Event Management System

Course Project Report

Student Name: Harman Singh
Registration Number: 25BCE10077
Course: CSE1021

Date of Submission: 24/11/2025

Submitted in partial fulfillment of the requirements for the course project.

Contents

1 Introduction

The **Event Management System (EMS)** is a software solution designed to streamline the process of organizing events and managing attendee registrations. In many academic and small business settings, events are still managed using manual methods like paper records or disconnected spreadsheets, leading to inefficiency and data loss. This project provides a centralized, digital platform to handle these tasks effectively using Python.

2 Problem Statement

2.1 Current Issue

Event organizers frequently struggle with:

- Loss of participant data due to poor record-keeping.
- Difficulty in retrieving attendee lists for specific events.
- Lack of a unified system to track both event details and registrations simultaneously.

2.2 Proposed Solution

A command-line based application that allows users to create events, securely store them in a file-based system, and manage guest lists without needing complex database software.

3 Functional Requirements

The system provides the following core functions:

1. **Event Management:**
    - Add new events with Name, Date, and Location.
    - View a list of all upcoming events.
    - Delete cancelled or completed events.
2. **Attendee Management:**
    - Register a participant for a specific event using Event ID.
    - View a master list of all registrations.
    - Remove an attendee from the list.
3. **Data Persistence:**
    - Automatically save all data to CSV files (`events.csv` and `attendees.csv`) so data remains available after restarting the program.

4 Non-Functional Requirements

1. **Usability:** The application uses a clear, numbered menu interface that is easy for non-technical users to navigate.

2. **Reliability:** Data is saved immediately after every add or delete operation, minimizing the risk of data loss in case of a crash.
3. **Portability:** Written in standard Python, the application runs on any operating system (Windows, macOS, Linux) without requiring external software installation.
4. **Maintainability:** The code is organized into separate modules (`events`, `attendees`, `storage`), making it easy to fix bugs or add features later.

5 System Architecture

The project follows a **Modular Architecture**. The `main.py` file acts as the controller, accepting user input and directing it to the appropriate logic modules (`events.py`, `attendees.py`). These modules then interact with the storage layer (`storage.py`) to handle data persistence.

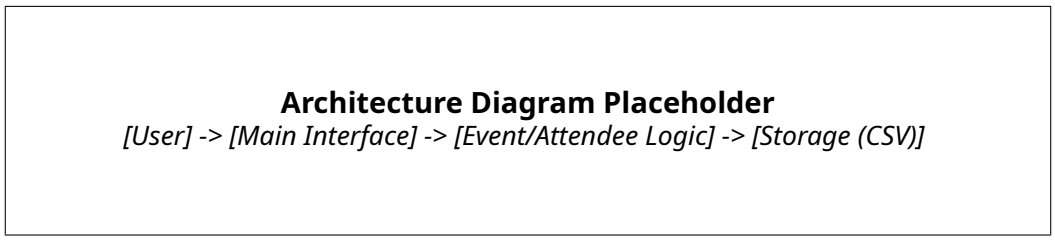

Figure 1: High-Level Modular Architecture

6 Design Diagrams

6.1 Workflow Diagram (User Flow)

This flow illustrates how a user interacts with the system to add an event.

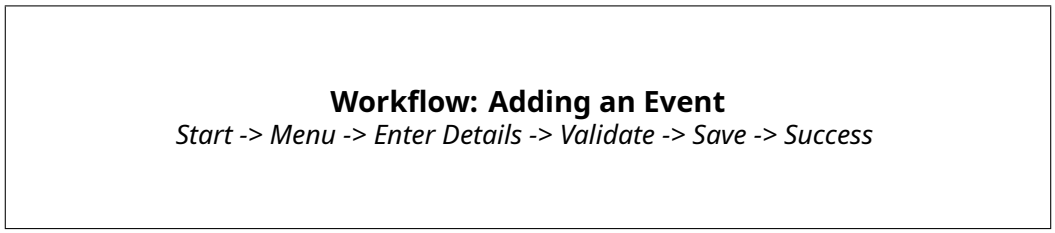

Figure 2: User Workflow for Adding Events

6.2 ER Diagram (Data Structure)

Since we use CSVs, this represents our logical data structure.

- **Event Entity:** `event_id` (PK), `name`, `date`, `location`
- **Attendee Entity:** `name`, `email` (PK), `event_id` (FK)

7 Design Decisions & Rationale

- **Choice of Language (Python):** Python was chosen for its simplicity and strong support for file handling (`csv` library) and date manipulation.

