EVENT MANAGEMENT SYSTEM - PROJECT REPORT

1. PROJECT OVERVIEW

The Event Management System is a console-based C++ application designed to provide a robust interface for managing events through a user-friendly menu. The system supports two types of users: Admin and Guest, each having different levels of access and control. The application uses arrays for data storage and supports data persistence through file operations. Admin users can add, update, and delete events, while Guest users are limited to viewing and searching events.

2. KEY FEATURES

I. User Authentication

- Admin access with credentials

- Guest access with read-only permissions

II. Event Operations

- Add new events (Admin only)

- Update existing events (Admin only)

- Delete events (Admin only)

- View all events

- Search events by: ID, Name, Date, Venue

III. Data Persistence

- Loads events from events.txt at startup
- Saves changes to the same file automatically
- Export options to CSV and TXT formats on desktop

IV. Input Validation

- Prevents invalid or out-of-bound data
- Ensures correct format for names, IDs, and dates

V. User Interface

- Clean console-based UI with region based modular functions
- Feedback messages for successful and failed operations

3. UML CLASS DIAGRAM

```
+-----------------------+
| Event |
 +-----------------------+
| - eventId[]: int |
| - eventName[]: string |
 | - eventDate[]: string |
 | - eventVenue[]: string |
 | - eventCount: int |
+----------------------+
| +loadEventsFromFile() |
 | +saveEventsToFile() |
| +addEvent() |
| +updateEvent() |
 | +deleteEvent() |
| +viewAllEvents() |
| +searchEventById() |
| +searchEventByName() |
| +searchEventByDate() |
| +searchEventByVenue() |
| +exportToCSV() |
 | +exportToTXT() |
```
^ | | Uses
```
 +---------------------------+
| AuthenticateUser |
 +---------------------------+
| - username: string |
| - password: string |
+---------------------------+
| +login(): bool |
| +isAdmin(): bool |
+----------------------------+
```

4. CLASS OVERVIEW AND RELATIONSHIPS

1. Event Class

PURPOSE: REPRESENTS AN EVENT ENTITY WITH ALL RELEVANT DETAILS.

Attributes:

- int id
- string name
- string date
- string time
- string venue
- string description

Functions:

- void inputDetails()
- void displayDetails()
- void updateDetails()

Relationship:

- Acts as the core data structure for storing and manipulating individual event information.

-- Utilized directly.

2. Authenticate User Class

PURPOSE: MANAGES USER LOGIN AND ROLE-BASED ACCESS CONTROL.

Attributes:

- string username

- string password

Functions:

- bool login()

- bool isAdmin(string username)

Relationship:

- Used at the beginning of the program to restrict access.

- Determines whether a user has Admin or Guest privileges.

- Influences the accessibility of event modification functions in the main() function.

3. main() Function

PURPOSE: ACTS AS THE USER INTERFACE THROUGH A MENU-DRIVEN CONSOLE INTERFACE.

Responsibilities:

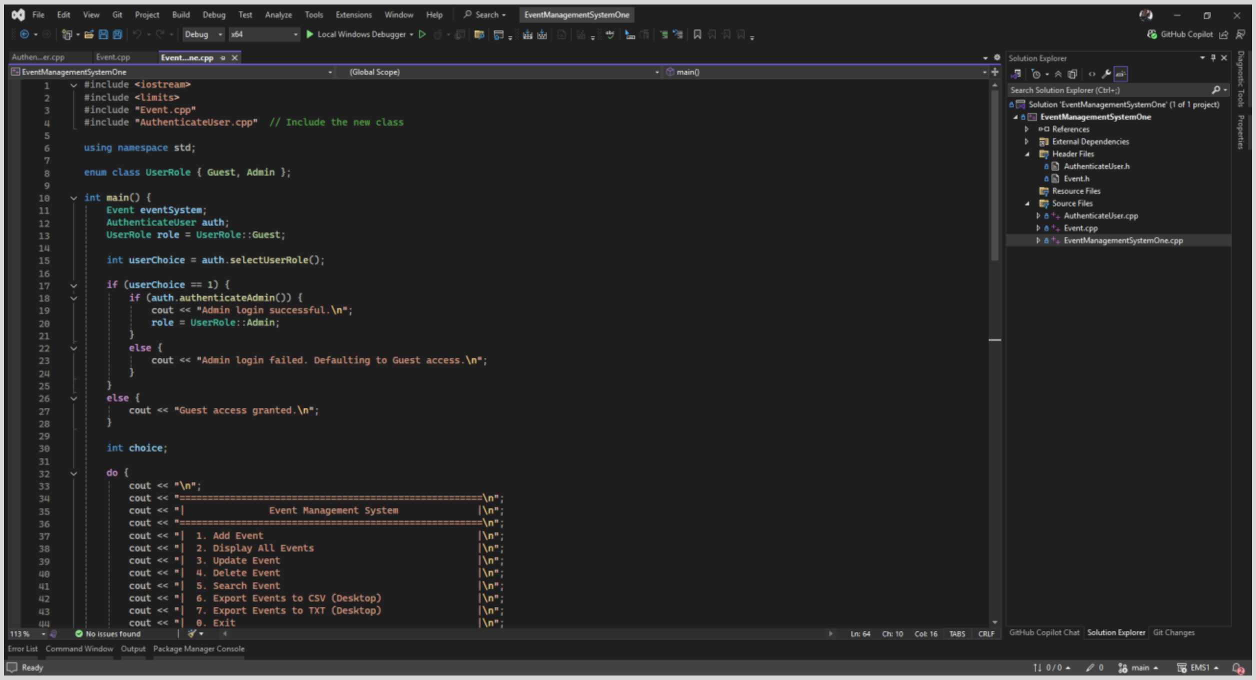
- Initiates user authentication via AuthenticateUser.

- Controls navigation and functionality based on user role.

