

***AN INTERNSHIP PROJECT***

***ON***

***“Developing a Python-Based Student Feedback System”***



***An Internship Project carried out of six phrase, submitted in partial***

***fulfillment of the requirements for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**IN**

**EMERGING TECHNOLOGIES IN COMPUTER SCIENCE**

**BY**

Name: D.Poojitha

REGD.NO: 229X1A3304

**DEPARTMENT OF EMERGING TECHNOLOGIES IN COMPUTER  
SCIENCE**

**G.Pullu Reddy Engineering College (Autonomous):**

**Kurnool – 518007, Andhra Pradesh, India**

**(Affiliated to Jawaharlal Nehru Technological University – Anantapur)**

**Internship Project carried out at : Six Phrase, Chennai**

**Internship Project Report**

**Submitted in accordance with the requirement for the degree of**

**BACHELOR OF TECHNOLOGY**

Name of the College : G.PULLA REDDY ENGINEERING COLLEGE

Department : EMERGING TECHNOLOGIES IN COMPUTER SCIENCE

Name of the Mentor : PRAVEEN KUMAR PARADESI

Company Name : Six Phrase, Chennai

Title of the Project : FEEDBACK MANAGEMENT SYSTEM

Name of the Student : D.POOJITHA

Regd Number : 229X1A3304

Program of study : Summer Internship

Date of Submission :26-05-2025

## **Table of Contents**

- Abstract
- Introduction
- Methodology
- Source Code
- Output
- Conclusion
- Future Work
- Reference

## Abstract

The **Feedback Management System** is a desktop-based application developed using Python's tkinter library to streamline the process of collecting, managing, and reviewing feedback in an academic or organizational environment. The system provides a role-based interface for both **Students** and **Administrators**, ensuring secure access and tailored functionality for each user type.

For **students**, the system offers a user-friendly form to submit feedback on specific subjects and Faculty members, including ratings, comments, and responses to compulsory questions. Feedback entries are timestamped and stored persistently in a JSON file for later analysis.

**Administrators** have access to an extended set of tools including the ability to view, delete, and manage feedback entries, add or remove users and receivers, and maintain a list of valid subjects. A tabbed interface allows easy navigation between feedback data, user and subject management panels, and visualization of existing records.

The system also includes dynamic theme switching to improve usability and personalization. It ensures data persistence through JSON-based storage and emphasizes modularity with the use of separate classes for feedback, subjects, and user management.

Overall, this project demonstrates core concepts of GUI programming, file handling, role-based Access control, and user experience design using Python and Tkinter, making it a practical and extensible solution for educational institutions or training environments seeking a lightweight feedback platform.

## Introduction

The **Feedback Management System** is a Python-based application built using the **Tkinter GUI toolkit**, designed to streamline and digitize the process of collecting, managing, and analyzing feedback from students or users. This system supports two types of users: **Admins** and **Students**, each with distinct roles and interfaces tailored to their functionalities.

### Purpose

The primary goal of this application is to:

- Collect structured feedback from students.
- Allow administrators to manage feedback records, users, receivers, and subjects.
- Provide a user-friendly interface for both data entry and administrative operations.

### Key Features

#### 1. Login System:

- Authenticates users as either Admin or Student.
- Redirects them to role-specific dashboards upon successful login.

#### 2. Student Panel:

- Submit feedback by entering name, selecting subject, giving a rating, and commenting.
- Mandatory yes/no questions to ensure basic quality checks on sessions.

#### 3. Admin Panel:

- View all submitted feedback in a sortable and deletable table.
- Add or remove **Users** and **Receivers** dynamically.
- Manage **Subjects** offered for feedback collection.
- View a list of all current subjects.

#### 4. Theme Selection:

- Users can select from various available themes to personalize their interface experience.

#### 5. Persistent Data Storage:

- Feedback and subject data are stored locally in **JSON files** (feedbacks.json and subjects.json), ensuring persistence across sessions.

### Modular Code Design

- **FeedbackManager**: Handles feedback storage, retrieval, addition, and deletion.
- **SubjectManager**: Manages the list of subjects that students can choose from.
- **UserManager**: Maintains lists of users and receivers, allowing Admins to modify them.
- **FeedbackApp**: The main GUI application that binds all components based on the user's role.

## METHODOLOGY

### Cell 1: Imports and Setup

- **Imports:**
  - tkinter, ttk, messagebox – Used for building the GUI.
  - datetime – Adds timestamps to feedback submissions.
  - json, os – Handles file operations and JSON data storage.

### Cell 2: FeedbackManager Class

- **Purpose:** Manages feedback records stored in feedbacks.json.
- **Functions:**
  - load\_feedbacks() – Reads feedback entries from the JSON file.
  - save\_feedbacks(feedbacks) – Saves the provided list of feedbacks to file.
  - add\_feedback(name, rating, comment, subject, compulsory\_answers) – Creates and stores a new feedback entry with timestamp.
  - delete\_feedback(date, name, rating, comment) – Deletes a feedback based on key identifying fields.

### Cell 3: SubjectManager Class

- **Purpose:** Manages the list of subjects in subjects.json.
- **Functions:**
  - load\_subjects() – Loads the list of available subjects.
  - save\_subjects(subjects) – Persists subject list to file.
  - add\_subject(subject) – Adds a subject if it's not already present.