- **Storage (CSV vs Database):** For a beginner project, setting up SQL databases (like MySQL) is complex. CSV files were chosen because they are human-readable, require no installation, and can be opened in Excel for verification.
- **Modular Structure:** Instead of putting all code in one file, I split it into 5 files. This satisfies the project requirement for modularity and makes the code cleaner.

8 Implementation Details

The project consists of 5 core Python files:

- `main.py`: The entry point containing the `while` loop for the menu system.
- `events.py`: Contains functions `add_event`, `list_events`, and `delete_event`.
- `attendees.py`: Contains logic to link people to event IDs.
- `storage.py`: A reusable module that handles `csv.DictReader` and `csv.DictWriter` operations.
- `utils.py`: Contains utility functions like `clear_screen()` and `get_valid_date()` to handle errors.

9 Screenshots & Results

Screenshot 1: Main Menu
Please insert screenshot of the main menu here.

Figure 3: Main Menu Interface

10 Testing Approach

I performed **Black Box Testing** by manually running the program and trying various inputs:

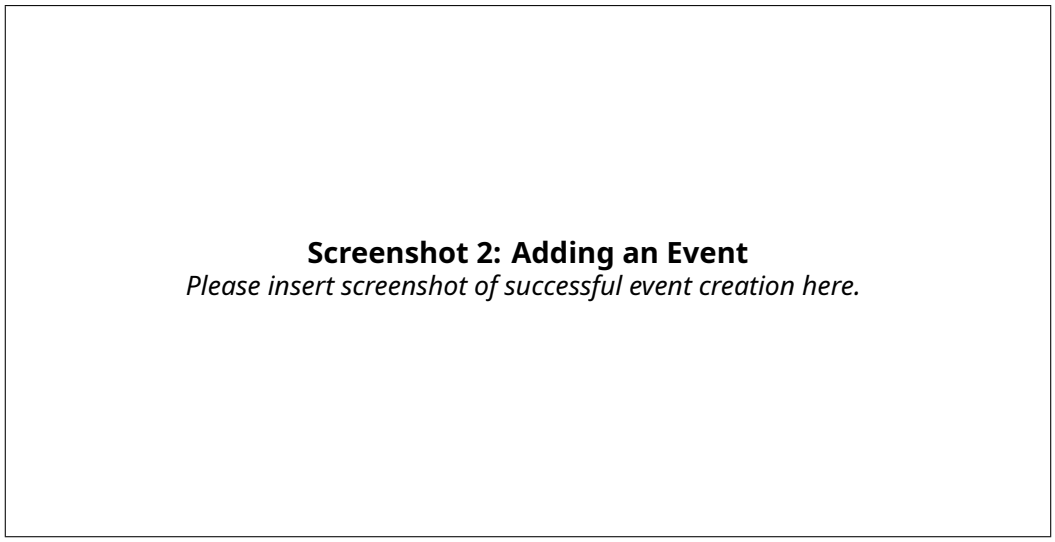

Figure 4: Adding an Event

ID	Description	Input Data	Result
TC01	Add valid event	Valid Name/Date	Pass
TC02	Date Validation	”25-12-2023”	Pass (Error caught)
TC03	Delete Event	Valid ID	Pass
TC04	Delete non-existent	ID: 9999	Pass (Error caught)
TC05	Data Persistence	Restart App	Pass

11 Challenges Faced

- **File Paths:** Initially, the program could not find `events.csv` when running from a different directory. I solved this by learning how to navigate directories in the terminal.
- **Date Formatting:** Users often enter dates differently. I implemented a `try-except` block in `utils.py` to force the `YYYY-MM-DD` format.

12 Learnings & Key Takeaways

- Learned how to use Python’s `csv` module to create persistent applications.
- Understood the importance of modular programming—splitting code makes it easier to debug.
- Gained experience with file I/O modes (`w`, `r`, `a`) and error handling.

13 Future Enhancements

- **Graphical User Interface (GUI):** Building a frontend using Tkinter.
- **Email Notifications:** Automatically sending confirmation emails to attendees upon registration.
- **Duplicate Checks:** Preventing the same person from registering for the same event twice.

14 References

1. Python Official Documentation: `https://docs.python.org/3/`
2. ”Automate the Boring Stuff with Python” by Al Sweigart.
3. StackOverflow discussions on `os.system('cls')`.