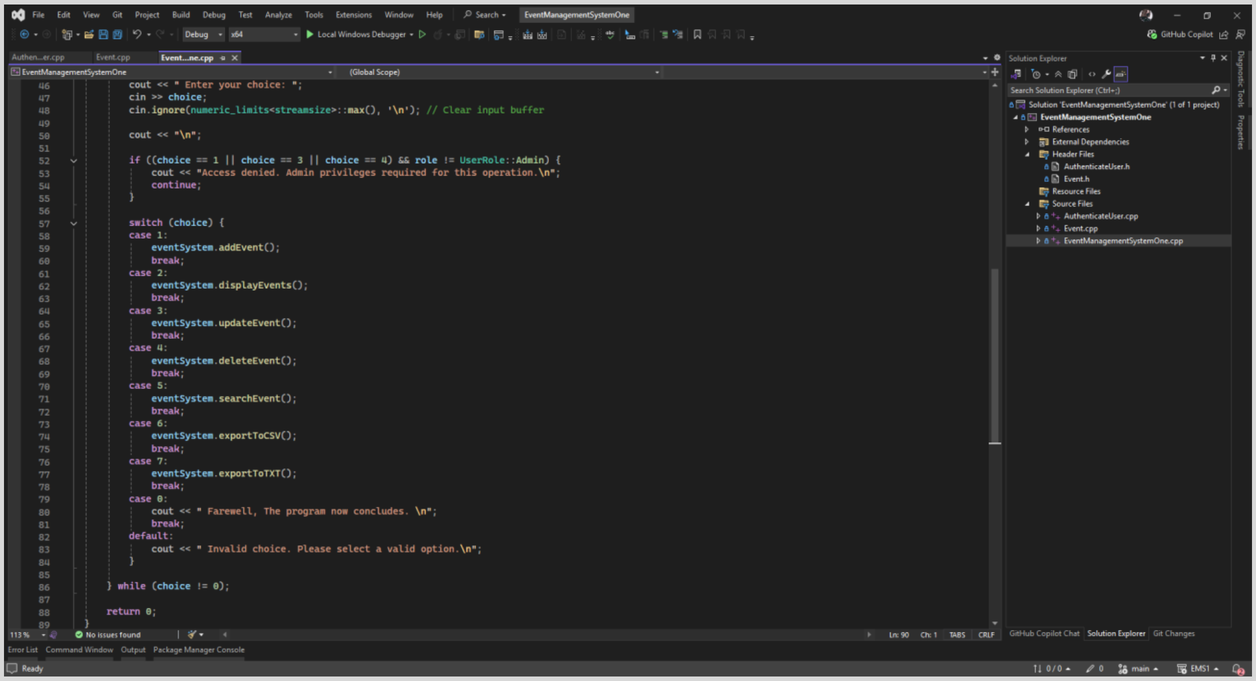
5. CODE SCREENSHOTS & EXPLAINATION

I. USER AUTHENTICATION

The Above given code implements a basic Event Management System with role-based access. It begins by prompting the user to select a role—Admin or Guest. Admins must authenticate before gaining access to sensitive operations such as adding, updating, or deleting events, while Guests can only view, search, and export events. Using a menu-driven interface, the system interacts with the Event and AuthenticateUser classes to perform the selected actions. The program enforces access restrictions and loops until the user chooses to exit.



```cpp
#include <iostream>
#include <limits>
#include "Event.cpp"
#include "AuthenticateUser.cpp"  // Include the new class

using namespace std;

enum class UserRole { Guest, Admin };

int main() {
    Event eventSystem;
    AuthenticateUser auth;
    UserRole role = UserRole::Guest;

    int userChoice = auth.selectUserRole();

    if (userChoice == 1) {
        if (auth.authenticateAdmin()) {
            cout << "Admin login successful.\n";
            role = UserRole::Admin;
        }
        else {
            cout << "Admin login failed. Defaulting to Guest access.\n";
        }
    }
    else {
        cout << "Guest access granted.\n";
    }

    int choice;

    do {
        cout << "\n";
        cout << "======================================================\n";
        cout << "|               Event Management System            |\n";
        cout << "======================================================\n";
        cout << "|  1. Add Event                                      |\n";
        cout << "|  2. Display All Events                             |\n";
        cout << "|  3. Update Event                                   |\n";
        cout << "|  4. Delete Event                                   |\n";
        cout << "|  5. Search Event                                   |\n";
        cout << "|  6. Export Events to CSV (Desktop)                 |\n";
        cout << "|  7. Export Events to TXT (Desktop)                 |\n";
        cout << "|  0. Exit                                           |\n";
```

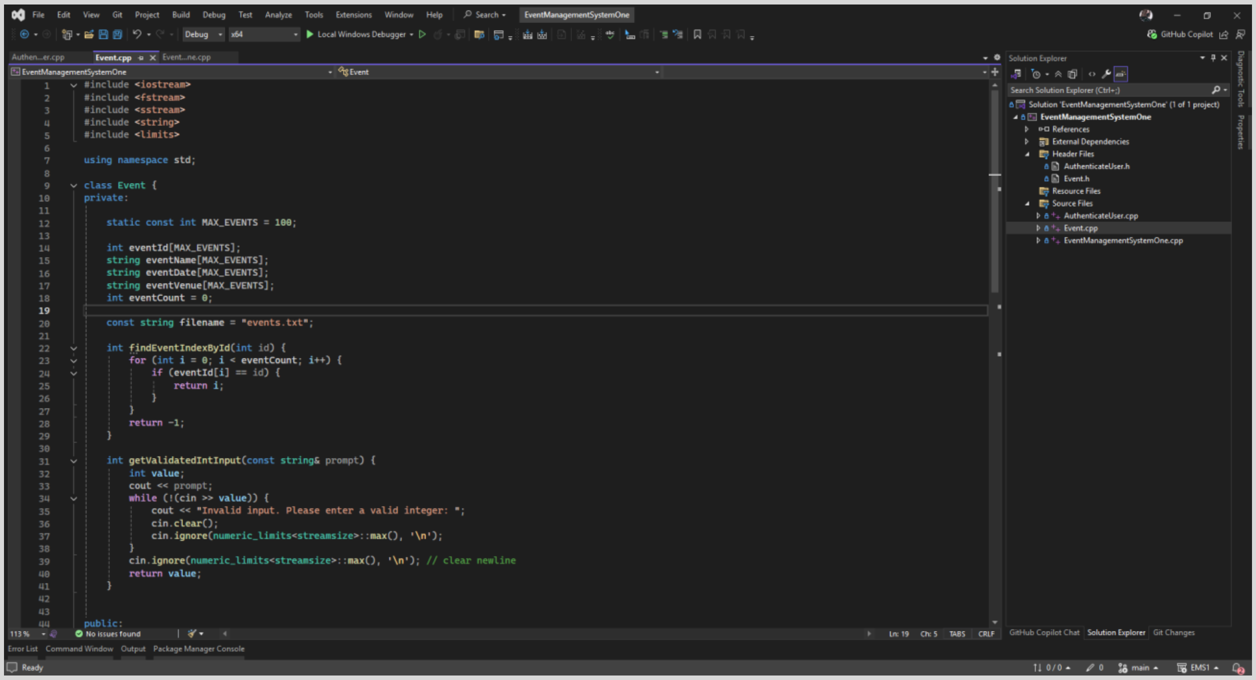


```
        cout << " Enter your choice: ";
        cin >> choice;
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer

        cout << "\n";

        if ((choice == 1 || choice == 3 || choice == 4) && role != UserRole::Admin) {
            cout << "Access denied. Admin privileges required for this operation.\n";
            continue;
        }

        switch (choice) {
        case 1:
            eventSystem.addEvent();
            break;
        case 2:
            eventSystem.displayEvents();
            break;
        case 3:
            eventSystem.updateEvent();
            break;
        case 4:
            eventSystem.deleteEvent();
            break;
        case 5:
            eventSystem.searchEvent();
            break;
        case 6:
            eventSystem.exportToCSV();
            break;
        case 7:
            eventSystem.exportToTXT();
            break;
        case 0:
            cout << " Farewell, The program now concludes. \n";
            break;
        default:
            cout << " Invalid choice. Please select a valid option.\n";
        }

    } while (choice != 0);

    return 0;
}
```

===

II. EVENT OPERATIONS & DATA PERSISTENCE

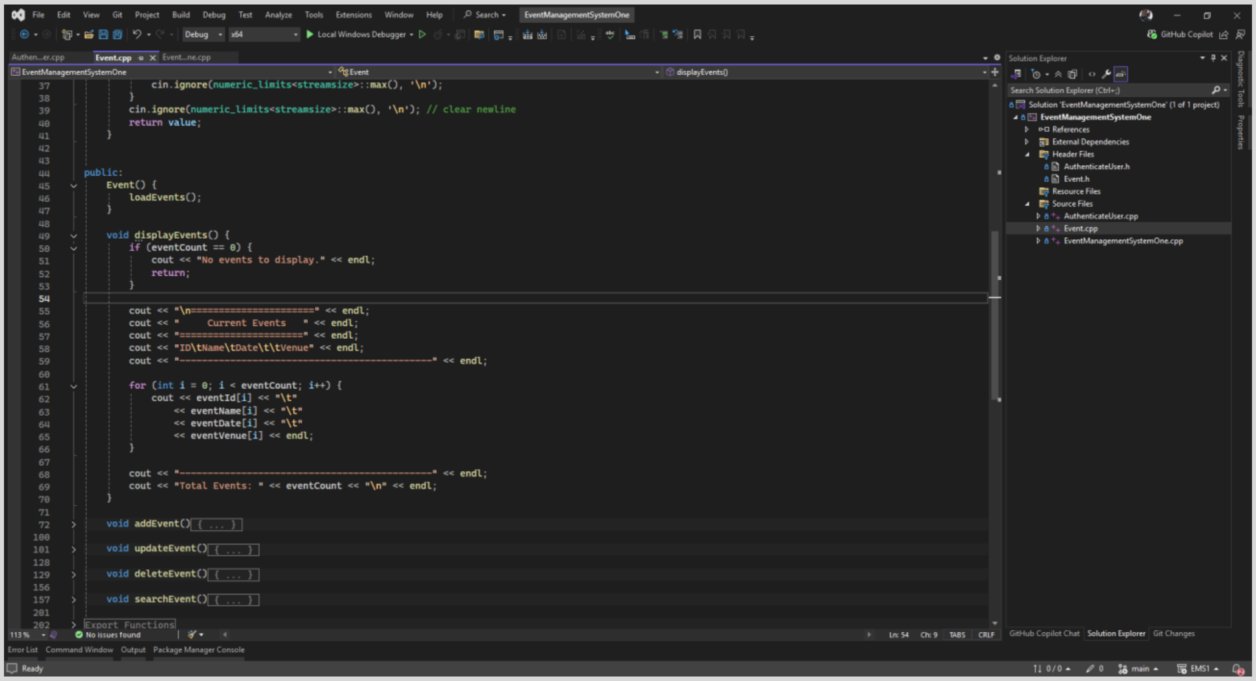
The following code defines the initial structure of the Event class used in the Event Management System. It includes private member variables to store a fixed number (maximum 100) of events, each with an ID, name, date, and venue. The data is stored using parallel arrays, and a counter tracks the current number of events. The class also declares a constant filename for persistent storage. Two private helper functions are included: findEventIndexById, which searches for an event by its ID, and getValidatedIntInput, which ensures robust and error-free integer input from the user.



```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <limits>

using namespace std;

class Event {
private:

    static const int MAX_EVENTS = 100;

    int eventId[MAX_EVENTS];
    string eventName[MAX_EVENTS];
    string eventDate[MAX_EVENTS];
    string eventVenue[MAX_EVENTS];
    int eventCount = 0;

    const string filename = "events.txt";

    int findEventIndexById(int id) {
        for (int i = 0; i < eventCount; i++) {
            if (eventId[i] == id) {
                return i;
            }
        }
        return -1;
    }

    int getValidatedIntInput(const string& prompt) {
        int value;
        cout << prompt;
        while (!(cin >> value)) {
            cout << "Invalid input. Please enter a valid integer: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // clear newline
        return value;
    }


public:
```

