </form>
        </div>
        <div class="todo">
            {% for index, note in notes %}
            <div class="note">
                <div class="text">
                    <h3>{{ note }}</h3>
                </div>
                <div class="buttons">
                    <button class="edit-button">Edit</button>
                    <form class="edit-form" method="POST" action="/edit">
                        <input type="hidden" name="action" value="edit">
                        <input type="hidden" name="note_index" value="{{ index }}">
                        <input class="edit-input" type="text" name="new_note" placeholder="Edit Your note"
                        <button type="submit" class="save-button">Save</button>
                        <button type="button" class="cancel-button">Cancel</button>
                    </form>
                    <form method="POST" action="/delete">
                        <input type="hidden" name="action" value="delete">
                        <input type="hidden" name="note_index" value="{{ index }}">
                        <button type="submit">Delete</button>
                    </form>
                </div>
            </div>
            {% endfor %}
        </div>
    </div>

    <script>
        // Add JavaScript to toggle the visibility of the edit form
        document.querySelectorAll('.edit-button').forEach(function(button) {
            button.addEventListener('click', function() {
                const noteContainer = this.closest('.note');
                noteContainer.querySelector('.edit-form').style.display = 'flex';
            });
        });

        // Add JavaScript to handle cancel button
        document.querySelectorAll('.cancel-button').forEach(function(button) {
            button.addEventListener('click', function() {
                const noteContainer = this.closest('.note');
                noteContainer.querySelector('.edit-form').style.display = 'none';
            });
        });
    </script>
</body>
```

1. **HTML Structure:**
- The HTML document defines the structure of a simple note-taking web application.
- It includes a heading with a stylized pen emoji and a tagline emphasizing effortless note-taking.

2. **Form Handling with Flask:**
- The form within the 'top' section captures user input using the POST method and sends it to the root route ('/') in the Fla skserver.
- The hidden input field 'action' is utilized to distinguish the purpose, and the 'Add Note' button triggers the form submission.

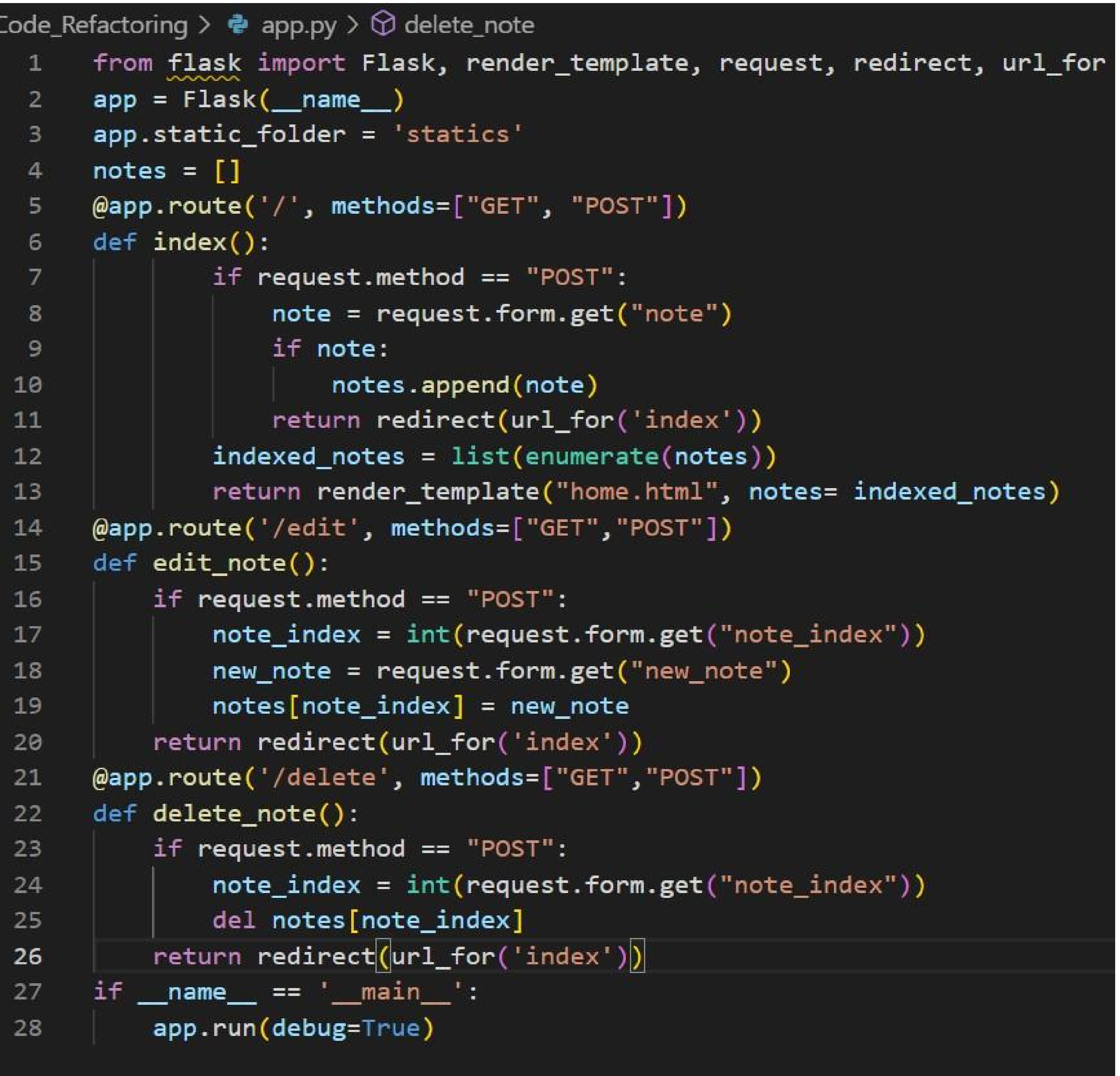
3. **Dynamic Note Display:**
- The 'todo' section dynamically displays existing notes fetched from the Flask server using Jinja templating.
