

Python Assessment

Events management - Web - Application



Overview:

EMS (events management system) is a web-based application designed to facilitate the management of events and attendees. It supports CRUD operations for events and attendees, providing a seamless experience for organizing and coordinating conferences.

Features:

1. List All Events:

- Display a paginated list of all events.
- Include key details like event name, date, and location.
- Each event entry should have links/buttons for viewing, editing, and deleting.

2. View Individual Event:

- Clicking on an event from the list leads to a detailed page.
- Display all relevant information about the event, such as date, time, venue, and attendees.
- Include options to edit and delete the event.

3. Edit Event:

- Provide a form to modify event details.
- Include fields for updating information like event name, date, time, and location.
- Save and Cancel buttons to confirm or discard changes.

4. Delete Event:

- Display a confirmation prompt before deleting an event.
- Implement a secure mechanism to prevent accidental deletions.

5. List Attendees:

- Show a paginated list of attendees for each event.
- Include attendee names, emails, or other relevant information.
- Provide links/buttons to view details, edit, or remove an attendee.

6. Add Attendee to Event:

- Include a form for adding new attendees to an event.
- Collect attendee information such as name, email, etc.
- Validate input data before submission.

7. Delete Attendee from Event:

- Add a delete option next to each attendee in the list.
- Display a confirmation prompt before removing an attendee.

Assessment Criteria:

• CRUD Operations:

- Ensure the ability to create, read, update, and delete both events and attendees.

• Return Values:

- After creating or editing an event/attendee, provide a confirmation message or redirect to the updated information.

• Update Values:

- Implement functionality to update event and attendee details.

• Delete Values:

- Implement secure and confirmed mechanisms for deleting events and attendees.

User Interface (UI):

- Keep the interface intuitive and user-friendly.
- Use responsive design for better accessibility on different devices.

Security:

- Implement authentication and authorization to ensure that only authorized users can perform CRUD operations.

Testing:

- Thoroughly test to identify and fix bugs.
 - Perform usability testing to ensure a positive user experience.
-

What Topics from what modules will be included:

Core Python
Datatypes
Variables
Operators
Control Statements
Arrays & Functions
Packing & Unpacking
Slicing
Modules & Exception
Object Oriented Programming Concepts
Exception and File Handling
Functional Programming Concepts

How do I plan to implement Core python topics:**1. Datatypes:**

- Define appropriate data types for event details, attendee information, and other relevant data in your application. For example, use strings for event names, integers for event IDs, and so on.

2. Variables:

- Use variables to store and manipulate data throughout your application. For instance, store event details in variables and update them as needed during CRUD operations.

3. Operators:

- Utilize operators for performing operations on variables and data. For instance, use comparison operators when filtering events or arithmetic operators for any calculations.

4. Control Statements:

- Implement control statements (if, else, while, for) to control the flow of your application. For example, use conditional statements to handle different cases during CRUD operations.

5. Arrays & Functions:

- Use arrays (or lists in Python) to store collections of data, such as a list of attendees for an event. Functions can be employed to encapsulate and modularize code for various operations like CRUD, allowing for better organization and reusability.

6. Packing & Unpacking:

- Packing and unpacking can be useful for passing multiple values between functions or components. For example, packing event details into a tuple or list and unpacking them in a function.

7. Slicing:

- Leverage slicing to retrieve specific segments of data. For example, in the context of displaying attendees for an event, slicing can be employed to showcase a manageable and segmented snippet from the entire list.

8. Modules & Exception:

- Organize your code into modules to promote code reusability. Use exception handling to gracefully manage errors during CRUD operations, ensuring a more robust and fault-tolerant application.

9. Object-Oriented Programming Concepts:

- Implement classes and objects to model entities in your application (e.g., Event, Attendee). This helps in organizing code and applying OOP principles such as encapsulation and inheritance.