==

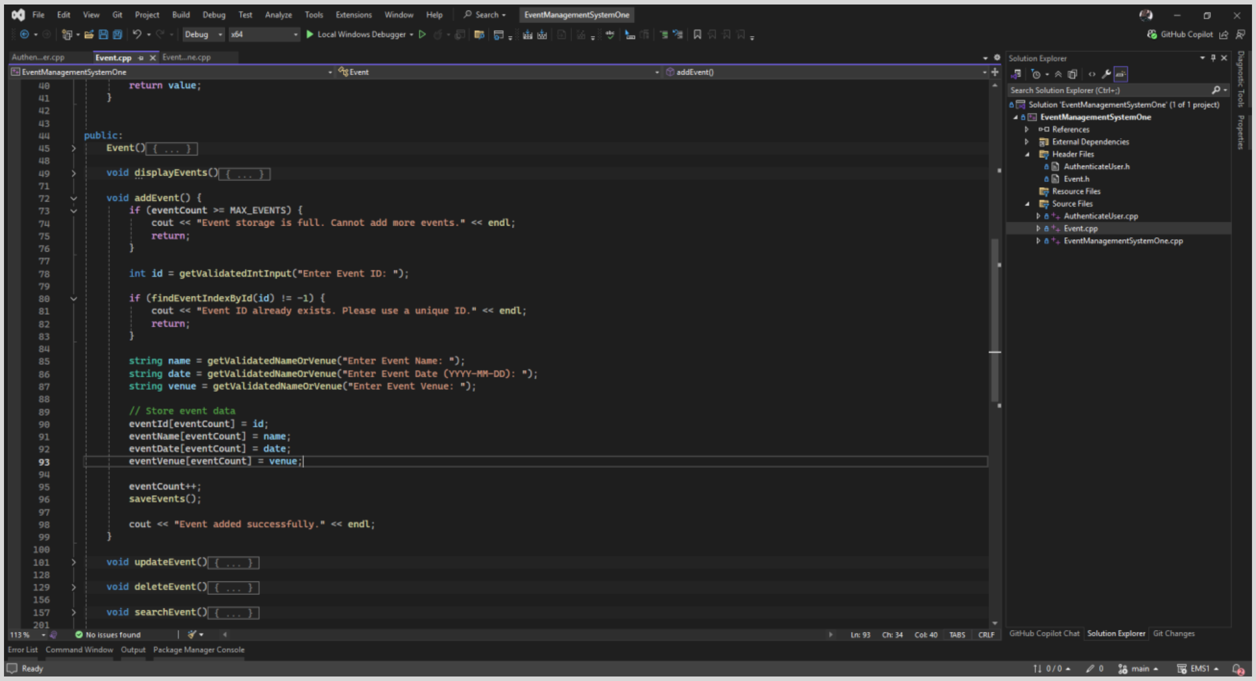
The following code implements the public interface of the Event class, beginning with the constructor, which automatically calls loadEvents() to populate the event data from persistent storage upon object creation. It also defines the displayEvents method, which outputs all currently stored events in a formatted table. If no events are present, the user is informed accordingly. Otherwise, the method iterates through the event arrays and prints each event's ID, name, date, and venue, along with the total number of events recorded.



```cpp
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // clear newline
        return value;
    }


public:
    Event() {
        loadEvents();
    }

    void displayEvents() {
        if (eventCount == 0) {
            cout << "No events to display." << endl;
            return;
        }

        cout << "\n=====================" << endl;
        cout << "     Current Events   " << endl;
        cout << "=====================" << endl;
        cout << "ID\tName\tDate\t\tVenue" << endl;
        cout << "---------------------------------------------" << endl;

        for (int i = 0; i < eventCount; i++) {
            cout << eventId[i] << "\t"
                << eventName[i] << "\t"
                << eventDate[i] << "\t"
                << eventVenue[i] << endl;
        }

        cout << "---------------------------------------------" << endl;
        cout << "Total Events: " << eventCount << "\n" << endl;
    }

    void addEvent() { ... }

    void updateEvent() { ... }

    void deleteEvent() { ... }

    void searchEvent() { ... }

Export Functions
```

===

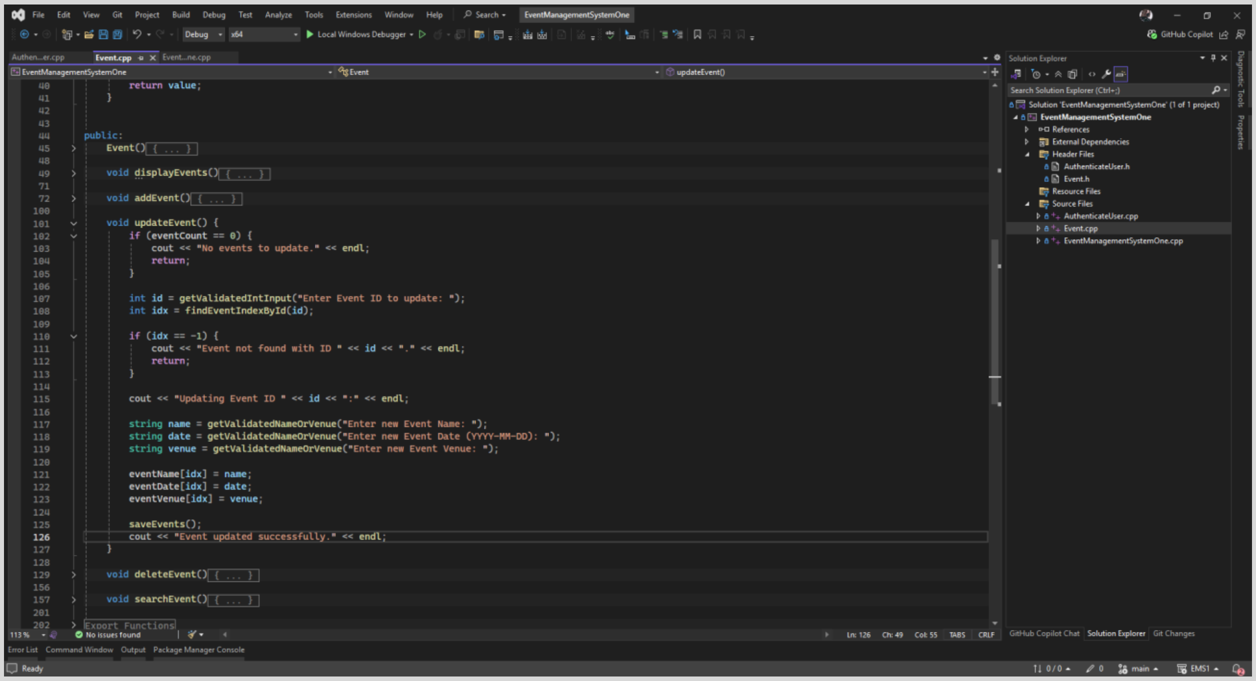
The following code defines the addEvent method, which facilitates the addition of a new event to the system. It first checks whether the maximum event limit has been reached. If not, it prompts the user to enter a unique event ID using a validated input method. If the ID already exists, the operation is aborted. Otherwise, the method collects the event name, date, and venue using a separate input validation function. The collected data is stored in their respective arrays, the event count is incremented, and the updated data is saved to persistent storage. A confirmation message is displayed upon successful addition.



```cpp
            return value;
        }


public:
    Event() { ... }

    void displayEvents() { ... }

    void addEvent() {
        if (eventCount >= MAX_EVENTS) {
            cout << "Event storage is full. Cannot add more events." << endl;
            return;
        }

        int id = getValidatedIntInput("Enter Event ID: ");

        if (findEventIndexById(id) != -1) {
            cout << "Event ID already exists. Please use a unique ID." << endl;
            return;
        }

        string name = getValidatedNameOrVenue("Enter Event Name: ");
        string date = getValidatedNameOrVenue("Enter Event Date (YYYY-MM-DD): ");
        string venue = getValidatedNameOrVenue("Enter Event Venue: ");

        // Store event data
        eventId[eventCount] = id;
        eventName[eventCount] = name;
        eventDate[eventCount] = date;
        eventVenue[eventCount] = venue;

        eventCount++;
        saveEvents();

        cout << "Event added successfully." << endl;
    }

    void updateEvent() { ... }

    void deleteEvent() { ... }

    void searchEvent() { ... }
```

