VenueSender Codebase Overview

Created by Spencer Lievens

August 24, 2023

Contents

1 main.cpp

Source Code

```
#include "include/main.h"

using namespace confPaths;
using namespace std;

ConfigManager configManager;
CsvReader csvReader;
CurlHandleWrapper curlWrapper;
EmailManager emailManager;
EncryptionManager encryptionManager;
MenuManager menuManager;
VenueFilter venueFilter;
VenueUtilities venueUtilities;

#ifndef UNIT_TESTING

ConfigManager configManager;
CsvReader csvReader;
CurlHandleWrapper& curlWrapper = CurlHandleWrapper::getInstance();

int main() {
  // ... (truncated for brevity)
  return 0;
}
#endif // UNIT_TESTING

```

Explanation

- **Dependencies**: The main source file includes its corresponding header and makes use of two namespaces: ‘confPaths‘ and ‘std‘.
- **Global Objects**: Several global objects are declared and will be used across different parts of the code. These objects handle configuration management, CSV reading, cURL operations, email management, encryption, menu display, venue filtering, and venue utility operations.
- **Main Function** (`int main()`): The entry point to the program. The function does the following:
  - Initializes various variables and loads configurations from a JSON file.
  - Sets up and initializes CURL.
  - Reads venue data from a CSV file and extracts unique genres, states, cities, and capacities.
  - Contains the main loop to display menu options, filter venues based on user criteria, view and edit email settings, and send emails to selected venues.
  - Handles exit scenarios and cleanup tasks.
- **Conditional Compilation**: The main source file’s entire body uses an `#ifndef UNIT_TESTING` directive. This implies that if `UNIT_TESTING` is defined (likely during unit tests), the main function won’t compile. Excluding parts of code during unit testing is a common practice.

1.1 main.h

Header File

```
#ifndef MAIL_H
#define MAIL_H

#include "curl.h"
```

```
5     #include "fileutils.h"
6     #include "menu.h"
7     #include "errorhandler.h"
8     #include "structs.h"
9
10    #include <ctime>
11    #include <filesystem>
12    #include <regex>
13    #include <thread>
14
15    class EmailManager {
16      public:
17      static inline const int MAX_MESSAGE_LENGTH = 2000;
18      static inline const int MAX_SUBJECT_LENGTH = 50;
19      const size_t MAX_ATTACHMENT_SIZE = 24 * 1024 * 1024;
20
21      std::string getCurrentDateRfc2822();
22      std::string sanitizeSubject(std::string& subject);
23      void viewEmailSettings(bool useSSL, bool verifyPeer, bool verifyHost, bool verbose,
24      const std::string& senderEmail, int smtpPort, const std::string& smtpServer);
25      bool isValidEmail(const std::string& email);
26      void constructEmail(std::string &subject, std::string &message, std::string &
      attachmentPath,
27      std::string &attachmentName, std::string &attachmentSize, std::istream &in = std::cin);
28      bool sendIndividualEmail(CURL* curl,
29      const SelectedVenue& selectedVenue,
30      const std::string& senderEmail,
31      std::string& subject,
32      std::string& message,
33      const std::string& smtpServer,
34      int smtpPort,
35      std::string& attachmentName,
36      std::string& attachmentSize,
37      const std::string& attachmentPath);
38      void viewEmailSendingProgress(const std::string& senderEmail);
39    };
40
41    #endif // MAIL_H
42
```

Explanation

- **Header Guards**: `MAIL_H` is used as the header guard. This ensures that the header file's content is included only once in the compilation process.
- **Dependencies**: This header file includes dependencies related to CURL operations, file utilities, menu options, error handling, and venue structures. Additionally, there are standard library includes for time, file system operations, regular expressions, and threading.
- **EmailManager Class**:
  - **Constants**: Defines maximum lengths for the email message body and subject. There's also a constant for the maximum attachment size.
  - **Utility Functions**: Provides methods for getting the current date in RFC 2822 format, sanitizing the email subject, and displaying the email settings.
  - **Validation**: Contains a method to validate the format of an email address.
  - **Email Construction**: Function to construct the content of an email, including the subject, message, and attachment details.
  - **Email Sending**: Functions to send individual emails and view the progress of the sending process.