10. Exception and File Handling:

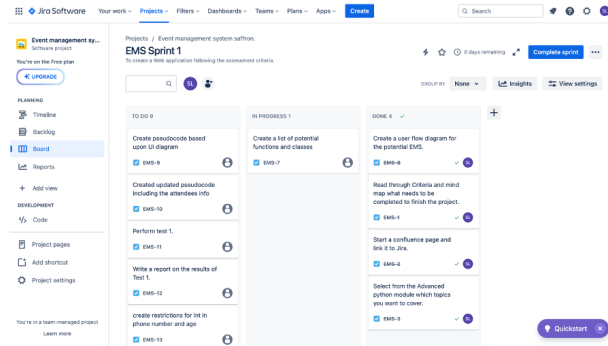
- Utilize exception handling to deal with errors during file operations, database interactions, or other I/O operations. File handling may be relevant for storing persistent data.

11. Functional Programming Concepts:

- Incorporate functional programming concepts like higher-order functions and immutability where applicable. Functions can be treated as first-class citizens, allowing for functional composition and a more declarative coding style

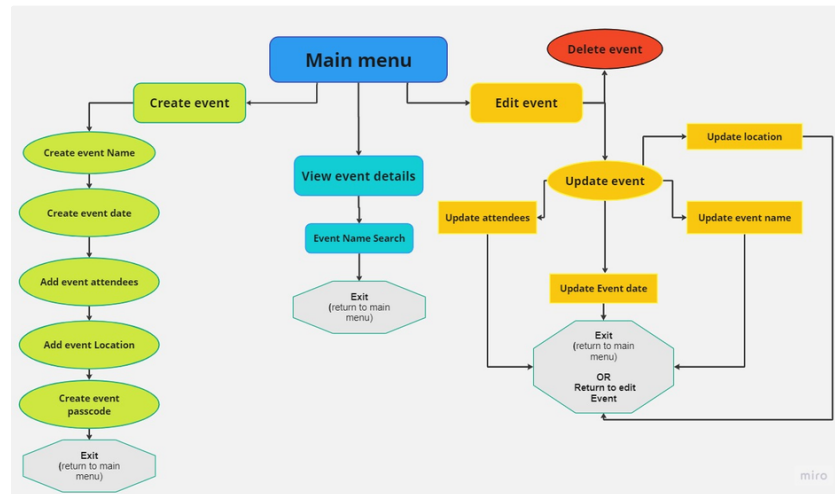
Management of the project:

This project will be managed using the scrum methodology on jira. The project time span is a week so the sprint will be a week long . The Jira page has a link to this confluence page where the progress of the project will be displayed.



Interface Design/ User Interface:

A user flow diagram for the EMS - Version 1



Interface logic:

The user should initially be met with a welcome message and the display of the main menu. There are three options in which the users can pick from: create event, view event details, and edit event.

The process of creating the event does not allow null data so each section of the event creation requires an input. Once the event is created the EMS will ask the user to produce a passcode for this event to protect this data. They will then be met with the Exit option which will return the interface to the main menu.

In the process of viewing an already existing event the user will be asked to input the name of the event, if the event already exists they will need to input a secret code to access the encrypted information. The user may only view the event details here and when they are done they can move onto the exit choice to return to the main menu.

In the scenario where a user need to edit or delete an event they can use the edit event option on the main menu. The edit event option will ask the user to input the name of the event and similar to viewing the event they will need to input a secret code. After inputting the code the user will have two more options either update the existing event or delete the event completely. If the user chooses to update the exiting event there are four options: update attendees, update location, update event date, or update event name. Once the user has completed any of the mentioned options they can either return to edit the event further or they can exit, returning them to the main menu.

Creating a pseudocode for the application:

This is the first version of the potential code that will be used to create the application. It gives a brief description of all the classes, methods, functions, and attributes that I will include.

```
class Attendee:
    attributes: name

class Event:
    attributes: name, date, venue, password, attendees (list of Attendee)

    methods:
        log_info(message)
        log_error(message)
        display_details()
        check_password(password)
        edit_event()

class EventManager:
    attributes: events (dictionary)

    methods:
        create_event(name, date, venue, password)
        edit_event(event)
        display_event_details(event)

class EventManagementApp:
    attributes: event_manager, csv_filename

    methods:
        save_events_to_csv()
        display_home_menu()
        display_main_menu()
        create_event()
        edit_event()
        display_event_details()
        run()
```