==

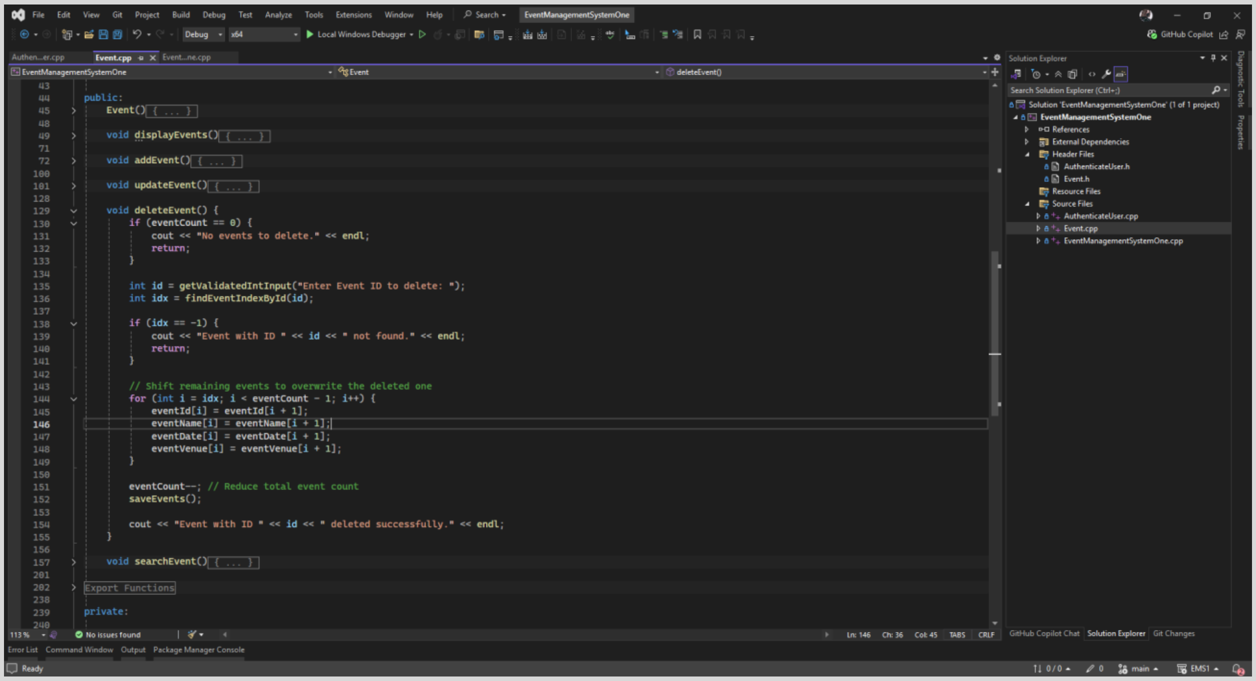
The following code defines the updateEvent method, which allows users to modify the details of an existing event. The method begins by verifying whether any events exist. If events are present, it prompts the user to enter the ID of the event they wish to update. Using this ID, it locates the corresponding index in the array; if the ID is not found, an appropriate message is displayed. Upon successful identification, the user is prompted to enter new values for the event's name, date, and venue. These updated values are stored in their respective arrays, and the changes are persisted to the storage file. A success message is displayed upon completion.



```cpp
            return value;
        }


public:
    Event() { ... }

    void displayEvents() { ... }

    void addEvent() { ... }

    void updateEvent() {
        if (eventCount == 0) {
            cout << "No events to update." << endl;
            return;
        }

        int id = getValidatedIntInput("Enter Event ID to update: ");
        int idx = findEventIndexById(id);

        if (idx == -1) {
            cout << "Event not found with ID " << id << "." << endl;
            return;
        }

        cout << "Updating Event ID " << id << ":" << endl;

        string name = getValidatedNameOrVenue("Enter new Event Name: ");
        string date = getValidatedNameOrVenue("Enter new Event Date (YYYY-MM-DD): ");
        string venue = getValidatedNameOrVenue("Enter new Event Venue: ");

        eventName[idx] = name;
        eventDate[idx] = date;
        eventVenue[idx] = venue;

        saveEvents();
        cout << "Event updated successfully." << endl;
    }

    void deleteEvent() { ... }

    void searchEvent() { ... }

Export Functions
```

==

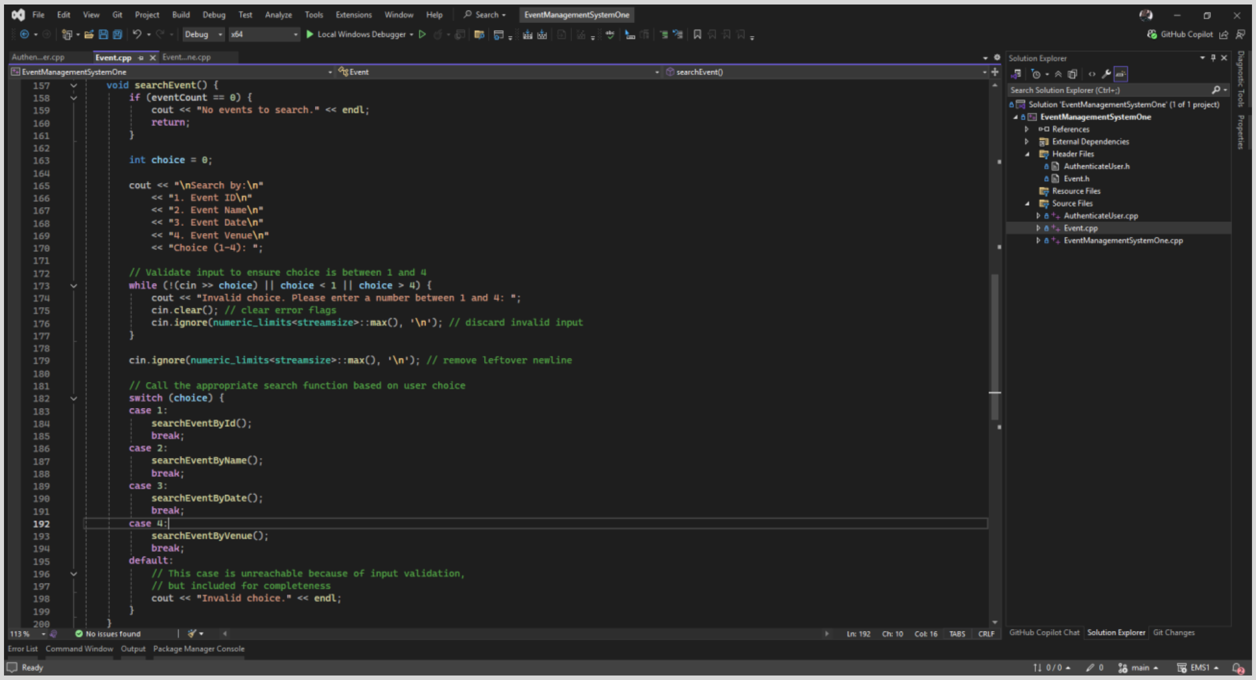
The following code defines the deleteEvent method, which enables the removal of an event based on its ID. It first checks whether any events exist in the system. If so, it prompts the user for the event ID to delete and locates the corresponding index. If the event is not found, a message is shown. Otherwise, it proceeds to delete the event by shifting subsequent event entries one position up in all arrays, effectively overwriting the targeted event. The total event count is then decremented, and the updated data is saved. A confirmation message indicates successful deletion.



```cpp
public:
    Event() { ... }

    void displayEvents() { ... }

    void addEvent() { ... }

    void updateEvent() { ... }

    void deleteEvent() {
        if (eventCount == 0) {
            cout << "No events to delete." << endl;
            return;
        }

        int id = getValidatedIntInput("Enter Event ID to delete: ");
        int idx = findEventIndexById(id);

        if (idx == -1) {
            cout << "Event with ID " << id << " not found." << endl;
            return;
        }

        // Shift remaining events to overwrite the deleted one
        for (int i = idx; i < eventCount - 1; i++) {
            eventId[i] = eventId[i + 1];
            eventName[i] = eventName[i + 1];
            eventDate[i] = eventDate[i + 1];
            eventVenue[i] = eventVenue[i + 1];
        }

        eventCount--; // Reduce total event count
        saveEvents();

        cout << "Event with ID " << id << " deleted successfully." << endl;
    }

    void searchEvent() { ... }

Export Functions

private:
```