2 menu.cpp

Source Code

```
#include "include/menu.h"

using namespace std;

const int MenuManager::FILTER_BY_GENRE_OPTION = static_cast<int>(MenuManager::MenuOption::FilterByGenre);
const int MenuManager::FILTER_BY_STATE_OPTION = static_cast<int>(MenuManager::MenuOption::FilterByState);
const int MenuManager::FILTER_BY_CITY_OPTION = static_cast<int>(MenuManager::MenuOption::FilterByCity);
const int MenuManager::FILTER_BY_CAPACITY_OPTION = static_cast<int>(MenuManager::MenuOption::FilterByCapacity);

const int MenuManager::CLEAR_SELECTED_VENUES_OPTION = static_cast<int>(MenuManager::MenuOption::ClearSelectedVenues);
const intconst int MenuManager::CLEAR_BOOKING_TEMPLATE_OPTION = static_cast<int>(MenuManager::MenuOption::ClearBookingTemplate); MenuManager::VIEW_SELECTED_VENUES_OPTION = static_cast<int>(MenuManager::MenuOption::ViewSelectedVenues);
const int MenuManager::SHOW_EMAIL_SETTINGS_OPTION = static_cast<int>(MenuManager::MenuOption::ShowEmailSettings);
const int MenuManager::VIEW_EDIT_EMAILS_OPTION = static_cast<int>(MenuManager::MenuOption::ViewEditEmail);
const int MenuManager::VENUE_BOOKING_TEMPLATE_OPTION = static_cast<int>(MenuManager::MenuOption::VenueBookingTemplate);
const int MenuManager::EMAIL_CUSTOM_ADDRESS_OPTION = static_cast<int>(MenuManager::MenuOption::EmailCustomAddress);
const int MenuManager::FINISH_AND_SEND_EMAILS_OPTION = static_cast<int>(MenuManager::MenuOption::FinishAndSendEmail);
const int MenuManager::EXIT_OPTION = static_cast<int>(MenuManager::MenuOption::Exit);

bool MenuManager::isValidMenuChoice(int choice) {
  return choice >= static_cast<int>(MenuOption::FilterByGenre) &&
  choice <= static_cast<int>(MenuOption::Exit);
}

int MenuManager::displayMenuOptions() {
  #ifdef UNIT_TESTING
  // ... (truncated for brevity)
  #else
  // ... (truncated for brevity)
  #endif
}

void MenuManager::displaySelectedVenues(const vector<SelectedVenue>& selectedVenues) {
  // ... (truncated for brevity)
}

```

Explanation

- **Namespace Utilization**: The standard namespace (`std`) is used throughout the code.
- **Menu Constants Initialization**: Constants representing different menu options are initialized using static casts.
- **Menu Validation**: The ‘isValidMenuChoice‘ function checks whether a given choice falls within the range of valid menu options.
- **Menu Display**: The ‘displayMenuOptions‘ function handles the display of the main menu. Depending on whether `UNIT_TESTING` is defined, different behaviors are executed.

- **Selected Venues Display**: The ‘displaySelectedVenues‘ function shows the venues chosen by the user. If no venues are selected, an error message is displayed.

2.1 menu.h

Header File

This header file defines the `MenuManager` class responsible for managing menu-related operations within VenueSender.

Explanation

- **Header Guards**: `MENU_H` is used to prevent double inclusion.
- **Dependencies**:
  - `errorhandler.h`: For handling errors.
  - `fileutils.h`: Utility functions for file operations.
  - `structs.h`: Contains the structure definitions used in the program.
  - `iostream`: For standard input/output operations.
  - `vector`: For using the vector data structure.
- **Class Definition**:
  - `MenuManager`: A class dedicated to manage menu-related operations.
  - **MenuOption Enumeration**: Represents the available options in the main menu.
  - **Constants for Menu Options**: These constants are used to represent the different menu options available to the user.
  - **isValidMenuChoice(int choice)**: Validates if the user’s menu choice is within the range of valid options.
  - **displayMenuOptions()**: Displays the available menu options to the user and returns the user’s choice.
  - **displaySelectedVenues(const std::vector<SelectedVenue>& selectedVenues)**: Displays the list of venues that the user has selected.

