Final Project	
MyMemento: A Digital Diary	
Course Code: CPE 201L	**Program:** BSCPE
Course Title: Data Structure and Algorithm	**Date Performed:** 10 / 21 / 2025
Section: 2A	**Date Submitted:** 11 / 08 / 2025
Leader: Villanueva, Bryan O. **Members:** Asugas, Kenneth R. Regondola, Jezreel P. Ruperto, April Anne A, Villacin, Justine R.	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s):

This project aims to implement the following:

- To develop a diary application in Python where users can organize their personal notes in a simple and structured way.
- To implement a **Singly Linked List (SLL)** as the main data structure for managing diary entries.
- To create an interface that is user-friendly and visually similar to modern note-taking applications.
- To apply operations such as adding, displaying, editing, and deleting diary entries using an SLL for efficient data handling.
- To demonstrate how data structures can be applied in building useful and functional real-world programs.
- To enhance problem-solving, teamwork, and programming skills through collaborative development.

2. Intended Learning Outcomes (ILOs):

After completing the project, the students are expected to:

- Understand the concept and application of a **Singly Linked List** in software development.
- Apply data structure and algorithm concepts in building a functional application.
- Utilize Python and Flask to design and integrate the backend with the front-end interface.
- Strengthen collaboration, analytical thinking, and documentation skills through project implementation.

3. Discussion:

In this project, we developed **MyMemento**, a diary application in Python that allows users to record and organize personal notes easily. A **Singly Linked List (SLL)** was used as the main data structure, where each entry is represented as a node containing the diary content and a link to the next entry. This structure enables efficient traversal and management of entries for operations such as adding, editing, viewing, and deleting. The system was built using **Flask**, **HTML**, **CSS**, and **JavaScript**, showing how data structures can be applied effectively in practical applications.

4. Materials and Equipments:

- **PC/Laptop**
- **Visual Studio Code**
- **Python (Flask Framework)**
- **JSON File**
- **Web Browser**

5. Procedure:

1. **Planning and Design** – The group planned the layout and determined how the SLL would handle diary entries.
2. **Interface Development** – Created the web interface using HTML and CSS for structure and design.
3. **Backend Implementation** – Developed Flask routes and implemented the SLL in Python for data management.
4. **Integration** – Connected the front-end interface with the backend for smooth data flow.
5. **Testing and Debugging** – Ensured that SLL operations worked properly and fixed encountered errors.
6. **Finalization** – Polished the interface and confirmed the system's functionality before submission.