==

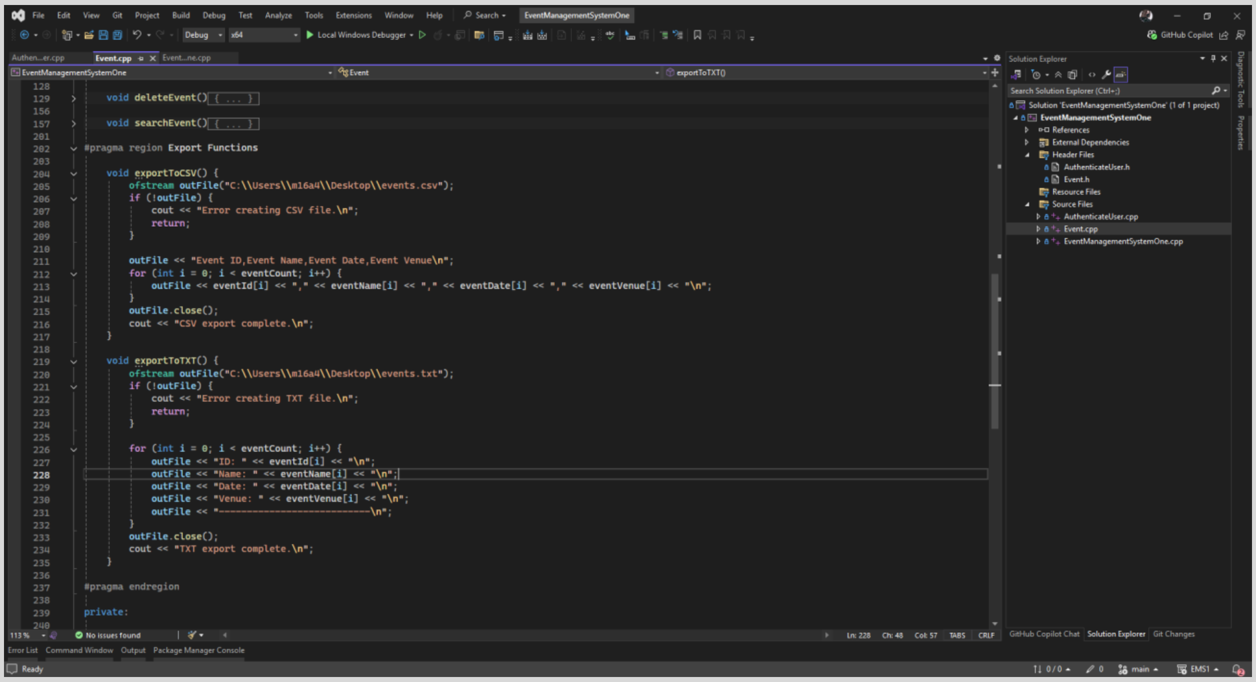
The following code implements the searchEvent method, which allows users to search for events based on different criteria. It begins by checking if there are any events to search. The user is then prompted to select a search category—ID, name, date, or venue—with input validation to ensure a valid choice between 1 and 4. Depending on the selected option, the corresponding helper function is called to perform the search. This modular approach enhances clarity and user interaction while ensuring input integrity.



```cpp
    void searchEvent() {
        if (eventCount == 0) {
            cout << "No events to search." << endl;
            return;
        }

        int choice = 0;

        cout << "\nSearch by:\n"
            << "1. Event ID\n"
            << "2. Event Name\n"
            << "3. Event Date\n"
            << "4. Event Venue\n"
            << "Choice (1-4): ";

        // Validate input to ensure choice is between 1 and 4
        while (!(cin >> choice) || choice < 1 || choice > 4) {
            cout << "Invalid choice. Please enter a number between 1 and 4: ";
            cin.clear(); // clear error flags
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // discard invalid input
        }

        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // remove leftover newline

        // Call the appropriate search function based on user choice
        switch (choice) {
        case 1:
            searchEventById();
            break;
        case 2:
            searchEventByName();
            break;
        case 3:
            searchEventByDate();
            break;
        case 4:
            searchEventByVenue();
            break;
        default:
            // This case is unreachable because of input validation,
            // but included for completeness
            cout << "Invalid choice." << endl;
        }
    }
```

===

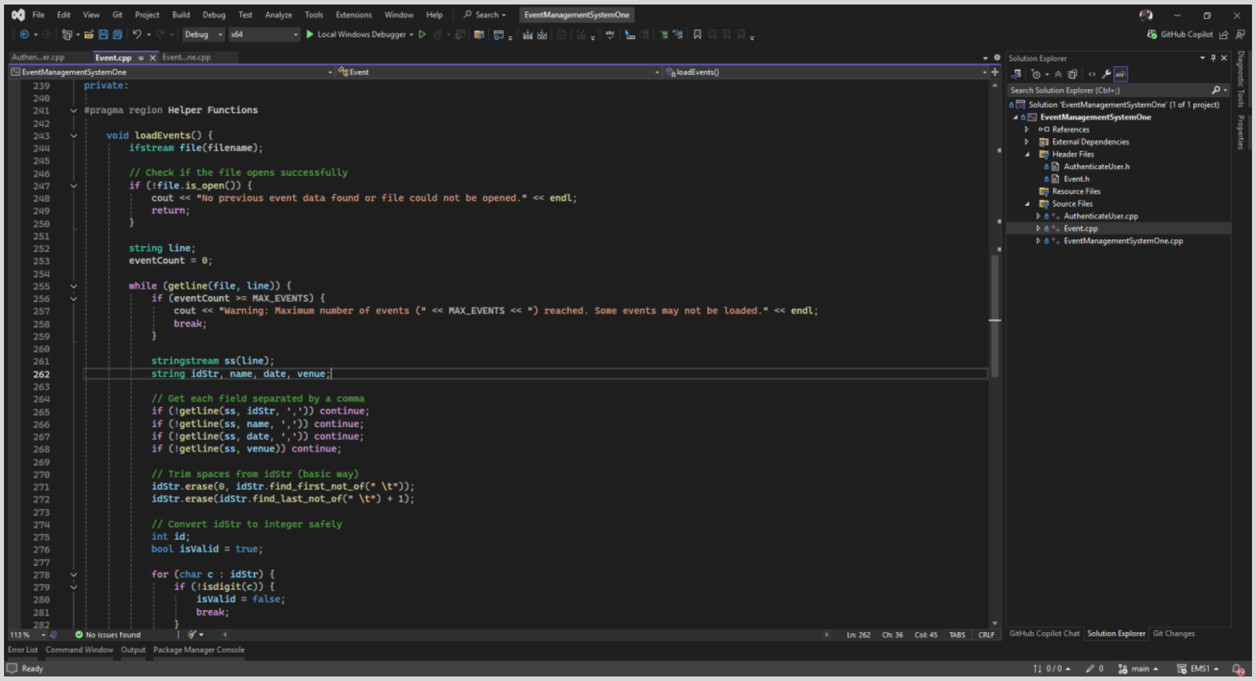
The following code defines two export functions: exportToCSV and exportToTXT, which allow event data to be saved externally in CSV and TXT formats, respectively. Both functions create output files on the desktop and iterate through the list of stored events, formatting and writing each event's details. The CSV export uses a comma-separated format suitable for spreadsheet applications, while the TXT export provides a more human-readable layout. Error checks are performed to handle file creation failures, and success messages are displayed upon completion.