3 filtercriteria.cpp

Source Code

```
#include "include/filtercriteria.h"

// Use the standard namespace
using namespace std;

// Utility function to convert a Venue object to a SelectedVenue object
SelectedVenue VenueUtilities::convertToSelectedVenue(const Venue& venue) {
  // ... (truncated for brevity)
}

// Utility function to get unique genres from a list of venues
set<string> VenueUtilities::getUniqueGenres(const vector<Venue>& venues) {
  // ... (truncated for brevity)
}

// Utility function to get unique states from a list of venues
set<string> VenueUtilities::getUniqueStates(const vector<Venue>& venues) {
  // ... (truncated for brevity)
```

```
}

// Utility function to get unique cities from a list of venues
set<string> VenueUtilities::getUniqueCities(const vector<Venue>& venues) {
  // ... (truncated for brevity)
}

// Utility function to get unique capacities from a list of venues
set<int> VenueUtilities::getUniqueCapacities(const vector<Venue>& venues) {
  // ... (truncated for brevity)
}

// Function to process venue selection based on user input
void VenueFilter::processVenueSelection(const vector<SelectedVenue>& temporaryFilteredVenues
,
vector<SelectedVenue>& selectedVenuesForEmail,
istream& input,
ostream& output) {
  // ... (truncated for brevity)
}

// Function to display filtered venues to the user
void VenueFilter::displayFilteredVenues(const vector<SelectedVenue>&
selectedVenuesForDisplay) {
  // ... (truncated for brevity)
}

// Common function for filtering venues by Genre, State, or City
vector<SelectedVenue> VenueFilter::filterByOptionCommon(const vector<Venue>& venues,
const set<string>& uniqueOptions,
const string& filterType,
vector<SelectedVenue>& temporaryFilteredVenues) {
  // ... (truncated for brevity)
}

// Function to filter venues by Genre, State, or City
vector<SelectedVenue> VenueFilter::filterByOption(const vector<Venue>& venues,
const string& filterType,
const set<string>& uniqueOptions,
vector<SelectedVenue>& temporaryFilteredVenues) {
  // ... (truncated for brevity)
}

// Function to filter venues by Capacity
vector<SelectedVenue> VenueFilter::filterByCapacity(const vector<Venue>& venues,
const set<int>& uniqueCapacities,
vector<SelectedVenue>& temporaryFilteredVenues) {
  // ... (truncated for brevity)
}

```

Explanation

- **Namespace Utilization**: The standard namespace (`std`) is used throughout the code.
- **VenueUtilities**: This class provides utility functions to convert between different types of venue objects, retrieve unique genre, state, city, and capacity values from a list of venues.
- **VenueFilter**: This class contains functions to filter and display venues based on different criteria such as genre, state, city, and capacity.
  - `processVenueSelection()`: Processes venue selection based on user input and updates the list of venues to be emailed.
  - `displayFilteredVenues()`: Displays the venues that match the applied filters.
  - `filterByOptionCommon()`: A common function used for filtering venues based on either genre, state, or city.

- `filterByOption()`: Filters venues by genre, state, or city.
- `filterByCapacity()`: Filters venues based on their capacity.

3.1 filtercriteria.h

Header File

```
#ifndef FILTERCRITERIA_H
#define FILTERCRITERIA_H

#include "fileutils.h"
#include "structs.h"

#include <iomanip>
#include <iostream>
#include <set>
#include <vector>

struct FilterCriteria {
  std::string genre;
  std::string state;
  std::string city;
};

class VenueUtilities {
  // ... (methods and members of VenueUtilities)
};

class VenueFilter {
  // ... (methods and members of VenueFilter)
};

#endif // FILTERCRITERIA_H

```

Explanation

- **Dependencies**: The header file includes dependencies related to file utilities, struct definitions, and various standard libraries like iostream, vector, set, and iomanip.
- **Struct Definition: FilterCriteria**:
  - Represents the filter criteria used to filter venues based on genre, state, and city.
- **Class Definition: VenueUtilities**:
  - A utility class that provides several static methods related to venues. It offers functions to extract unique genres, states, cities, and capacities from a list of venues. Additionally, there's a function to convert a 'Venue' object to a 'SelectedVenue' object.
- **Class Definition: VenueFilter**:
  - A class that manages venue filtering logic. It contains methods for processing user input related to venue selection, displaying filtered venues, filtering venues based on specific criteria like genre, state, city, and capacity. The class also maintains a list of venues selected for emailing.