- For each note, a 'note' div is generated with an 'Edit' button, allowing users to modify, save, or cancel note edits.

4. **Interactive JavaScript:**
- JavaScript functions are embedded to enhance user interaction.
- 'Edit' buttons toggle the visibility of corresponding edit forms, allowing users to edit notes in place.
- 'Cancel' buttons within edit forms hide the form, enabling users to discard their edits.

5. **Styling and Branding:**
- The design is enhanced with a 'container' div for layout consistency, CSS styles for hiding edit forms and styling buttons, and acohesive colour scheme.
- The application conveys a friendly and accessible aesthetic, utilizing emojis and concise messaging to promote a positive userexperience.

INNOMATICS RESEARCH LABS

Python file after problem solving And style css :-

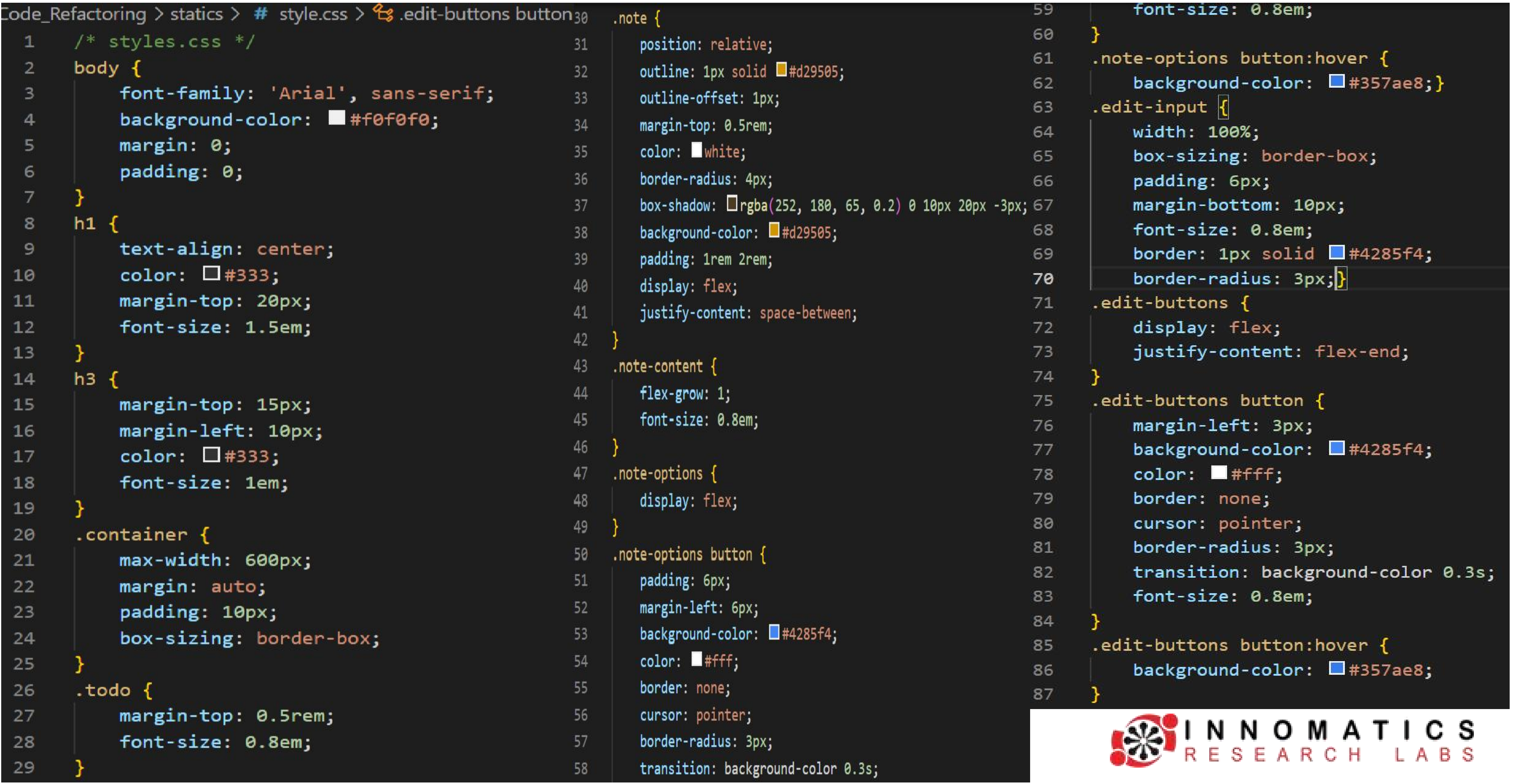

ode_Refactoring > app.py > delete_note

```python
from flask import Flask, render_template, request, redirect, url_for
app = Flask(__name__)
app.static_folder = 'statics'
notes = []
@app.route('/', methods=["GET", "POST"])
def index():
        if request.method == "POST":
            note = request.form.get("note")
            if note:
                notes.append(note)
            return redirect(url_for('index'))
        indexed_notes = list(enumerate(notes))
        return render_template("home.html", notes= indexed_notes)
@app.route('/edit', methods=["GET","POST"])
def edit_note():
    if request.method == "POST":
        note_index = int(request.form.get("note_index"))
        new_note = request.form.get("new_note")
        notes[note_index] = new_note
    return redirect(url_for('index'))
@app.route('/delete', methods=["GET","POST"])
def delete_note():
    if request.method == "POST":