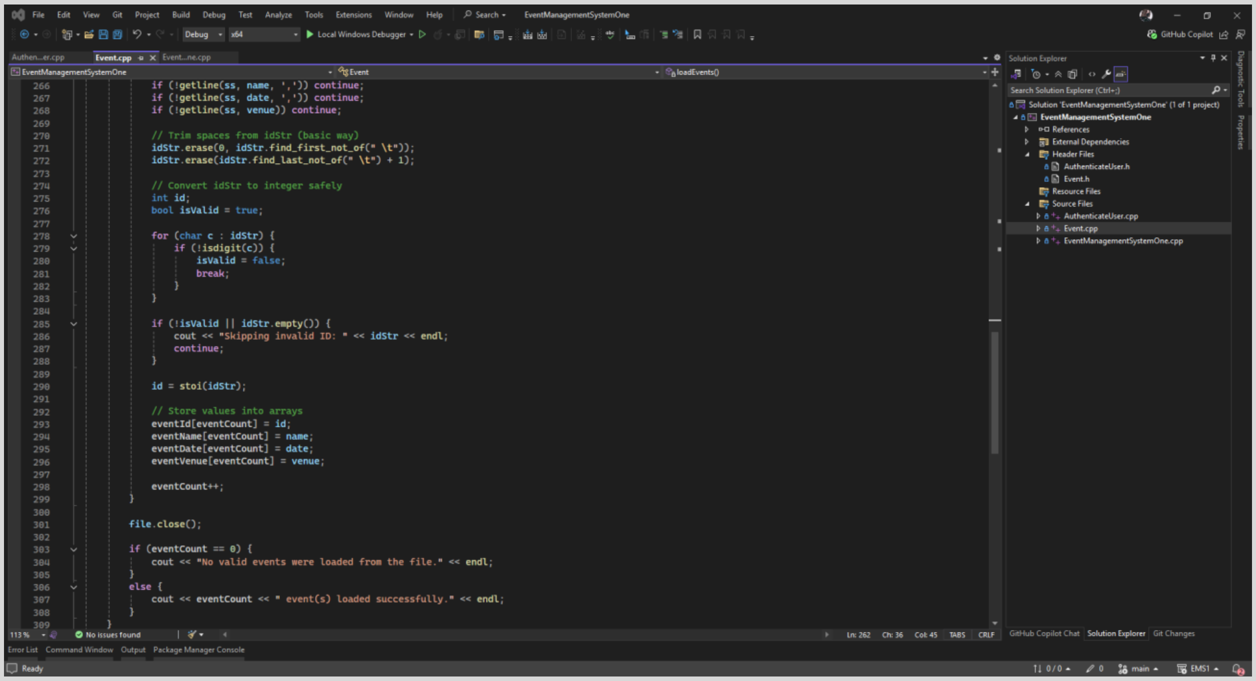
```
    void deleteEvent() { ... }

    void searchEvent() { ... }

#pragma region Export Functions

    void exportToCSV() {
        ofstream outFile("C:\\Users\\m16a4\\Desktop\\events.csv");
        if (!outFile) {
            cout << "Error creating CSV file.\n";
            return;
        }

        outFile << "Event ID,Event Name,Event Date,Event Venue\n";
        for (int i = 0; i < eventCount; i++) {
            outFile << eventId[i] << "," << eventName[i] << "," << eventDate[i] << "," << eventVenue[i] << "\n";
        }
        outFile.close();
        cout << "CSV export complete.\n";
    }

    void exportToTXT() {
        ofstream outFile("C:\\Users\\m16a4\\Desktop\\events.txt");
        if (!outFile) {
            cout << "Error creating TXT file.\n";
            return;
        }

        for (int i = 0; i < eventCount; i++) {
            outFile << "ID: " << eventId[i] << "\n";
            outFile << "Name: " << eventName[i] << "\n";
            outFile << "Date: " << eventDate[i] << "\n";
            outFile << "Venue: " << eventVenue[i] << "\n";
            outFile << "-------------------------\n";
        }
        outFile.close();
        cout << "TXT export complete.\n";
    }

#pragma endregion

private:
```

==

The following code defines the loadEvents method, responsible for reading event data from a persistent file. It attempts to open the designated file and, if unsuccessful, notifies the user that no prior data could be loaded. The method then reads the file line by line, parsing each event's fields separated by commas. It validates the event ID to ensure it consists only of digits before converting it to an integer. Valid event data is stored in the internal arrays until the maximum event capacity is reached. Upon completion, the method reports how many events were successfully loaded or indicates if none were valid.

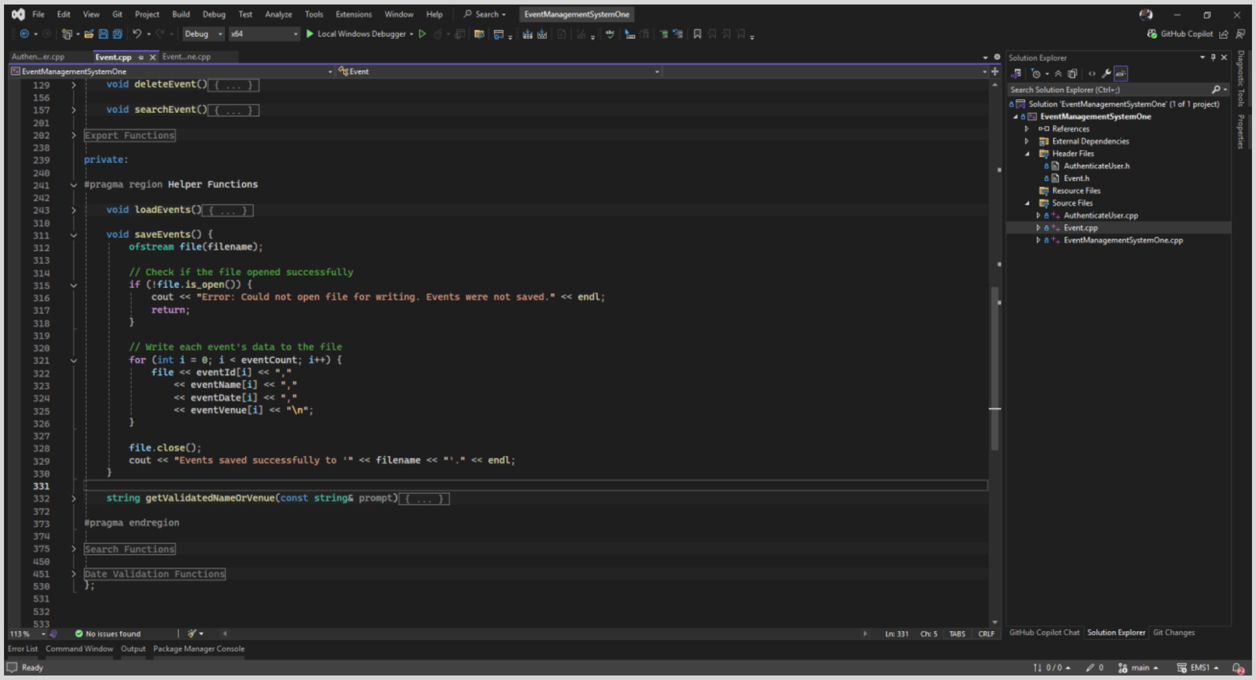


```cpp
private:

#pragma region Helper Functions

    void loadEvents() {
        ifstream file(filename);

        // Check if the file opens successfully
        if (!file.is_open()) {
            cout << "No previous event data found or file could not be opened." << endl;
            return;
        }

        string line;
        eventCount = 0;

        while (getline(file, line)) {
            if (eventCount >= MAX_EVENTS) {
                cout << "Warning: Maximum number of events (" << MAX_EVENTS << ") reached. Some events may not be loaded." << endl;
                break;
            }

            stringstream ss(line);
            string idStr, name, date, venue;

            // Get each field separated by a comma
            if (!getline(ss, idStr, ',')) continue;
            if (!getline(ss, name, ',')) continue;
            if (!getline(ss, date, ',')) continue;
            if (!getline(ss, venue)) continue;

            // Trim spaces from idStr (basic way)
            idStr.erase(0, idStr.find_first_not_of(" \t"));
            idStr.erase(idStr.find_last_not_of(" \t") + 1);

            // Convert idStr to integer safely
            int id;
            bool isValid = true;

            for (char c : idStr) {
                if (!isdigit(c)) {
                    isValid = false;
                    break;
                }
```



```cpp
                if (!getline(ss, name, ',')) continue;
                if (!getline(ss, date, ',')) continue;
                if (!getline(ss, venue)) continue;

                // Trim spaces from idStr (basic way)
                idStr.erase(0, idStr.find_first_not_of(" \t"));
                idStr.erase(idStr.find_last_not_of(" \t") + 1);

                // Convert idStr to integer safely
                int id;
                bool isValid = true;

                for (char c : idStr) {
                    if (!isdigit(c)) {
                        isValid = false;
                        break;
                    }
                }

                if (!isValid || idStr.empty()) {
                    cout << "Skipping invalid ID: " << idStr << endl;
                    continue;
                }

                id = stoi(idStr);

                // Store values into arrays
                eventId[eventCount] = id;
                eventName[eventCount] = name;
                eventDate[eventCount] = date;
                eventVenue[eventCount] = venue;

                eventCount++;
            }

            file.close();

            if (eventCount == 0) {
                cout << "No valid events were loaded from the file." << endl;
            }
            else {
                cout << eventCount << " event(s) loaded successfully." << endl;
            }
        }
```