### Cell 4: UserManager Class

**Purpose:** Manages in-memory lists of users and receivers.

- **Functions:**

- modify\_users(choice, name) – Adds ('1') or removes ('0') users.
- modify\_receivers(choice, name) – Adds ('1') or removes ('0') receivers.

## **Cell 5: FeedbackApp Class**

- **Initialization:**

- Configures the main window, initializes manager instances, theme dropdown, tabs, and logout button.

- **Theme Handling:**

- Theme can be changed using a dropdown menu.

- **Tabs Setup:**

- Student Role: Only the “Submit Feedback” tab is shown.
- Admin Role: Includes tabs for viewing feedback, managing users, receivers, and subjects.

- **Feedback Submission Tab:**

- Inputs:
  - Name (Entry), Subject (Dropdown), Rating (Entry), Comment (Entry)
  - Two Yes/No checkboxes for compulsory questions
- Button:
  - Submit Feedback – Validates and submits data

- **View Feedback Tab (Admin):**

- Displays feedback in a Treeview.
- Buttons:
  - Load Feedbacks – Loads data from storage
  - Delete Selected – Deletes selected row

- **User/Receiver Management Tabs (Admin):**
  - Adds and removes users/receivers through entry field and buttons.
- **Subject Management Tabs (Admin):**
  - Add Subject: Text field and add button
  - View Subjects: Shows all added subjects in a text label
- **Submit Feedback Logic:**
  - Validates required fields, rating, and checkboxes before saving.
- **Load/Delete Feedback Logic:**
  - Loads feedback into Treeview
  - Deletes selected entry based on unique fields

#### **Cell 6: Login Functions**

- **login\_action()**
  - Authenticates based on role and credentials.
  - Accepts only “admin” and “student” roles.
- **show\_login\_dialog()**
  - Displays a login form with username, password, and role selection (radio buttons).

#### **Cell 7: Start the App**

- **Function Call:**
  - show\_login\_dialog() is invoked initially to begin the application



## Source Code

### # Cell 1: Imports and Setup

```
import tkinter as tk
from tkinter import ttk, messagebox
from datetime import datetime
import json
import os
```

### # Cell 2: FeedbackManager Class

```
class FeedbackManager:
    def __init__(self):
        self.file_path = "feedbacks.json"
    def load_feedbacks(self):
        if not os.path.exists(self.file_path):
            return []
        try:
            with open(self.file_path, "r") as f:
                return json.load(f)
        except json.JSONDecodeError:
            return []
    def save_feedbacks(self, feedbacks):
        with open(self.file_path, "w") as f:
            json.dump(feedbacks, f, indent=4)
    def add_feedback(self, name, rating, comment, subject, compulsory_answers):
        date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        feedback = {
            "date": date, "name": name, "rating": rating,
            "comment": comment, "subject": subject,
            "answers": compulsory_answers
        }
        feedbacks = self.load_feedbacks()
        feedbacks.append(feedback)
        self.save_feedbacks(feedbacks)
    def delete_feedback(self, date, name, rating, comment):
        feedbacks = self.load_feedbacks()
        new_feedbacks = [fb for fb in feedbacks if not (
            fb['date'] == date and fb['name'] == name and
```

```
        fb['rating'] == rating and fb['comment'] == comment
    )]
    self.save_feedbacks(new_feedbacks)
```

### **# Cell 3: SubjectManager Class**

```
class SubjectManager:
    def __init__(self):
        self.file_path = "subjects.json"
    def load_subjects(self):
        if not os.path.exists(self.file_path):
            return []
        try:
            with open(self.file_path, "r") as f:
                return json.load(f)
        except json.JSONDecodeError:
            return []
    def save_subjects(self, subjects):
        with open(self.file_path, "w") as f:
            json.dump(subjects, f, indent=4)
    def add_subject(self, subject):
        subjects = self.load_subjects()
        if subject not in subjects:
            subjects.append(subject)
            self.save_subjects(subjects)
```

### **# Cell 4: UserManager Class**

```
class UserManager:
    def __init__(self):
        self.users = []
        self.receivers = []
    def modify_users(self, choice, name):
        if choice == '1':
            if name and name not in self.users:
                self.users.append(name)
                print(f"User '{name}' added.")
            else:
                print(f"User '{name}' already exists or invalid.")
```

```

elif choice == '0':
    if name in self.users:
        self.users.remove(name)
        print(f"User '{name}' removed.")
    else:
        print(f"User '{name}' not found.")
def modify_receivers(self, choice, name):
    if choice == '1':
        if name and name not in self.receivers:
            self.receivers.append(name)
            print(f"Receiver '{name}' added.")
        else:
            print(f"Receiver '{name}' already exists or invalid.")
    elif choice == '0':
        if name in self.receivers:
            self.receivers.remove(name)
            print(f"Receiver '{name}' removed.")
        else:
            print(f"Receiver '{name}' not found.")