        note_index = int(request.form.get("note_index"))
        del notes[note_index]
    return redirect(url_for('index'))
if __name__ == '__main__':
    app.run(debug=True)
```

Flask App Setup:
A Flask web application is created using Flask(__name__).
The static folder is set to 'statics' using app.static_folder = 'statics', but it is recommended to use the default 'static' folder for static files.
Global Variable and Routes:
A global list named notes is defined to store the notes added by users.
Three routes are defined:
'/' (Root Route):
Handles both GET and POST requests.
If the request method is POST, it retrieves the submitted note from the form andappends it to the notes list.
Redirects to the root route after adding a note or displays the existing notes usingthe 'home.html' template.
'/edit':
Handles both GET and POST requests.
If the request method is POST, it retrieves the index of the note to be edited andthe new note content. It then updates the corresponding note in the notes list. Redirects to the root route after editing a note.
'/delete':
Handles both GET and POST requests.
If the request method is POST, it retrieves the index of the note to be deleted andremoves it from the notes list.
Redirects to the root route after deleting a note.
Run the Application:
The if __name__== '__main__': block ensures that the Flask app runs only if the
script is executed directly (not imported as a module).
The app.run(debug=True) statement runs the Flask application in debug mode, allowing for real-time debugging and automatic reloading of the server on codechanges.
Template Rendering:
The render_template function is used to render the 'home.html' template, passing the list of notes to be displayed on the webpage.