7. **Source Codes:**

```
# LINKED LIST IMPLEMENTATION
class Node:
    """Node for linked list storing journal notes"""
    def __init__(self, note_data):
        self.data = note_data
        self.next = None

class NotesLinkedList:
    """Linked list to store journal notes"""
    def __init__(self):
        self.head = None
        self.size = 0

    def add_note(self, note_data):
        """Add a new note to the beginning (most recent first)"""
        new_node = Node(note_data)
        new_node.next = self.head
        self.head = new_node
        self.size += 1
        return new_node

    def delete_note(self, note_id):
        """Delete a note by ID"""
        if self.head is None:
            return False

        if self.head.data['id'] == note_id:
            self.head = self.head.next
            self.size -= 1
            return True

        current = self.head
        while current.next:
            if current.next.data['id'] == note_id:
                current.next = current.next.next
                self.size -= 1
                return True
            current = current.next

        return False

    def get_notes_by_date(self, target_date):
        """Get all notes for a specific date (YYYY-MM-DD)"""
        notes = []
        current = self.head

        while current:
```

```
            note_date = current.data['date'].split('T')[0]
            if note_date == target_date:
                notes.append(current.data)
            current = current.next

        return notes

    def get_all_notes(self):
        """Get all notes in the list"""
        notes = []
        current = self.head
        while current:
            notes.append(current.data)
            current = current.next
        return notes

    def get_notes_with_photos(self):
        """Get all photos from notes"""
        photos = []
        current = self.head

        while current:
            if current.data.get('photos'):
                for photo_data in current.data['photos']:
                    photos.append({
                        'src': photo_data,
                        'mood': current.data['mood'],
                        'date': current.data['date'],
                        'text': current.data['text'],
                        'note_id': current.data['id'],
                        'deleted': current.data.get('deleted', False)
                    })
            current = current.next

        return photos

    def get_deleted_notes(self):
        """Get all deleted notes"""
        deleted = []
        current = self.head
        while current:
            if current.data.get('deleted'):
                deleted.append(current.data)
            current = current.next
        return deleted

    def restore_note(self, note_id):
        """Restore a deleted note"""
        current = self.head
```

```
        while current:
            if current.data['id'] == note_id and current.data.get('deleted'):
                current.data['deleted'] = False
                return True
            current = current.next
        return False

# JOURNAL APP BACKEND
import json
import os
from datetime import datetime, timedelta
from flask import Flask, request, jsonify, send_from_directory
from flask_cors import CORS

class JournalAppBackend:
    def __init__(self):
        self.app = Flask(__name__, static_folder='static')
        CORS(self.app)
        self.notes_list = NotesLinkedList()
        self.data_file = 'journal_data.json'

        self.load_data()
        self.setup_routes()

    def load_data(self):
        """Load notes from JSON file"""
        if os.path.exists(self.data_file):
            try:
                with open(self.data_file, 'r', encoding='utf-8') as f:
                    notes_data = json.load(f)
                    for note in reversed(notes_data):
                        self.notes_list.add_note(note)
                print(f"✓ Loaded {self.notes_list.size} notes")
            except Exception as e:
                print(f"✗ Error loading data: {e}")

    def save_data(self):
        """Save notes to JSON file"""
        try:
            all_notes = self.notes_list.get_all_notes()
            with open(self.data_file, 'w', encoding='utf-8') as f:
                json.dump(all_notes, f, indent=2, ensure_ascii=False)
        except Exception as e:
            print(f"✗ Error saving: {e}")

    def setup_routes(self):
        """Setup Flask routes"""

        @self.app.route('/')
```