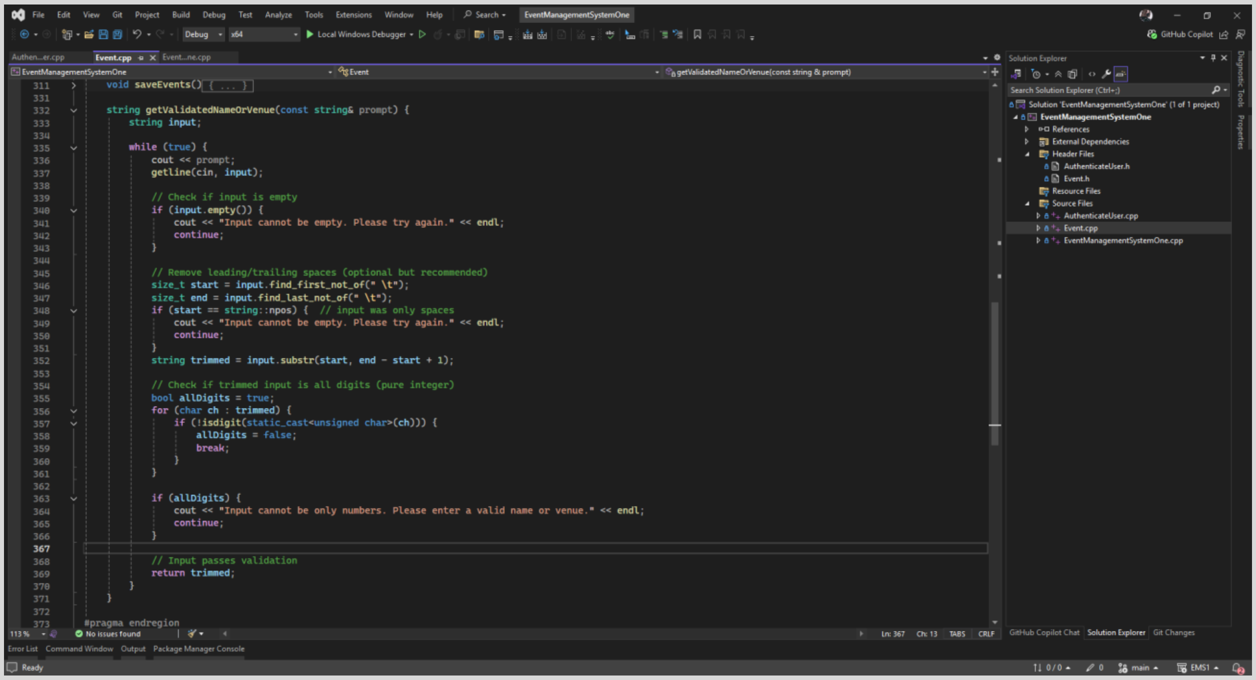
The following code implements the saveEvents method, which writes the current list of events to a persistent file. It opens the designated file for writing and checks for successful access. Each event's details are then output in a comma-separated format, one event per line. After writing all events, the file is closed, and a confirmation message indicates successful saving. If the file cannot be opened, an error message is displayed, and no data is saved.



```cpp
    void deleteEvent(){ ... }

    void searchEvent(){ ... }

Export Functions

private:

#pragma region Helper Functions

    void loadEvents(){ ... }

    void saveEvents() {
        ofstream file(filename);

        // Check if the file opened successfully
        if (!file.is_open()) {
            cout << "Error: Could not open file for writing. Events were not saved." << endl;
            return;
        }

        // Write each event's data to the file
        for (int i = 0; i < eventCount; i++) {
            file << eventId[i] << ","
                << eventName[i] << ","
                << eventDate[i] << ","
                << eventVenue[i] << "\n";
        }

        file.close();
        cout << "Events saved successfully to '" << filename << "'." << endl;
    }

    string getValidatedNameOrVenue(const string& prompt){ ... }

#pragma endregion

Search Functions

Date Validation Functions
};
```

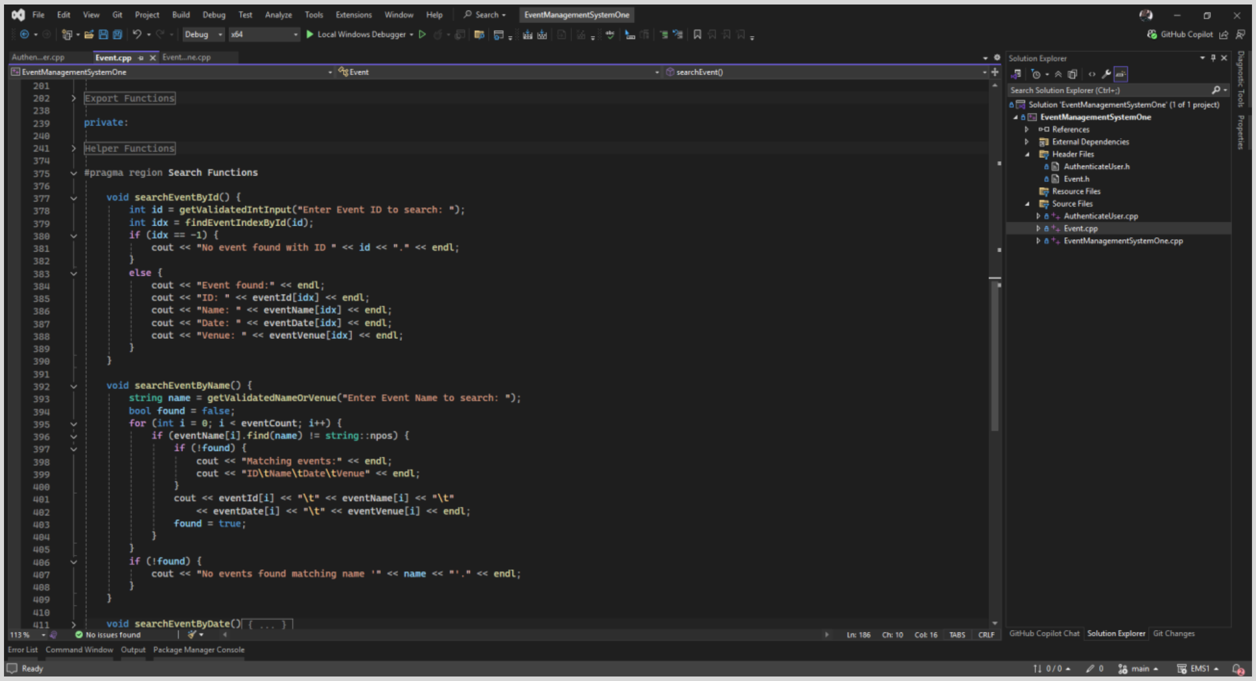
==

The following code defines the getValidatedNameOrVenue method, which prompts the user for input and validates it to ensure it is a non-empty, meaningful string. The method repeatedly requests input until the user provides a value that is not blank, not composed solely of whitespace, and not purely numeric. Leading and trailing spaces are trimmed before validation. If the input fails any of these checks, an appropriate message is displayed, and the prompt repeats. Once valid, the trimmed input is returned

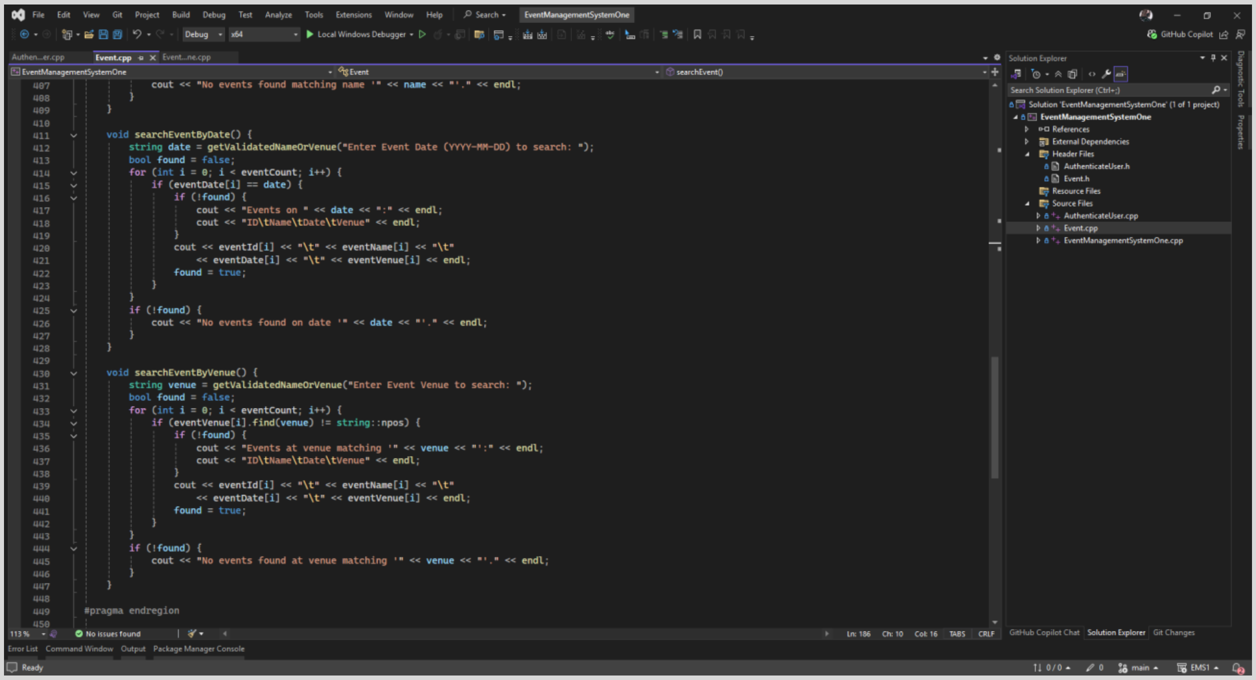


```cpp
void saveEvents() { ... }

string getValidatedNameOrVenue(const string& prompt) {
    string input;

    while (true) {
        cout << prompt;
        getline(cin, input);

        // Check if input is empty
        if (input.empty()) {
            cout << "Input cannot be empty. Please try again." << endl;
            continue;
        }

        // Remove leading/trailing spaces (optional but recommended)
        size_t start = input.find_first_not_of(" \t");
        size_t end = input.find_last_not_of(" \t");
        if (start == string::npos) {  // input was only spaces
            cout << "Input cannot be empty. Please try again." << endl;
            continue;
        }
        string trimmed = input.substr(start, end - start + 1);

        // Check if trimmed input is all digits (pure integer)
        bool allDigits = true;
        for (char ch : trimmed) {
            if (!isdigit(static_cast<unsigned char>(ch))) {
                allDigits = false;
                break;
            }
        }

        if (allDigits) {
            cout << "Input cannot be only numbers. Please enter a valid name or venue." << endl;
            continue;
        }

        // Input passes validation
        return trimmed;
    }
}

#pragma endregion
```