INNOMATICS RESEARCH LABS

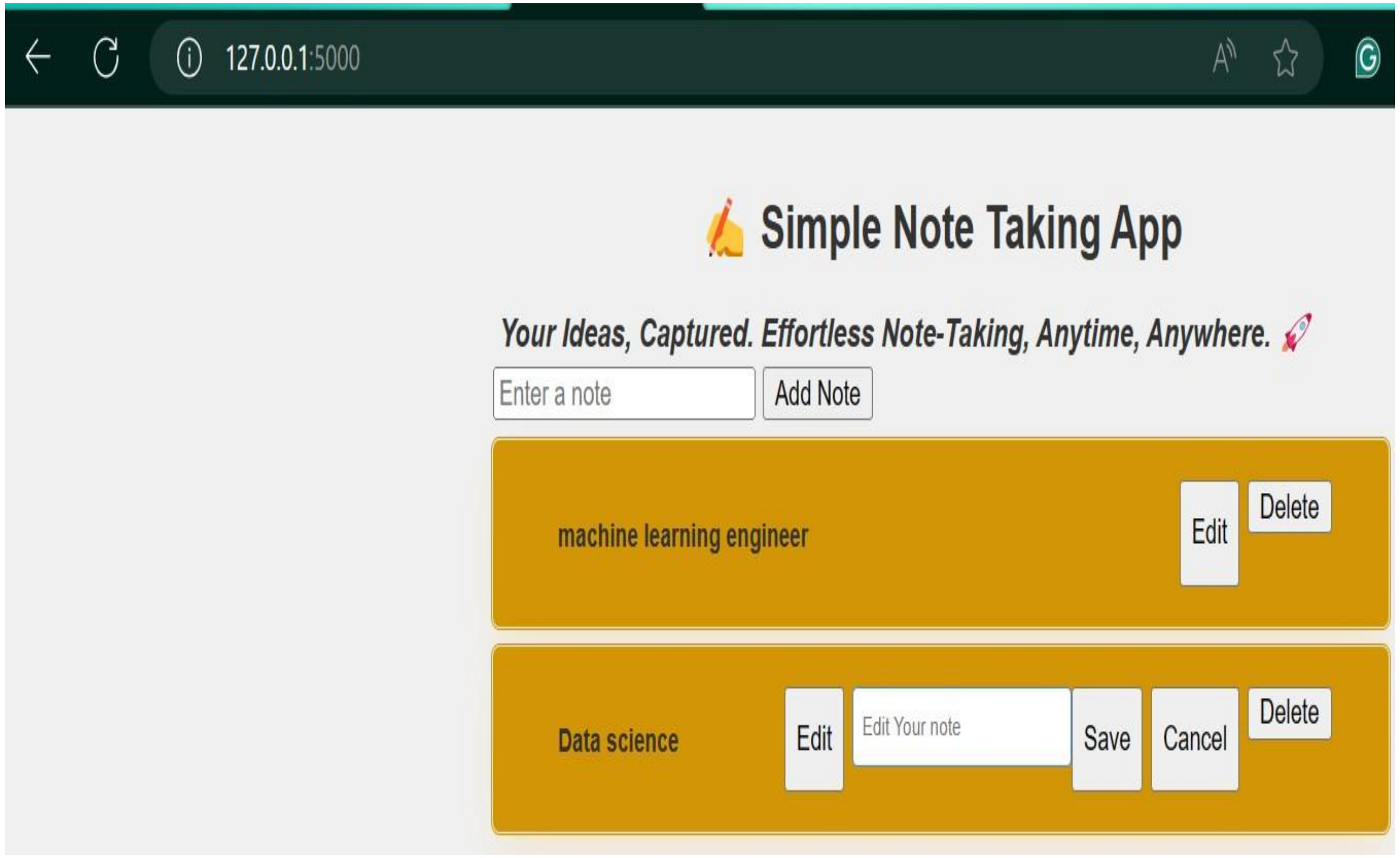
Style Css:



Code_Refactoring > statics > # style.css > .edit-buttons button

```css
/* styles.css */
body {
    font-family: 'Arial', sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
}
h1 {
    text-align: center;
    color: #333;
    margin-top: 20px;
    font-size: 1.5em;
}
h3 {
    margin-top: 15px;
    margin-left: 10px;
    color: #333;
    font-size: 1em;
}
.container {
    max-width: 600px;
    margin: auto;
    padding: 10px;
    box-sizing: border-box;
}
.todo {
    margin-top: 0.5rem;
    font-size: 0.8em;
}
.note {
    position: relative;
    outline: 1px solid #d29505;
    outline-offset: 1px;
    margin-top: 0.5rem;
    color: white;
    border-radius: 4px;
    box-shadow: rgba(252, 180, 65, 0.2) 0 10px 20px -3px;
    background-color: #d29505;
    padding: 1rem 2rem;
    display: flex;
    justify-content: space-between;
}
.note-content {
    flex-grow: 1;
    font-size: 0.8em;
}
.note-options {
    display: flex;
}
.note-options button {
    padding: 6px;
    margin-left: 6px;
    background-color: #4285f4;
    color: #fff;
    border: none;
    cursor: pointer;
    border-radius: 3px;
    transition: background-color 0.3s;
    font-size: 0.8em;
}
.note-options button:hover {
    background-color: #357ae8;}
.edit-input {
    width: 100%;
    box-sizing: border-box;
    padding: 6px;
    margin-bottom: 10px;
    font-size: 0.8em;
    border: 1px solid #4285f4;
    border-radius: 3px;}
.edit-buttons {
    display: flex;
    justify-content: flex-end;
}
.edit-buttons button {
    margin-left: 3px;
    background-color: #4285f4;
    color: #fff;
    border: none;
    cursor: pointer;
    border-radius: 3px;
    transition: background-color 0.3s;
    font-size: 0.8em;
}
.edit-buttons button:hover {
    background-color: #357ae8;
}
```

Web application developed by using the Python Flask and HTML after Code Refactoring and Bug Fixing:-



THANK YOU

INNOMATICS
RESEARCH LABS