```

### # Cell 5: FeedbackApp Class

```

class FeedbackApp:
    def __init__(self, root, role):
        self.root = root
        self.root.title("Feedback Management System")
        self.root.geometry("800x600")
        self.root.configure(padx=10, pady=10)
        self.role = role
        self.feedback_mgr = FeedbackManager()
        self.subject_mgr = SubjectManager()
        self.user_mgr = UserManager()
        self.style = ttk.Style()
        self.available_themes = self.style.theme_names()
        self.current_theme = tk.StringVar(value=self.style.theme_use())
        theme_frame = ttk.Frame(root)
        theme_frame.pack(fill='x', pady=5)
        ttk.Label(theme_frame, text="Select Theme:").pack(side='left', padx=5)

```

```

        self.theme_menu = ttk.OptionMenu(
            theme_frame, self.current_theme,
            self.style.theme_use(),
            *self.available_themes,
            command=self.change_theme
        )
        self.theme_menu.pack(side='left')
        self.tab_control = ttk.Notebook(root)
        self.setup_tabs()
        self.tab_control.pack(expand=1, fill="both", padx=5, pady=5)
        logout_frame = ttk.Frame(root)
        logout_frame.pack(fill='x', side='bottom')
        ttk.Button(logout_frame, text="Logout", command=self.logout).pack(side='right', padx=10,
pady=10)
def logout(self):
    self.root.destroy()
    show_login_dialog()
def change_theme(self, selected_theme):
    try:
        self.style.theme_use(selected_theme)
    except tk.TclError:
        messagebox.showerror("Theme Error", f"Theme '{selected_theme}' is not available.")
def setup_tabs(self):
    if self.role == "Student":
        self.create_feedback_tab()
    else:
        self.create_view_tab()
        self.create_user_tab()
        self.create_receiver_tab()
        self.create_subject_tab()
        self.create_subject_view_tab()
def create_feedback_tab(self):
    tab = ttk.Frame(self.tab_control)
    self.tab_control.add(tab, text='Submit Feedback')
    form_frame = ttk.Frame(tab, padding=20)
    form_frame.pack(anchor='center')

```

```

ttk.Label(form_frame, text="Name:").grid(row=0, column=0, sticky="e", padx=5, pady=5)
self.name_entry = ttk.Entry(form_frame, width=40)
self.name_entry.grid(row=0, column=1, pady=5)
ttk.Label(form_frame, text="Subject:").grid(row=1, column=0, sticky="e", padx=5, pady=5)
self.subject_var = tk.StringVar()
self.subject_menu = ttk.Combobox(form_frame, textvariable=self.subject_var,
values=self.subject_mgr.load_subjects())
self.subject_menu.grid(row=1, column=1, pady=5)
ttk.Label(form_frame, text="Rating (0-100):").grid(row=2, column=0, sticky="e", padx=5,
pady=5)
self.rating_entry = ttk.Entry(form_frame, width=40)
self.rating_entry.grid(row=2, column=1, pady=5)
ttk.Label(form_frame, text="Comment:").grid(row=3, column=0, sticky="e", padx=5, pady=5)
self.comment_entry = ttk.Entry(form_frame, width=40)
self.comment_entry.grid(row=3, column=1, pady=5)
ttk.Label(form_frame, text="Did the faculty arrive on time?").grid(row=4, column=0,
columnspan=2, sticky='w')
self.q1 = tk.BooleanVar()
ttk.Checkbutton(form_frame, variable=self.q1).grid(row=4, column=1, sticky='e')
ttk.Label(form_frame, text="Was the session interactive?").grid(row=5, column=0,
columnspan=2, sticky='w')
self.q2 = tk.BooleanVar()
ttk.Checkbutton(form_frame, variable=self.q2).grid(row=5, column=1, sticky='e')
ttk.Button(form_frame, text="Submit Feedback", command=self.submit_feedback).grid(row=6,
column=0, columnspan=2, pady=15)
def create_view_tab(self):
    tab = ttk.Frame(self.tab_control)
    self.tab_control.add(tab, text='View Feedbacks')
    top_frame = ttk.Frame(tab, padding=10)
    top_frame.pack(fill='x')
    ttk.Button(top_frame, text="Load Feedbacks", command=self.load_feedbacks).pack(side='left',
padx=5)
    self.tree = ttk.Treeview(tab, columns=("Date", "Name", "Rating", "Comment", "Subject"),
show='headings', height=15)
    for col in self.tree['columns']:
        self.tree.heading(col, text=col)