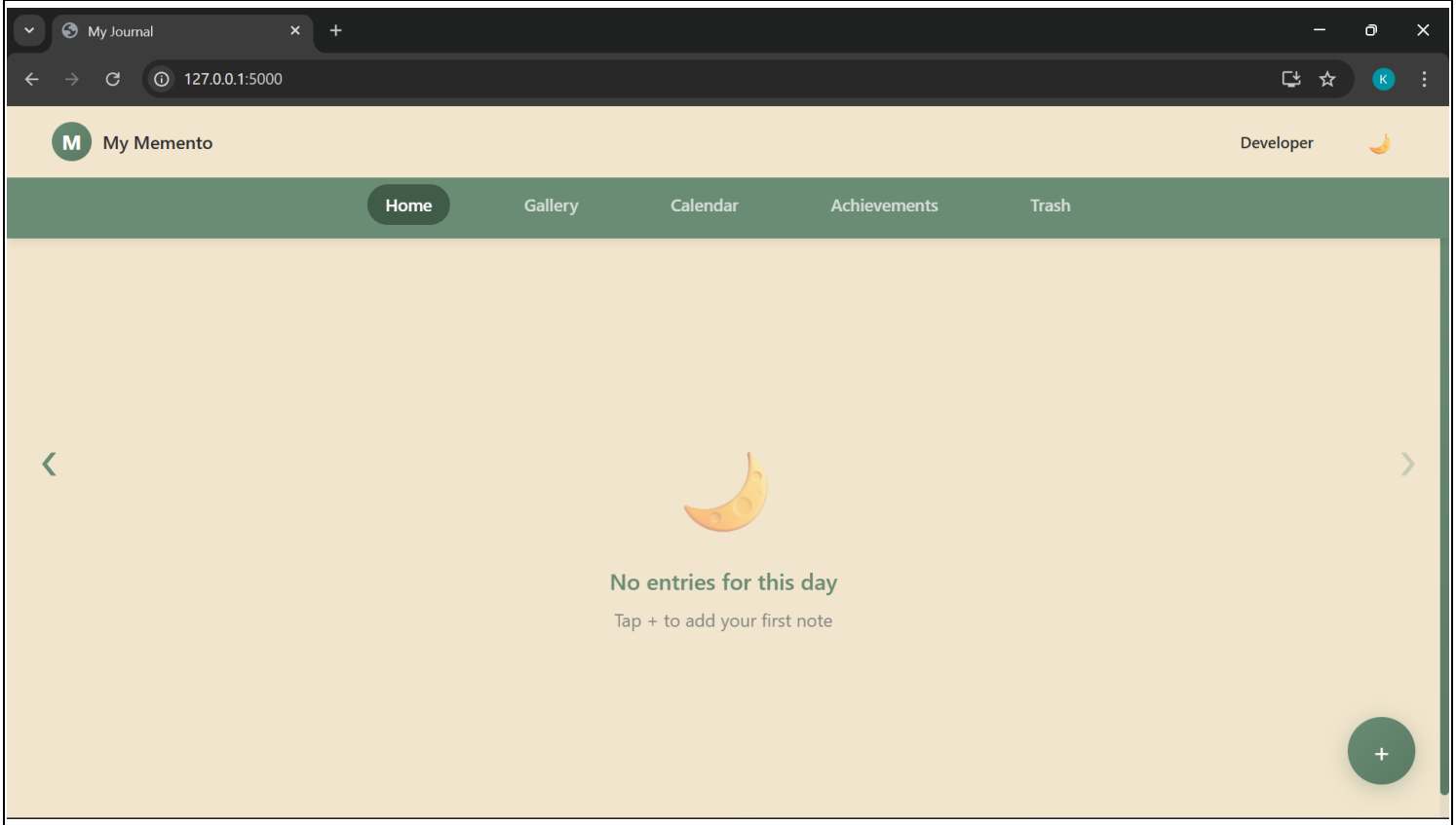
```
        def serve_index():
            return send_from_directory('.', 'index.html')

        @self.app.route('/<path:path>')
        def serve_static(path):
            return send_from_directory('.', path)

        @self.app.route('/api/notes', methods=['GET'])
        def get_notes():
            """Get notes with optional date filter"""
            date_filter = request.args.get('date')

            if date_filter:
                notes = self.notes_list.get_notes_by_date(date_filter)
            else:
                notes = self.notes_list.get_all_notes()

            return jsonify(notes)

        @self.app.route('/api/notes', methods=['POST'])
        def add_note():
            """Add a new note"""
            try:
                note_data = request.get_json()

                if not note_data.get('text') or not note_data.get('mood'):
                    return jsonify({'error': 'Text and mood required'}), 400

                if not note_data.get('date'):
                    note_data['date'] = datetime.now().isoformat()

                if not note_data.get('id'):
                    note_data['id'] = int(datetime.now().timestamp() * 1000)

                if 'photos' not in note_data:
                    note_data['photos'] = []

                if 'title' not in note_data:
                    note_data['title'] = ''

                if 'deleted' not in note_data:
                    note_data['deleted'] = False

                self.notes_list.add_note(note_data)
                self.save_data()

                return jsonify({'message': 'Success', 'note': note_data}), 201

            except Exception as e:
```