Pseudocode Version 1. (Before I added in the attendees attributes)

```
class Attendee:
    attributes: name, phone_number, age, gender

class Event:
    attributes: name, date, venue, password, attendees (list of Attendee)

    methods:
        log_info(message)
        log_error(message)
        display_details()
        check_password(password)
        edit_event()

class EventManager:
    attributes: events (dictionary)

    methods:
        create_event(name, date, venue, password)
        edit_event(event)
        display_event_details(event)

class EventManagementApp:
    attributes: event_manager, csv_filename

    methods:
        save_events_to_csv()
        display_home_menu()
        display_main_menu()
        create_event()
        edit_event()
        display_event_details()
        run()
```

Pseudocode Version 2 (Including the attendees attributes)

Creating Rules and restrictions within the code:

In a real life situation you would not be able to have null values for certain attributes when your entering in details onto an application.

I created a list of rules/ restrictions I need to implement in the application:

Rule	Attribute
No Null Values	All attendee values, event details.
Character Requirements	Event password
Date has to be in the future	Event date
Gender (only 'F' or 'M')	Gender
Integer only	Phone number, age.

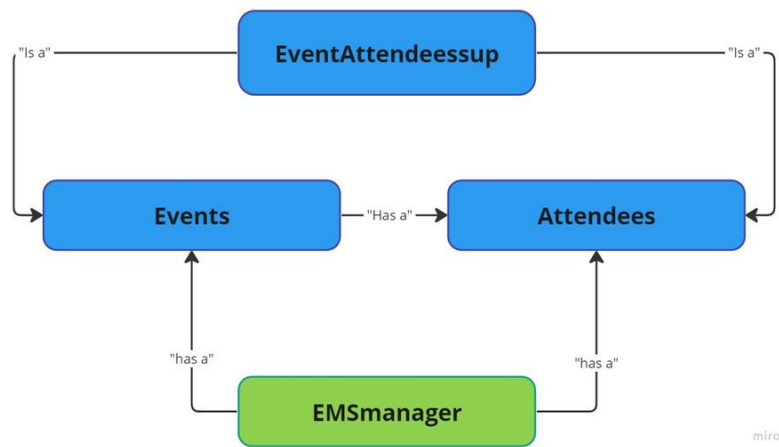
Creating a way of saving the data inputted by the user:

After the user creates the events within the app this information should be saved onto a CSV file. In order to do this I need to create a method that will open a CSV file, create a CSV writer that will write the header, and then iterate through the events, adding in each and every event detail.

```
1 with open(self.csv_filename, 'w', newline='') as csvfile:
2     fieldnames = ['name', 'date', 'venue', 'password', 'attendee_name', 'attendee_phone', 'attendee_age',
3                 'attendee_gender']
4     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

- The open function opens the CSV file specified by 'self.csv_filename' in write mode ('w').
- 'csv.DictWriter' then creates a CSV writer (writer). It takes the CSV file and a list of fieldnames as parameters.
- writeheader() is then called to write the header row to the CSV file using the specified field names.
- I will then input the parameters that I need to add to the file (the event details) and the code will iterate over each event and its attendees in self.event_manager.events.
- For each attendee, I will create a dictionary with keys corresponding to the field names and values populated with the relevant data. writerow() is called to write this dictionary as a row to the CSV

Classes/ Relationships/ Inheritance:



The inheritance relationships shown in blue represent the "is a" relationships". The parent class is the EventAttendeesup class and the child classes are events and attendees.

There are also "has a" relationships also known as composition relationships that exist between event and attendee (event has attendees), and also between EMSmanager and events and attendees.

Results of Test 1:

Performed on version 1 - (based upon the pseudocode version 2)

😊 What went well:

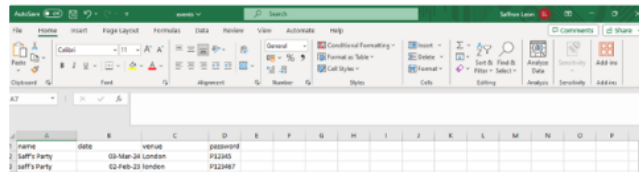
- The functions all work well and the expected comments are printed.
- The logging works well.

- The CSV file is being created when the user inputted the data.