```

```

        self.tree.column(col, anchor="center")
        self.tree.pack(expand=True, fill='both', padx=10, pady=10)
        ttk.Button(tab, text="Delete Selected", command=self.delete_feedback).pack(pady=10)
def create_user_tab(self):
    tab = ttk.Frame(self.tab_control)
    self.tab_control.add(tab, text='Manage Users')
    self.manage_entity(tab, "User", self.user_mgr.modify_users)
def create_receiver_tab(self):
    tab = ttk.Frame(self.tab_control)
    self.tab_control.add(tab, text='Manage Receivers')
    self.manage_entity(tab, "Receiver", self.user_mgr.modify_receivers)
def create_subject_tab(self):
    tab = ttk.Frame(self.tab_control)
    self.tab_control.add(tab, text='Manage Subjects')
    frame = ttk.Frame(tab, padding=20)
    frame.pack()
    ttk.Label(frame, text="Subject Name:").grid(row=0, column=0, padx=5, pady=5)
    self.subject_entry = ttk.Entry(frame, width=30)
    self.subject_entry.grid(row=0, column=1, padx=5, pady=5)
    ttk.Button(frame, text="Add Subject", command=self.add_subject).grid(row=1, column=0,
        colspan=2, pady=10)
def create_subject_view_tab(self):
    tab = ttk.Frame(self.tab_control)
    self.tab_control.add(tab, text='View Subjects')
    subjects = self.subject_mgr.load_subjects()
    text = "\n".join(subjects) if subjects else "No subjects added."
    ttk.Label(tab, text=text, padding=20).pack()
def manage_entity(self, tab, role, modify_func):
    frame = ttk.Frame(tab, padding=20)
    frame.pack()
    ttk.Label(frame, text=f"{role} Name:").grid(row=0, column=0, padx=5, pady=5)
    name_entry = ttk.Entry(frame, width=30)
    name_entry.grid(row=0, column=1, padx=5, pady=5)
def add_entity():
    name = name_entry.get().strip()
    if name:

```

```

        modify_func('1', name)
        messagebox.showinfo(f"{role} Added", f"{role} '{name}' added.")
        name_entry.delete(0, tk.END)
    else:
        messagebox.showerror("Error", f"Please enter {role.lower()} name.")
def remove_entity():
    name = name_entry.get().strip()
    if name:
        modify_func('0', name)
        messagebox.showinfo(f"{role} Removed", f"{role} '{name}' removed.")
        name_entry.delete(0, tk.END)
    else:
        messagebox.showerror("Error", f"Please enter {role.lower()} name.")
    ttk.Button(frame, text=f"Add {role}", command=add_entity).grid(row=1, column=0,
        pady=10)
    ttk.Button(frame, text=f"Remove {role}", command=remove_entity).grid(row=1,
        column=1, pady=10)
def add_subject(self):
    subject = self.subject_entry.get().strip()
    if subject:
        self.subject_mgr.add_subject(subject)
        messagebox.showinfo("Subject Added", f"Subject '{subject}' added.")
        self.subject_entry.delete(0, tk.END)
    if hasattr(self, 'subject_menu'):
        self.subject_menu['values'] = self.subject_mgr.load_subjects()
    else:
        messagebox.showerror("Error", "Please enter a subject name.")
def submit_feedback(self):
    name = self.name_entry.get().strip()
    subject = self.subject_var.get().strip()
    rating = self.rating_entry.get().strip()
    comment = self.comment_entry.get().strip()
    q1 = self.q1.get()
    q2 = self.q2.get()
    if not name or not subject or not rating or not comment:
        messagebox.showerror("Error", "All fields including subject must be filled.") return

```

```

try:
    rating_val = int(rating)
    if rating_val < 0 or rating_val > 100:
        raise ValueError
except ValueError:
    messagebox.showerror("Error", "Rating must be an integer between 0 and 100.")
    return
if not q1 or not q2:
    messagebox.showerror("Error", "Please answer all compulsory questions.")
    return
answers = {
    "faculty_arrived_on_time": q1,
    "session_interactive": q2
}
self.feedback_mgr.add_feedback(name, rating_val, comment, subject, answers)
messagebox.showinfo("Feedback Submitted", "Thank you for your feedback!")

self.name_entry.delete(0, tk.END)
self.rating_entry.delete(0, tk.END)
self.comment_entry.delete(0, tk.END)
self.q1.set(False)
self.q2.set(False)
self.subject_var.set("")
def load_feedbacks(self):
    for item in self.tree.get_children():
        self.tree.delete(item)
    feedbacks = self.feedback_mgr.load_feedbacks()
    if not feedbacks:
        messagebox.showinfo("No Feedback", "No feedback entries found.")
        return
    for fb in feedbacks:
        date = fb.get('date', "")
        name = fb.get('name', "")
        rating = fb.get('rating', "")
        comment = fb.get('comment', "")
        subject = fb.get('subject', "")

```



```

        self.tree.insert("", 'end', values=(date, name, rating, comment, subject))
def delete_feedback(self):
    selected = self.tree.selection()
    if not selected:
        messagebox.showerror("Error", "Please select a feedback to delete.")
        return
    confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete the
selected feedback?")
    if not confirm:
        return
    for item in selected:
        values = self.tree.item(item, 'values')
        date, name, rating, comment, _ = values
        self.feedback_mgr.delete_feedback(date, name, rating, comment)
        self.tree.delete(item)