4 structs.h

Header File

```
#ifndef STRUCTS_H
#define STRUCTS_H

#include <iostream>

struct Venue {
  std::string name;
  std::string email;
  std::string genre;
  std::string state;
  std::string city;
  int capacity;

  Venue() = default;
  Venue(const std::string& name, const std::string& email, const std::string& genre,
  const std::string& state, const std::string& city, int capacity)
  : name(name), email(email), genre(genre), state(state), city(city), capacity(capacity) {}
};

struct SelectedVenue {
  std::string name;
  std::string email;
  std::string genre;
  std::string state;
  std::string city;
  int capacity;

  SelectedVenue() = default;
  SelectedVenue(const std::string& name, const std::string& email, const std::string& genre,
  const std::string& state, const std::string& city, int capacity)
  : name(name), email(email), genre(genre), state(state), city(city), capacity(capacity) {}
};

#endif // STRUCTS_H

```

Explanation

- **Dependencies**: The header file includes the ‘iostream‘ library for string manipulation and input/output operations.
- **Struct Definition: Venue**:
  - Represents a venue with members like name, email, genre, state, city, and capacity.
  - **Default Constructor**: Allows creating uninitialized Venue objects.
  - **Parameterized Constructor**: Initializes all members of the struct.
- **Struct Definition: SelectedVenue**:
  - Represents a selected venue with the same members as the Venue struct.
  - **Default Constructor**: Allows creating uninitialized SelectedVenue objects.
  - **Parameterized Constructor**: Initializes all members of the struct.

5 mail.cpp

Source Code

```
  #include "include/mail.h"

  using namespace std;
  namespace fs = filesystem;

```

```
    CurlHandleWrapper& curlHandleWrapper = CurlHandleWrapper::getInstance();
    ErrorHandler errorHandler;

    static int successfulSends = 0; // Counter for successful email sends
    int totalEmails;

    string EmailManager::getCurrentDateRfc2822() {
      // Implementation to get the current date in RFC 2822 format...
    }

    string EmailManager::sanitizeSubject(string& subject) {
      // Implementation to sanitize the email subject...
    }

    void EmailManager::viewEmailSettings(bool useSSL, bool verifyPeer, bool verifyHost, bool verbose,
    const string& senderEmail, int smtpPort, const string& smtpServer) {
      // Implementation to display current email settings...
    }

    bool EmailManager::isValidEmail(const string& email) {
      // Implementation to check if a provided string is a valid email format...
    }

    void EmailManager::constructEmail(string &subject, string &message, string &attachmentName,
    string &attachmentSize, string &attachmentPath, istream &in) {
      // Implementation to guide the user in constructing an email...
    }

    void EmailManager::viewEditEmails(CURL* curl, const string& smtpServer, int smtpPort, const vector<SelectedVenue>& selectedVenuesForEmail, const string& senderEmail,
    string &subject, string &message, string& attachmentName, string& attachmentSize, string& attachmentPath, bool& templateExists) {
      // Implementation to view and edit emails...

    bool EmailManager::sendIndividualEmail(CURL* curl,
    const SelectedVenue& selectedVenue,
    const string& senderEmail,
    string& subject,
    string& message,
    const string& smtpServer,
    int smtpPort,
    string& attachmentName,
    string& attachmentSize,
    const string& attachmentPath,
    const vector<SelectedVenue>& selectedVenuesForEmail) {
      // Implementation to send an individual email...

    void EmailManager::createBookingTemplate(CURL* curl,
    const string& senderEmail,
    string& subject,
    string& message,
    const string& smtpServer,
    int smtpPort,
    string& attachmentName,
    string& attachmentSize,
    const string& attachmentPath,
    const vector<SelectedVenue>& selectedVenuesForEmail,
    bool& templateExists) {
      // Implementation to create an email from a booking template...

    void EmailManager::emailCustomAddress(CURL* curl,
    const std::string& senderEmail,
    std::string& subject,
    std::string& message,
    const std::string& smtpServer,
    int smtpPort,
    std::string& attachmentName,
```

```
71        std::string& attachmentSize,
72        std::string& attachmentPath) {
73          // Implementation to send an individual email to a custom address...
74
75        void EmailManager::confirmSendEmail(CURL* curl,
76        const vector<SelectedVenue>& selectedVenuesForEmail,
77        const string& senderEmail,
78        string& subject,
79        string& message,
80        const string& smtpServer,
81        int smtpPort,
82        string& attachmentName,
83        string& attachmentSize,
84        string& attachmentPath) {
85          // Implementation to confirm emails before sending...
86
87
```