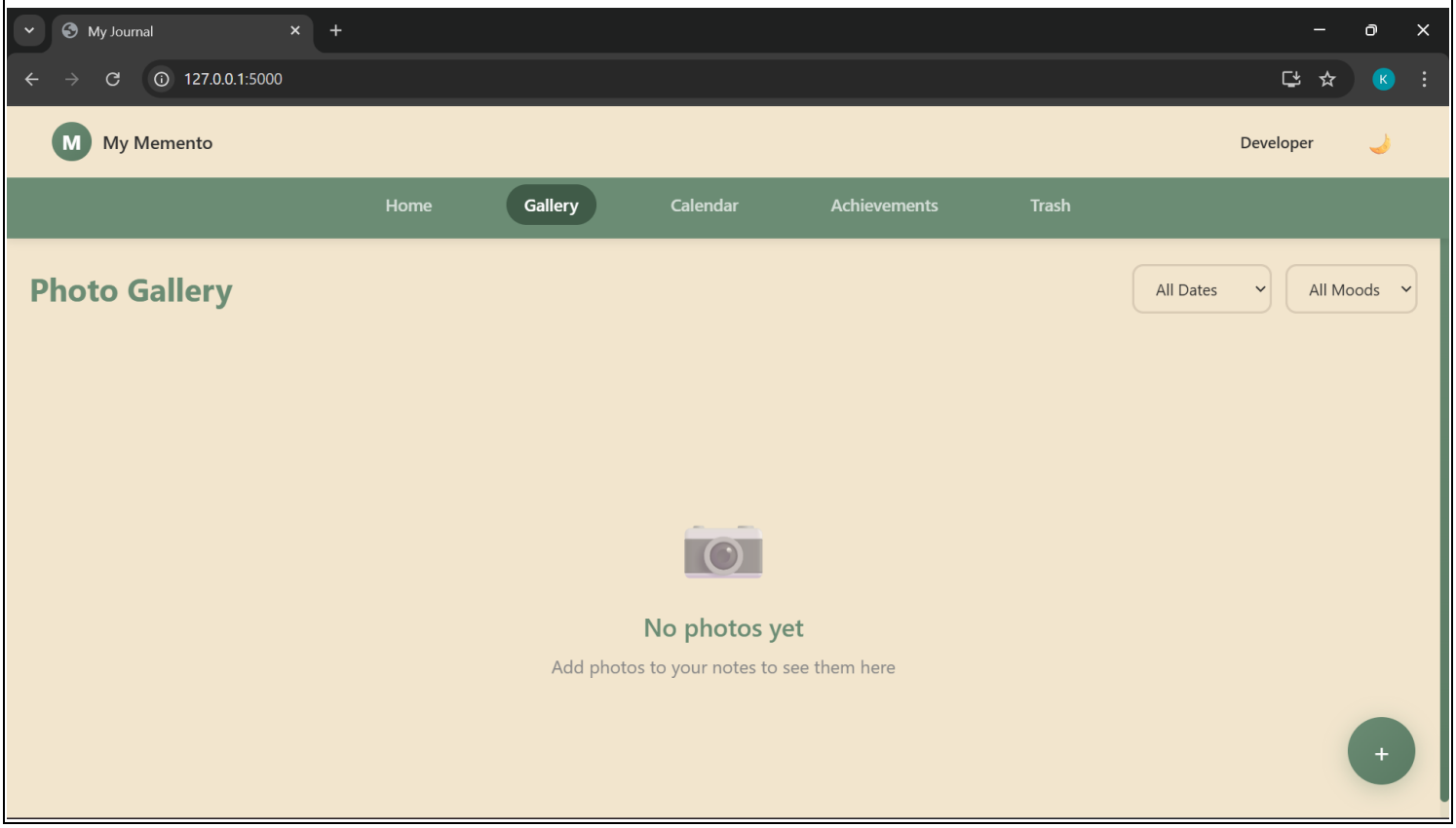
```
            return jsonify({'error': str(e)}), 500

        @self.app.route('/api/notes/<int:note_id>', methods=['DELETE'])
        def delete_note(note_id):
            """Soft delete a note (move to trash)"""
            try:
                current = self.notes_list.head
                while current:
                    if current.data['id'] == note_id:
                        current.data['deleted'] = True
                        self.save_data()
                        return jsonify({'message': 'Moved to trash'}), 200
                    current = current.next
                return jsonify({'error': 'Not found'}), 404
            except Exception as e:
                return jsonify({'error': str(e)}), 500

        @self.app.route('/api/photos', methods=['GET'])
        def get_photos():
            """Get photos with filters"""
            date_filter = request.args.get('date_filter', 'all')
            mood_filter = request.args.get('mood_filter', 'all')

            all_photos = self.notes_list.get_notes_with_photos()
            filtered = []

            today = datetime.now().date()

            for photo in all_photos:
                # Skip deleted photos
                if photo.get('deleted'):
                    continue

                photo_date = photo['date'].split('T')[0]

                # Date filter
                if date_filter == 'today' and photo_date != today.isoformat():
                    continue
                elif date_filter == 'week':
                    week_ago = today - timedelta(days=7)
                    if photo_date < week_ago.isoformat():
                        continue
                elif date_filter == 'month':
                    photo_dt = datetime.fromisoformat(photo_date)
                    if photo_dt.month != today.month or photo_dt.year != today.year:
                        continue

                # Mood filter
                if mood_filter != 'all' and photo['mood'] != mood_filter:
```