```

### **# Cell 6: Login Functions**

```

def login_action():
    username = username_entry.get().strip()
    password = password_entry.get().strip()
    role = role_var.get()
    if (username == "admin" and password == "admin" and role == "Admin") or \
        (username == "student" and password == "student" and role == "Student"):
        login_root.destroy()
        root = tk.Tk()
        app = FeedbackApp(root, role)
        root.mainloop()
    else:
        messagebox.showerror("Invalid Credentials", "Username, password or role is incorrect.")
def show_login_dialog():
    global login_root, username_entry, password_entry, role_var
    login_root = tk.Tk()
    login_root.title("Login")
    login_root.geometry("300x220")
    login_root.configure(padx=20, pady=20)
    ttk.Label(login_root, text="Username:").pack(anchor='w', pady=5)
    username_entry = ttk.Entry(login_root)

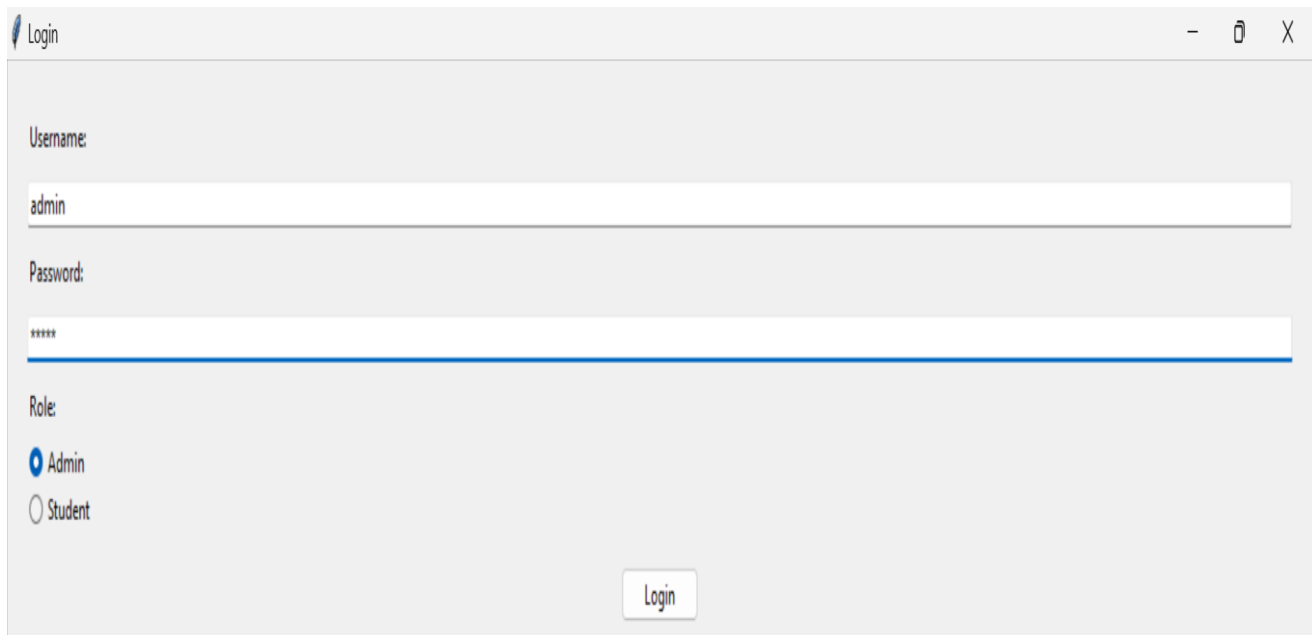
```

```
username_entry.pack(fill='x', pady=5)
ttk.Label(login_root, text="Password:").pack(anchor='w', pady=5)
password_entry = ttk.Entry(login_root, show="*")
password_entry.pack(fill='x', pady=5)
ttk.Label(login_root, text="Role:").pack(anchor='w', pady=5)
role_var = tk.StringVar(value="Admin")
ttk.Radiobutton(login_root, text="Admin", variable=role_var, value="Admin").pack(anchor='w')
ttk.Radiobutton(login_root, text="Student", variable=role_var, value="Student").pack(anchor='w')
ttk.Button(login_root, text="Login", command=login_action).pack(pady=15)
login_root.mainloop()
```

### **# Cell 7: Start the App**

```
show_login_dialog()
```

## OUTPUT



A screenshot of a web application's login window. The window has a title bar with a feather icon and the text 'Login'. It contains three input fields: 'Username:' with 'admin' entered, 'Password:' with '\*\*\*\*\*' entered, and 'Role:' with radio buttons for 'Admin' (selected) and 'Student'. A 'Login' button is at the bottom right.

Username:

admin

Password:

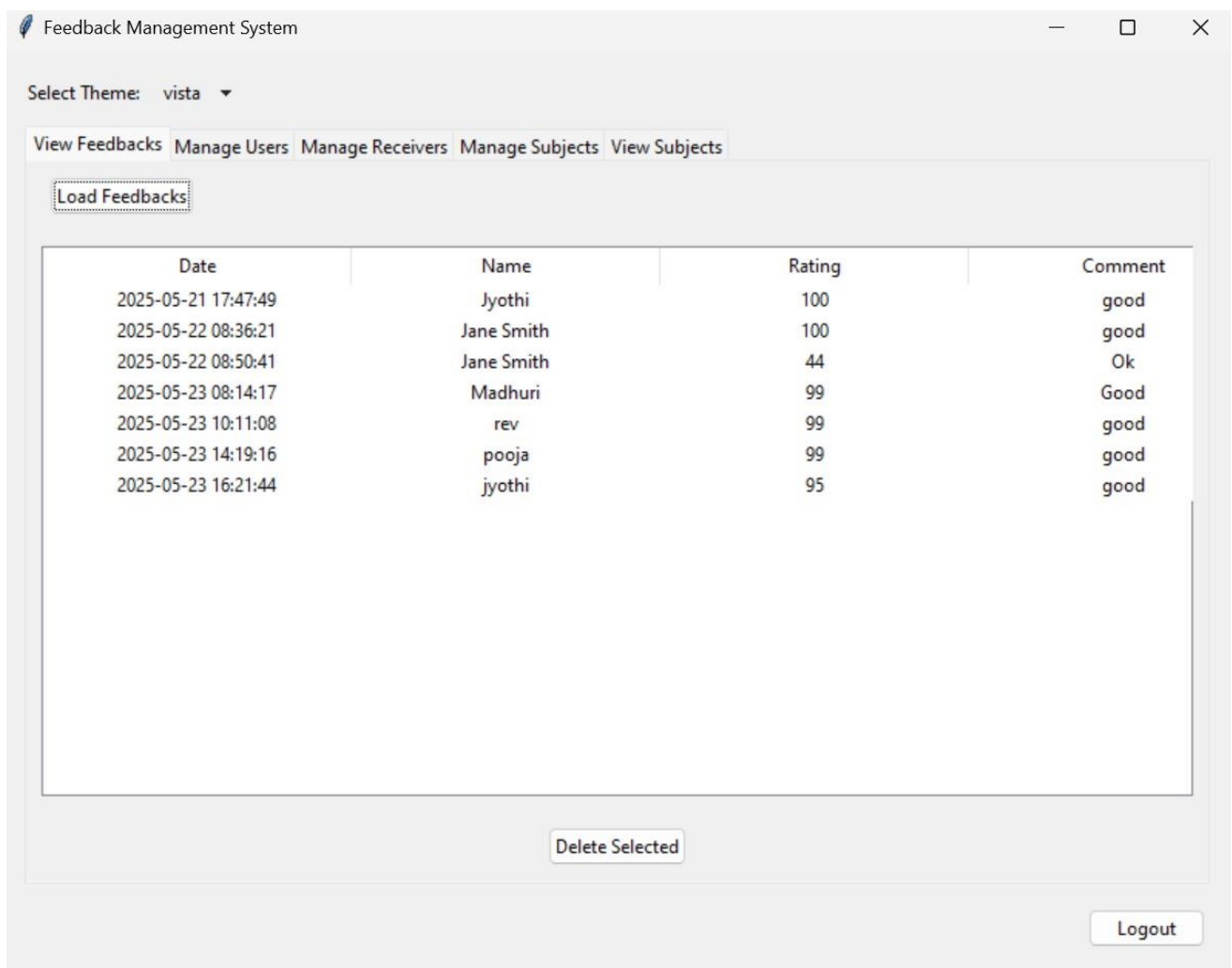
\*\*\*\*\*

Role:

☒ Admin

☐ Student

Login



A screenshot of the 'Feedback Management System' window. It has a title bar with a feather icon and the text 'Feedback Management System'. Below the title bar is a 'Select Theme:' dropdown set to 'vista'. There are five tabs: 'View Feedbacks' (active), 'Manage Users', 'Manage Receivers', 'Manage Subjects', and 'View Subjects'. A 'Load Feedbacks' button is above a table. The table has four columns: 'Date', 'Name', 'Rating', and 'Comment'. Below the table are 'Delete Selected' and 'Logout' buttons.