```
Event 'Test party' added successfully with a password.
Main Menu:
1. Create Event
2. View Event Details
3. Edit Event
4. Exit
Enter your choice (1-4): 1
Enter Event Name: Test party
Event already exists. Try a different name.
```

😊 Even better if:

- When the CSV file is created it did not include the attendees information/ details.
- The classes and heritage do not align with the updated version (there is a pointless class).
- More restrictions around data type (ie for the phone number and age there shouldn't be letters)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		name	date	venue	password											
2		Test's Party	03-Mar-24	London	P12345											
3		Test's Party	02-Feb-23	London	P1234567											

Results of Test 2:

Performed on version 2.

😊 What went well:

- The CSV file now includes both the event details and the attendee details.
- Removing the extra class makes the code more understandable.

😊 Even better if:

- Implement the restrictions for the phone number and age - only int values
- Implement the unit testing in the code.
- I need solve the update name issue - When the new name is called back it did not appear- need to update the event name to the 'events' dictionary.

Results of Test 3:

Performed on the final draft Version 3.

😊 What went well:

- The code ran smoothly and all the functions worked.
- When the name was updated in the edit events option it recognised this when it was called later on.
- The event was successfully deleted from both the app and the CSV file.

Project Summary:



```
298         self.create_event()
299     elif choice == '2':
300         self.display_event_details()
301     elif choice == '3':
302         self.edit_event()
303     elif choice == '4':
304         print("Exiting the Events Management System. Goodbye!")
305         self.save_events_to_csv()
306         break
307     else:
308         print("Invalid choice. Please enter a number between 1 and 4.")
309
310 app = EMSManager()
311 app.run()
```

Quick overview on how the EMS app Works (ran on Jupiter Notebook).

This project involves the development of an Events Management System (EMS), a web-based application designed to facilitate the management of events and attendees. The key features include CRUD operations for events and attendees, a user-friendly interface, security measures, and thorough testing. The implementation plan encompasses

various topics from core and advanced Python, including iterators, logging, and unit testing.

Overview:

- CRUD operations for events and attendees.
- Responsive user interface with paginated event lists and detailed views.
- Secure authentication and authorization for authorized CRUD operations.
- Thorough testing for identifying and fixing bugs.
- Implementation of core and advanced Python concepts for robust development.

Core Python Topics:

- Datatypes, variables, operators, control statements, arrays, functions.
- Packing and unpacking, slicing, modules, exception handling.
- Object-oriented programming concepts.
- Functional programming concepts.

Management of the Project:

- Scrum methodology on Jira with a one-week sprint.
- Progress displayed on Confluence.

Interface Design:

- User flow diagram for EMS Version 1.
- User-friendly interface with options for creating, viewing, and editing events.

Interface Logic:

- Main menu with options for creating, viewing, and editing events.
- Creation of events involves inputting event details and creating a passcode.
- Viewing and editing events require inputting the event name and a secret code.

Pseudocode:

- Pseudocode versions for creating, viewing, and editing events.
- Inclusion of rules and restrictions to ensure data integrity.

Saving Data:

- CSV file handling to save event details.

Classes/Relationships/Inheritance:

- Inheritance relationships for events, attendees, and EMS manager.
- Composition relationships between events and attendees, and EMS manager with events and attendees.

Incorporation of Advanced Python:

- Logging integrated for monitoring and debugging.
- Iteration used in the add_attendees method.
- Unit testing implemented for systematic testing.

Test Results:

- Unit tests performed with positive results.
- Identified areas for improvement in data handling and data type restrictions.

Project Summary: The EMS project successfully incorporates core and advanced Python concepts, providing a robust system for managing events and attendees. The implementation follows a well-defined plan, including comprehensive testing and integration of advanced Python features. The iterative development process ensures continuous improvement, and the project is well-organized using the Scrum methodology on Jira. Further enhancements can focus on refining data handling, implementing additional restrictions, and improving user feedback. Overall, the EMS is a functional and well-designed web application for event management.