==



```cpp

    Export Functions

    private:

    Helper Functions

    #pragma region Search Functions

        void searchEventById() {
            int id = getValidatedIntInput("Enter Event ID to search: ");
            int idx = findEventIndexById(id);
            if (idx == -1) {
                cout << "No event found with ID " << id << "." << endl;
            }
            else {
                cout << "Event found:" << endl;
                cout << "ID: " << eventId[idx] << endl;
                cout << "Name: " << eventName[idx] << endl;
                cout << "Date: " << eventDate[idx] << endl;
                cout << "Venue: " << eventVenue[idx] << endl;
            }
        }

        void searchEventByName() {
            string name = getValidatedNameOrVenue("Enter Event Name to search: ");
            bool found = false;
            for (int i = 0; i < eventCount; i++) {
                if (eventName[i].find(name) != string::npos) {
                    if (!found) {
                        cout << "Matching events:" << endl;
                        cout << "ID\tName\tDate\tVenue" << endl;
                    }
                    cout << eventId[i] << "\t" << eventName[i] << "\t"
                        << eventDate[i] << "\t" << eventVenue[i] << endl;
                    found = true;
                }
            }
            if (!found) {
                cout << "No events found matching name '" << name << "'." << endl;
            }
        }

        void searchEventByDate() { ... }
```



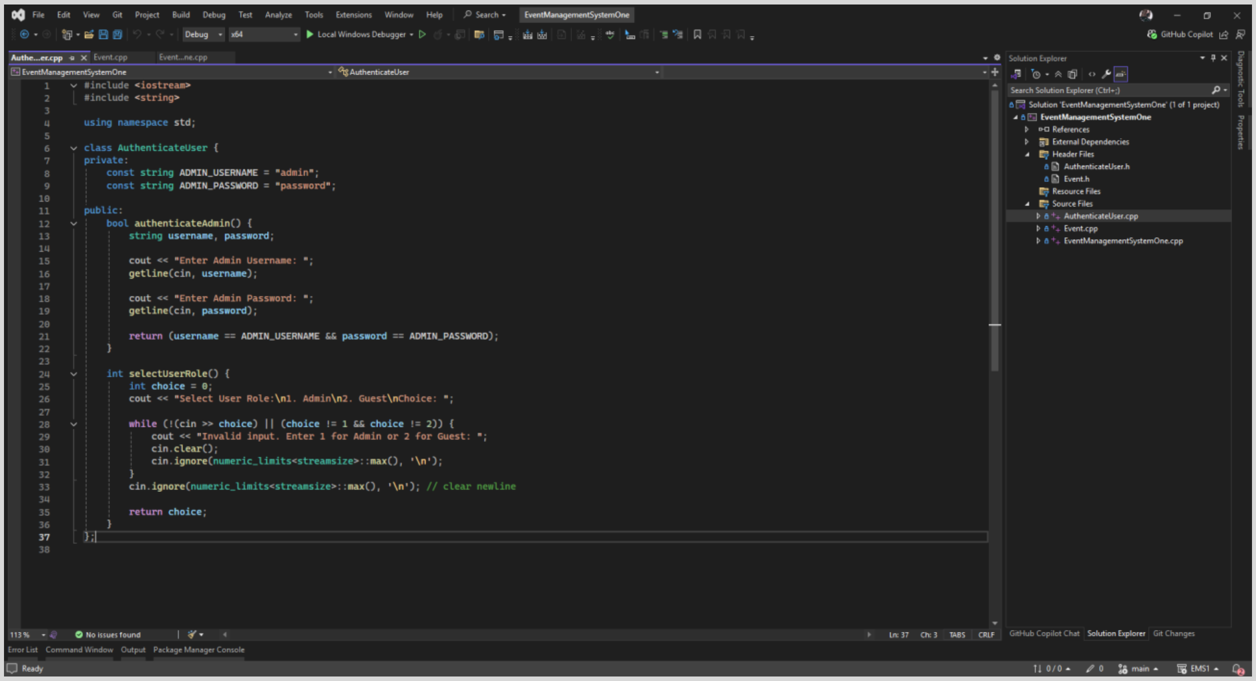
```cpp
                cout << "No events found matching name '" << name << "'." << endl;
            }
        }

        void searchEventByDate() {
            string date = getValidatedNameOrVenue("Enter Event Date (YYYY-MM-DD) to search: ");
            bool found = false;
            for (int i = 0; i < eventCount; i++) {
                if (eventDate[i] == date) {
                    if (!found) {
                        cout << "Events on " << date << ":" << endl;
                        cout << "ID\tName\tDate\tVenue" << endl;
                    }
                    cout << eventId[i] << "\t" << eventName[i] << "\t"
                        << eventDate[i] << "\t" << eventVenue[i] << endl;
                    found = true;
                }
            }
            if (!found) {
                cout << "No events found on date '" << date << "'." << endl;
            }
        }

        void searchEventByVenue() {
            string venue = getValidatedNameOrVenue("Enter Event Venue to search: ");
            bool found = false;
            for (int i = 0; i < eventCount; i++) {
                if (eventVenue[i].find(venue) != string::npos) {
                    if (!found) {
                        cout << "Events at venue matching '" << venue << "':" << endl;
                        cout << "ID\tName\tDate\tVenue" << endl;
                    }
                    cout << eventId[i] << "\t" << eventName[i] << "\t"
                        << eventDate[i] << "\t" << eventVenue[i] << endl;
                    found = true;
                }
            }
            if (!found) {
                cout << "No events found at venue matching '" << venue << "'." << endl;
            }
        }

    #pragma endregion

```

The above code implements four search methods for the searchEvent method in the system, each facilitating a different search criterion. The searchEventById method prompts for an event ID, validates it, and then searches the array; if found, it displays the event details, otherwise it reports no match. The searchEventByName, searchEventByDate, and searchEventByVenue methods similarly prompt for a string input and perform substring or exact matching on their respective fields. Each method outputs all matching events or informs the user if no matches are found. These methods enhance the system's usability by allowing flexible event lookup based on multiple attributes.

==

3. USER AUTHENTICATION

The AuthenticateUser class provides basic role-based access control for the event management system. It authenticates administrators by verifying entered credentials against predefined values and allows users to select their role as Admin or Guest with input validation. This ensures secure access, restricting privileged operations to authorized admins while permitting limited guest usage.



```cpp
#include <iostream>
#include <string>

using namespace std;

class AuthenticateUser {
private:
    const string ADMIN_USERNAME = "admin";
    const string ADMIN_PASSWORD = "password";

public:
    bool authenticateAdmin() {
        string username, password;

        cout << "Enter Admin Username: ";
        getline(cin, username);

        cout << "Enter Admin Password: ";
        getline(cin, password);

        return (username == ADMIN_USERNAME && password == ADMIN_PASSWORD);
    }

    int selectUserRole() {
        int choice = 0;
        cout << "Select User Role:\n1. Admin\n2. Guest\nChoice: ";

        while (!(cin >> choice) || (choice != 1 && choice != 2)) {
            cout << "Invalid input. Enter 1 for Admin or 2 for Guest: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // clear newline

        return choice;
    }
};
```

Links:

GitHub: https://github.com/M16A4PAPER/EMS1

LinkedIn: https://www.linkedin.com/posts/iqbal-hassan-63865b367_github-m16a4paperems1-project-event-management-activity-7333477591875526657-h02o?utm_source=share&utm_medium=member_desktop&rcm=ACoAAFsLGswBRcoGVz46P2rIQ9Utf5_XS-aoGyY

END OF DOCUMENT