Select Theme: vista

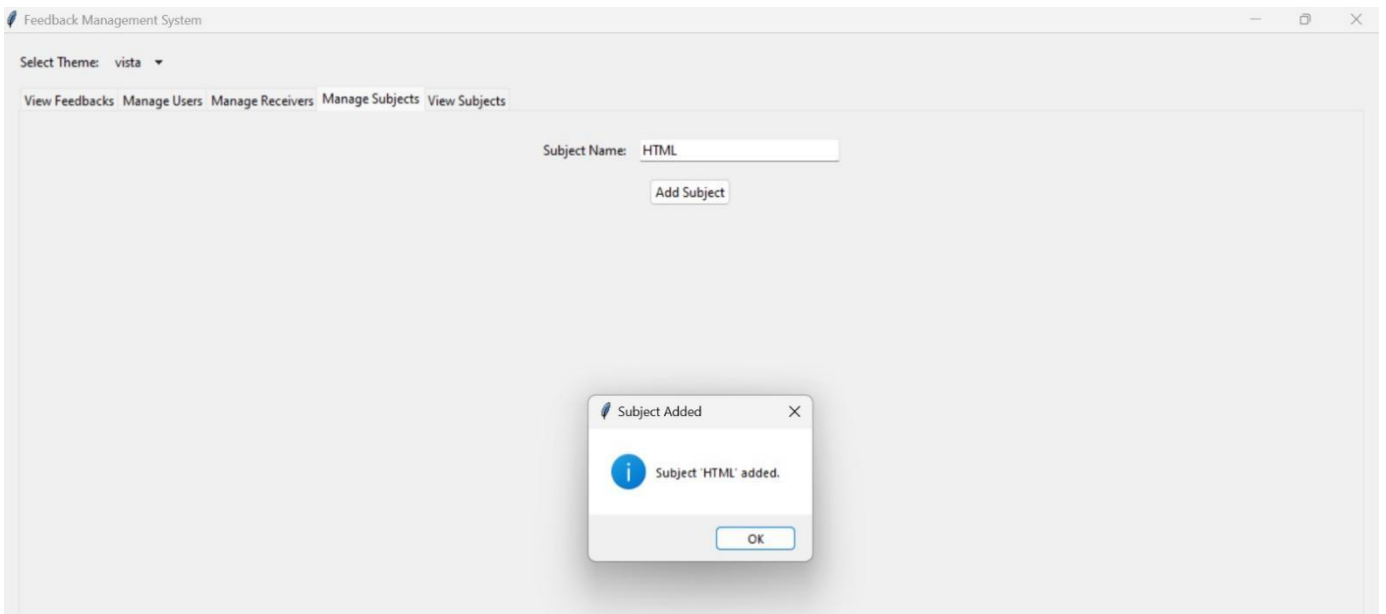
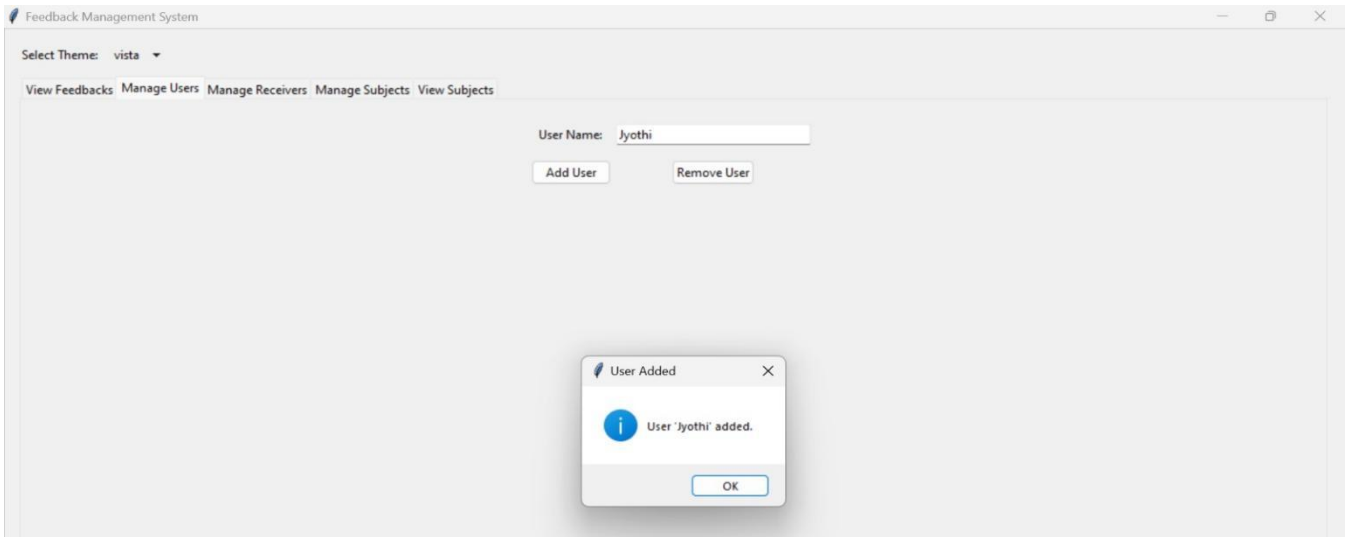
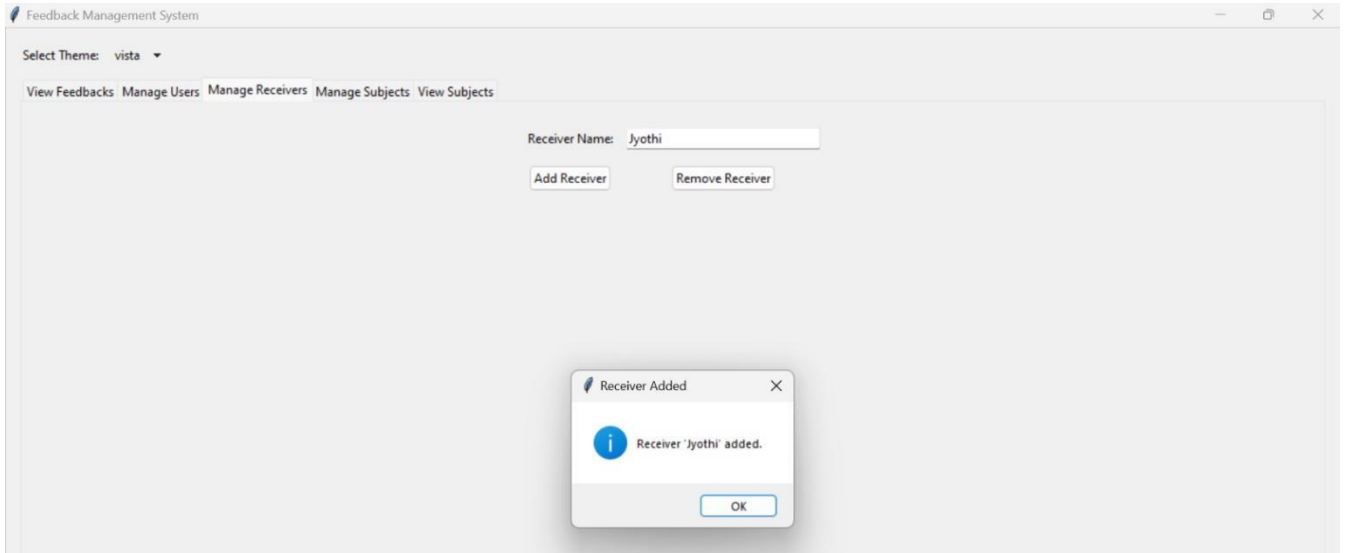
View Feedbacks Manage Users Manage Receivers Manage Subjects View Subjects