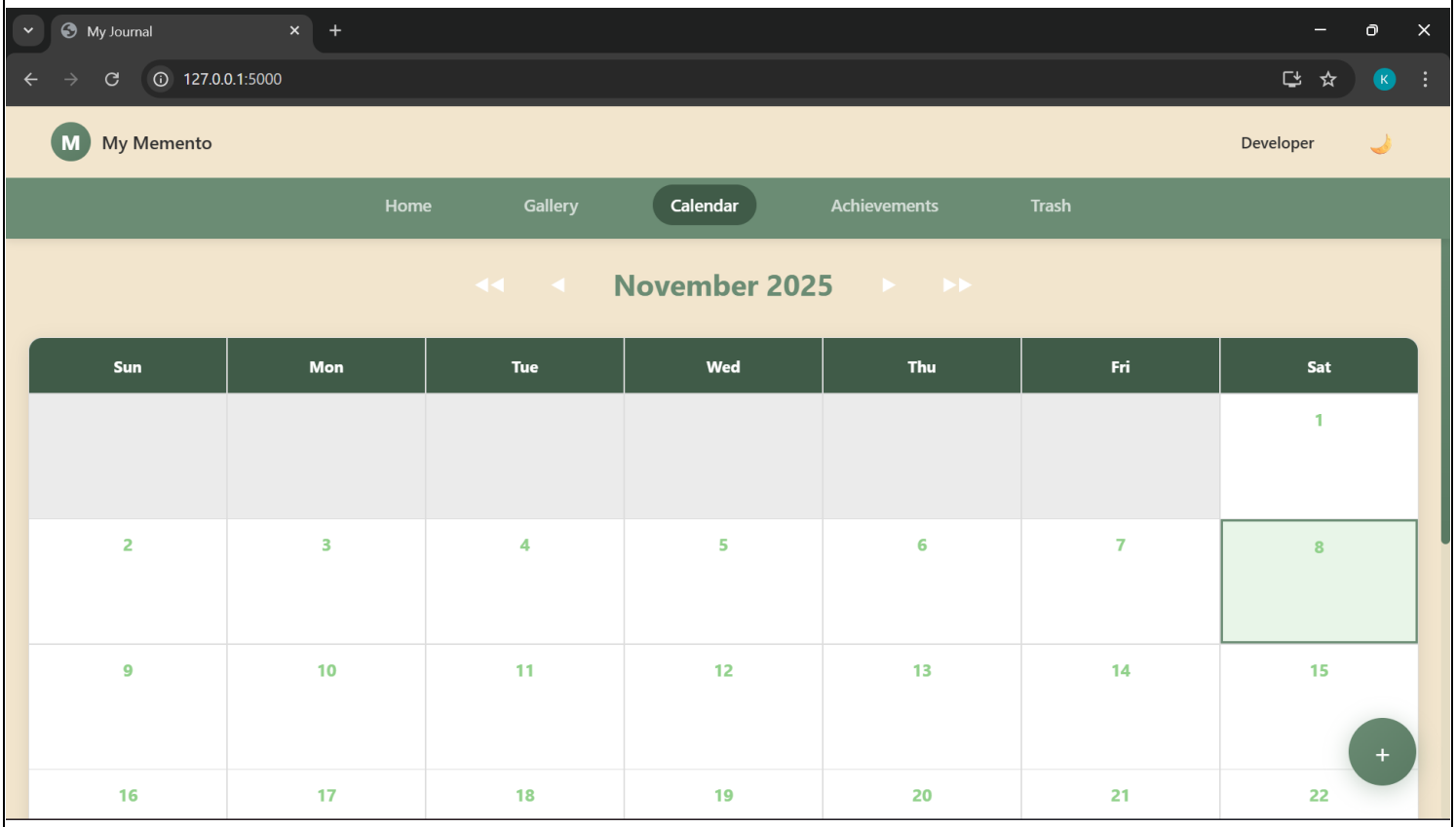
```
                continue

            filtered.append(photo)

        return jsonify(filtered)

    @self.app.route('/api/calendar/<int:year>/<int:month>', methods=['GET'])
    def get_calendar_data(year, month):
        """Get calendar data for a month"""
        try:
            start_date = datetime(year, month, 1).date()
            if month == 12:
                end_date = datetime(year + 1, 1, 1).date() - timedelta(days=1)
            else:
                end_date = datetime(year, month + 1, 1).date() - timedelta(days=1)

            calendar_data = {}
            current = self.notes_list.head

            while current:
                # Skip deleted notes
                if current.data.get('deleted'):
                    current = current.next
                    continue

                note_date = current.data['date'].split('T')[0]
                note_dt = datetime.fromisoformat(note_date).date()

                if start_date <= note_dt <= end_date:
                    if note_date not in calendar_data:
                        calendar_data[note_date] = {'notes': []}
                    calendar_data[note_date]['notes'].append(current.data)

                current = current.next

            return jsonify(calendar_data)

        except Exception as e:
            return jsonify({'error': str(e)}), 500

    @self.app.route('/api/trash', methods=['GET'])
    def get_trash():
        """Get all deleted notes"""
        try:
            deleted_notes = self.notes_list.get_deleted_notes()
            return jsonify(deleted_notes)
        except Exception as e:
            return jsonify({'error': str(e)}), 500
```