Load Feedbacks

Date	Name	Rating	Comment
2025-05-21 17:47:49	Jyothi	100	good
2025-05-22 08:36:21	Jane Smith	100	good
2025-05-22 08:50:41	Jane Smith	44	Ok
2025-05-23 08:14:17	Madhuri	99	Good
2025-05-23 10:11:08	rev	99	good
2025-05-23 14:19:16	pooja	99	good
2025-05-23 16:21:44	jyothi	95	good

Delete Selected

Logout



Feedback Management System

Select Theme: vista

Submit Feedback

Name: M Jyothi

Subject: HTML

Rating (0-100): 99

Comment: Good

Did the faculty arrive on time?☒

Was the session interactive?☒

Submit Feedback

Feedback Submitted

i

Thank you for your feedback!

OK

Feedback Management System

Select Theme: vista

View Feedbacks

Manage Users

Manage Receivers

Manage Subjects

View Subjects

Subject Name: HTML

Add Subject

Feedback Management System

Select Theme: vista ▼

Submit Feedback

Name: M Jyothi

Subject:

Rating (0-100):

Comment:

Did the faculty arrive on time? ☐

Was the session interactive? ☐

Submit Feedback

Feedback Management System

Select Theme: vista ▼

Submit Feedback

Name: M Jyothi

Subject:

Rating (0-100):

Comment:

Did the faculty arrive on time? ☒

Was the session interactive? ☒

Submit Feedback

## Conclusion

This project implements a comprehensive **Feedback Management System** using Python's Tkinter library. It supports two user roles: **Students** and **Admins**, each with tailored interfaces and functionalities.

- **Students** can submit feedback on various subjects, including ratings and answers to compulsory questions. The app validates inputs and stores feedback persistently in a JSON file.
- **Admins** have access to extended features, such as viewing and deleting feedback, managing users and receivers, and adding or viewing subjects. This role-based design ensures appropriate access control and usability.

The system uses separate manager classes to handle feedback, subjects, and users, promoting modularity and easier maintenance. It also supports theme selection via Tkinter's built-in themes to enhance user experience.

The login mechanism verifies credentials and user roles before granting access, providing basic security.

Overall, the application offers a clean, user-friendly GUI for both submitting and managing feedback, with data persistence through JSON storage, making it suitable for small to medium scale feedback collection and administrative tasks.

## **FUTURE WORK**

### **1. Implement Persistent User Authentication**

Replace the hardcoded login credentials with a secure user authentication system backed by a database or encrypted file storage.

### **2. Add Edit and Update Feedback Functionality**

Enable users (especially admins) to modify existing feedback entries rather than only deleting or adding new ones.

### **3. Export and Import Data Features**

Provide options to export feedback and subjects data to formats like CSV, Excel, or PDF for reporting and archival purposes. Allow importing data from these formats too.

### **4. Improve User Interface and UX**

Add more intuitive UI elements such as search, filter, and sorting capabilities in feedback and subject views for easier navigation and analysis.

### **5. Add Role-Based Access Control (RBAC)**

Implement fine-grained access control to allow different permissions for various admin levels or roles, improving security and flexibility.

### **6. Add Notification System**

Integrate email or in-app notifications to alert users or receivers when new feedback is submitted or important changes occur.

### **7. Integrate a Database Backend**

Transition from JSON file storage to a database system (like SQLite, PostgreSQL) to improve scalability, concurrency, and data integrity.

### **8. Add Analytics and Reporting**

Include dashboards and visual charts summarizing feedback ratings, trends, and subject-wise statistics for better decision-making.

### **9. Enable Multi-Language Support**

Localize the app to support multiple languages to broaden its accessibility.

### **10. Implement Persistent Theme Preferences**

Save the user's selected UI theme across sessions to enhance personalization.

### **11. Improve Validation and Security**

Add more robust input sanitization, password hashing, and protection against injection or file corruption.

### **12. Mobile-Friendly or Web-Based Version**

Consider creating web app or mobile version to allow access from devices and locations.



## REFERENCES

### 1. Tkinter Documentation

Official documentation and tutorials for Tkinter, the standard Python GUI library used for building desktop applications.

- <https://docs.python.org/3/library/tkinter.html>
- <https://tkdocs.com/tutorial/index.html>

### 2. Python JSON Module

Documentation on how to work with JSON data for saving and loading feedback and subjects in JSON format.

- <https://docs.python.org/3/library/json.html>

## GIT-HUB REPOSITORY LINK:

<https://github.com/LakshmiaPannangi-2004/Feedback-Management-System>