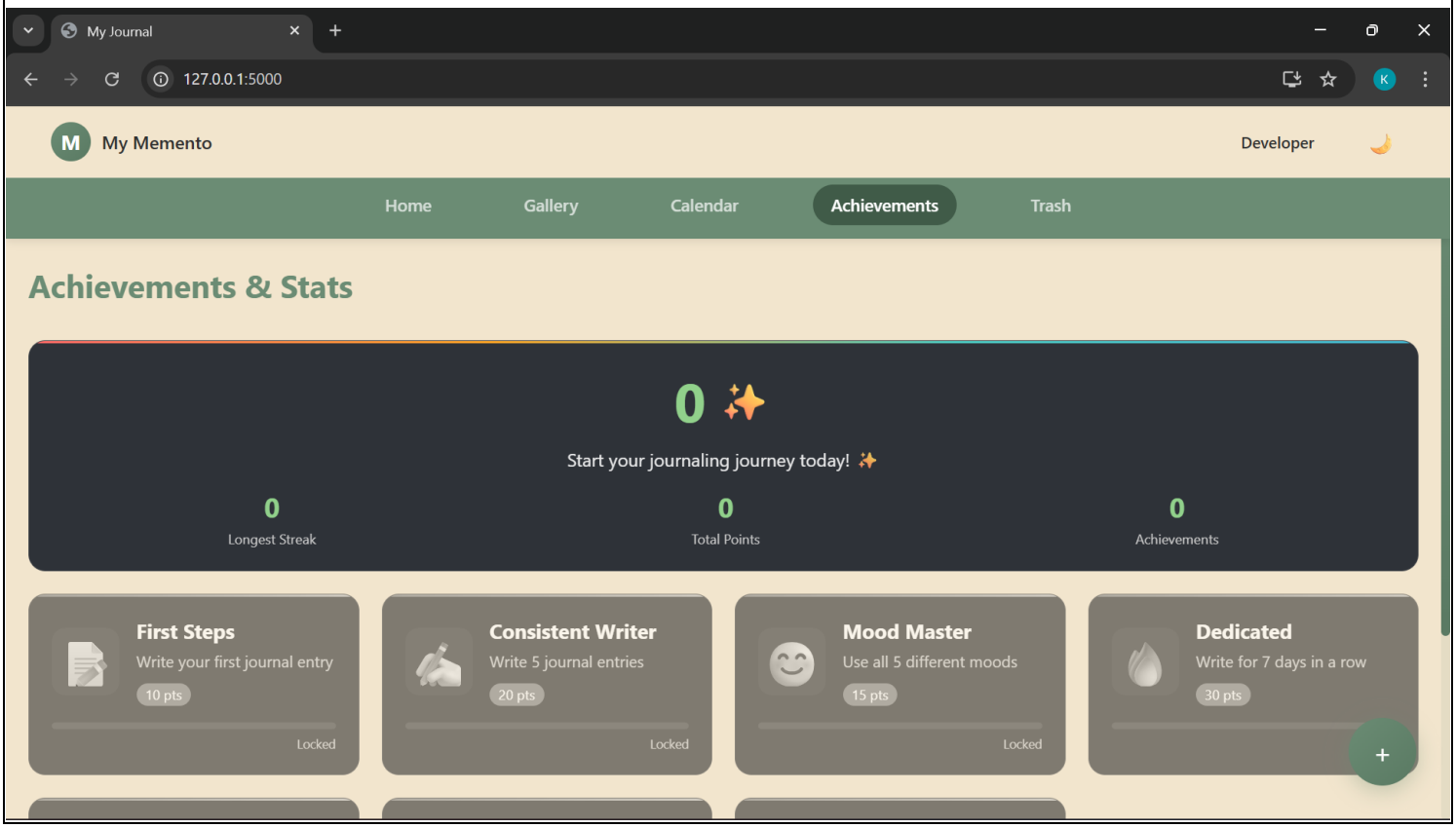
```
        @self.app.route('/api/trash/<int:note_id>/restore', methods=['POST'])
        def restore_note(note_id):
            """Restore a note from trash"""
            try:
                if self.notes_list.restore_note(note_id):
                    self.save_data()
                    return jsonify({'message': 'Note restored'}), 200
                return jsonify({'error': 'Not found in trash'}), 404
            except Exception as e:
                return jsonify({'error': str(e)}), 500

        @self.app.route('/api/trash/<int:note_id>', methods=['DELETE'])
        def permanently_delete(note_id):
            """Permanently delete a note from trash"""
            try:
                if self.notes_list.delete_note(note_id):
                    self.save_data()
                    return jsonify({'message': 'Permanently deleted'}), 200
                return jsonify({'error': 'Not found'}), 404
            except Exception as e:
                return jsonify({'error': str(e)}), 500

    def run(self, host='127.0.0.1', port=5000, debug=True):
        """Run the Flask app"""
        print(f"\n🚀 Journal App running at http://{host}:{port}")
        print(f"📊 {self.notes_list.size} notes loaded\n")
        self.app.run(host=host, port=port, debug=debug)

if __name__ == '__main__':
    backend = JournalAppBackend()
    backend.run()
```

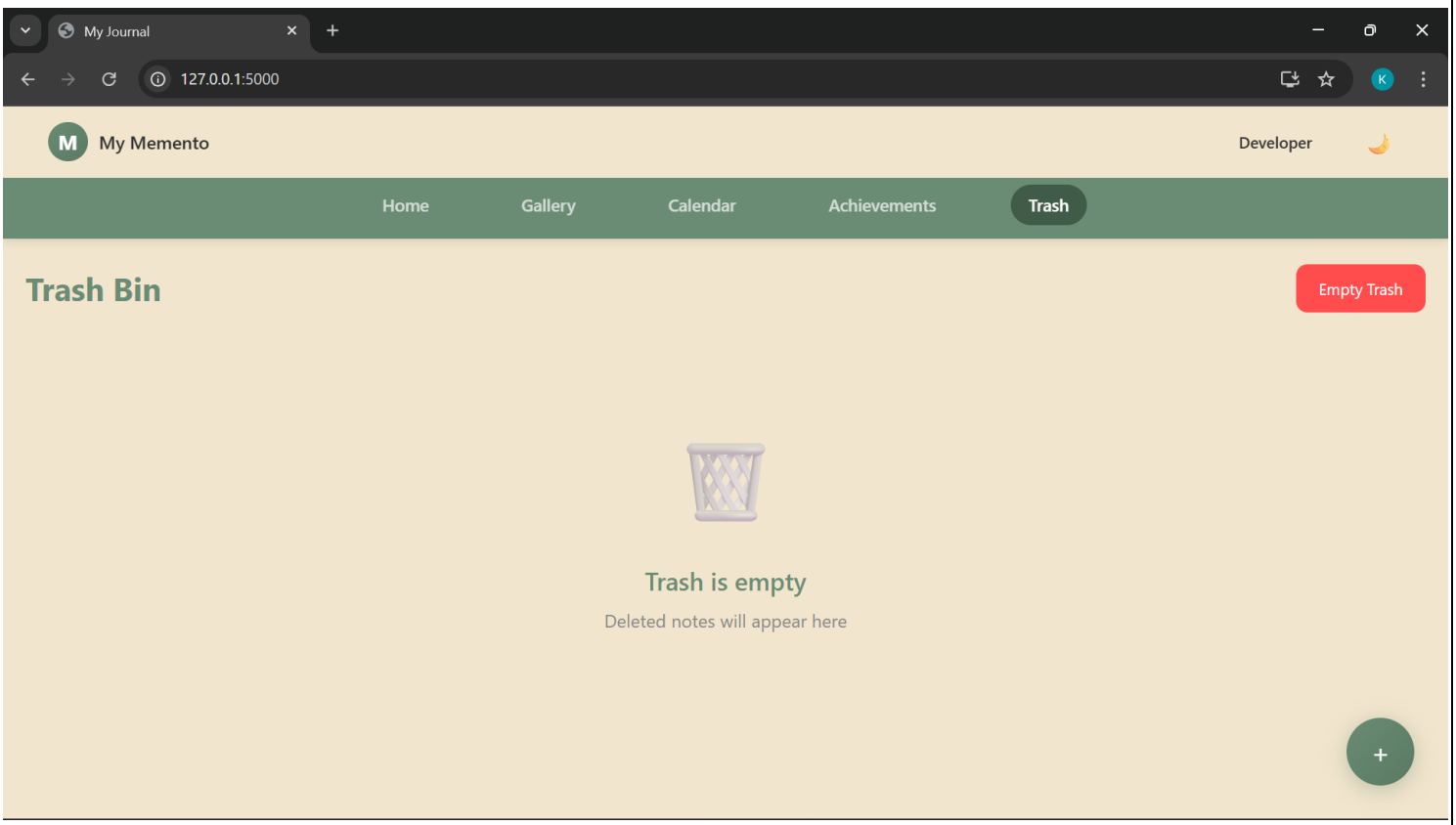
6. Output


My Journal
127.0.0.1:5000
My Memento
Developer
Home
Gallery
Calendar
Achievements
Trash
No entries for this day
Tap + to add your first note


My Journal
127.0.0.1:5000
My Memento
Developer
Home
Gallery
Calendar
Achievements
Trash
Photo Gallery
All Dates
All Moods
No photos yet
Add photos to your notes to see them here


My Journal
127.0.0.1:5000
My Memento
Developer
Home
Gallery
Calendar
Achievements
Trash
November 2025
Sun
Mon
Tue
Wed
Thu
Fri
Sat
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22


My Journal
127.0.0.1:5000
My Memento
Developer
Home
Gallery
Calendar
Achievements
Trash
Achievements & Stats
0
Start your journaling journey today!
0
Longest Streak
0
Total Points
0
Achievements
First Steps
Write your first journal entry
10 pts
Locked
Consistent Writer
Write 5 journal entries
20 pts
Locked
Mood Master
Use all 5 different moods
15 pts
Locked
Dedicated
Write for 7 days in a row
30 pts



The final output is a functional diary application named *MyMemento*. It allows users to add, view, edit, and delete diary entries using a **Singly Linked List** for efficient organization and traversal of data. The program features a simple and user-friendly interface, storing entries in a JSON file for accessibility and reliability.

7. Conclusion:

In conclusion, the group successfully developed **MyMemento**, a diary system that applies the **Singly Linked List (SLL)** as its main data structure. Each diary entry functions as a node containing content and a link to the next node, enabling efficient addition, traversal, and deletion of entries.

The project allowed the group to apply the concepts of data structures and algorithms in a real-world application. It also improved their programming skills and understanding of linked lists while encouraging collaboration and creativity. *MyMemento* demonstrates that even simple structures like the SLL can be effectively used in building practical and interactive applications.

8. References

[1] *Flask Documentation*, “Flask Web Framework,” 2024. [Online]. Available: https://flask.palletsprojects.com/
[2] *Mozilla Developer Network (MDN)*, “HTML, CSS, and JavaScript Guides,” 2024. [Online]. Available: https://developer.mozilla.org/
[3] *W3Schools*, “JSON Tutorial,” 2024. [Online]. Available: https://www.w3schools.com/js/js_json_intro.asp
[4] *GeeksforGeeks*, “Singly Linked List in Python,” 2024. [Online]. Available: https://www.geeksforgeeks.org/linked-list-set-1-introduction/