Explanation

- **Namespace, Global Objects, and Counters**: This source file utilizes the standard namespace and a namespace alias for ‘filesystem‘. There are global objects for cURL handling and error handling. There are also global progress counters for successful email sends and total emails.
- **getCurrentDateRfc2822()**: Returns the current date in RFC 2822 format.
- **sanitizeSubject()**: Sanitizes the subject line of an email.
- **viewEmailSettings()**: Displays the current email settings.
- **isValidEmail()**: Validates whether the provided string conforms to a valid email format.
- **constructEmail()**: Guides the user in constructing an email, including subject, message, and optional attachment.
- **viewEditEmails()**: Views the current email (if one has been created) and allows the user to edit the email.
- **sendIndividualEmail()**: Sends an individual email to a selected venue based on specified configurations and updates the progress of email sending.
- **createBookingTemplate()**: Allows the user to create an email based off of a pre-designed booking template.
- **emailCustomAddress()**: Sends an individual email to a custom address based on specified configurations and updates the progress of email sending.
- **confirmSendEmail()**: Allows the user to view the email and confirm whether to send or return to the menu.

5.1 mail.h

Header File

```
1         #ifndef MAIL_H
2         #define MAIL_H
3
4         #include "curl.h"
5         #include "encryption.h"
6         #include "fileutils.h"
7         #include "menu.h"
8         #include "errorhandler.h"
9         #include "structs.h"
10
11        #include <ctime>
```

```
#include <filesystem>
#include <regex>
#include <thread>

// Class responsible for managing email-related operations
class EmailManager {
  public:
  // Maximum length for email message body
  static inline const int MAX_MESSAGE_LENGTH = 2000;

  // Maximum length for email subject
  static inline const int MAX_SUBJECT_LENGTH = 50;

  // Maximum attachment size (24 MB)
  const size_t MAX_ATTACHMENT_SIZE = 24 * 1024 * 1024;  // 24 MB in bytes

  // Function to get the current date in RFC 2822 format
  inline std::string getCurrentDateRfc2822() {
    char buffer[100];
    time_t now;
    struct tm *tm_now;
    time(&now);
    tm_now = gmtime(&now);
    strftime(buffer, sizeof buffer, "%a, %d %b %Y %H:%M:%S %Z", tm_now);
    return buffer;
  }

  // Function to sanitize the subject line of an email by replacing newline and carriage return characters with spaces
  inline std::string sanitizeSubject(std::string& subject) {
    std::string sanitized = subject;
    replace(sanitized.begin(), sanitized.end(), '\n', ' '); // replace newlines with spaces
    replace(sanitized.begin(), sanitized.end(), '\r', ' '); // replace carriage returns with spaces
    return sanitized;
  }

  // Function to display the email settings from the configuration file
  void viewEmailSettings(bool useSSL, bool verifyPeer, bool verifyHost, bool verbose,
  const std::string& senderEmail, int smtpPort, const std::string& smtpServer);

  // Function to validate an email address format
  bool isValidEmail(const std::string& email);

  // Function to construct an email, including the subject, message, and attachment details
  void constructEmail(std::string &subject, std::string &message, std::string &attachmentPath, std::string &attachmentName, std::string &attachmentSize, std::istream &in = std::cin);

  // Function to allow the user to modify the email
  void viewEditEmails(CURL* curl,
  const std::string& smtpServer,
  int smtpPort,
  const std::vector<SelectedVenue>& selectedVenuesForEmail,
  const std::string& senderEmail,
  std::string &subject,
  std::string &message,
  std::string& attachmentName,
  std::string& attachmentSize,
  std::string& attachmentPath,
  bool& templateExists);

  // Function to send an individual email to a selected venue
  bool sendIndividualEmail(CURL* curl,
  const SelectedVenue& selectedVenue,
  const std::string& senderEmail,
  std::string& subject,
```

```
75          std::string& message,
76          const std::string& smtpServer,
77          int smtpPort,
78          std::string& attachmentName,
79          std::string& attachmentSize,
80          const std::string& attachmentPath,
81          const std::vector<SelectedVenue>& selectedVenuesForEmail);
82
83          // Function to send a booking template email
84          void createBookingTemplate(CURL* curl,
85          const std::string& senderEmail,
86          std::string& subject,
87          std::string& message,
88          const std::string& smtpServer,
89          int smtpPort,
90          std::string& attachmentName,
91          std::string& attachmentSize,
92          const std::string& attachmentPath,
93          const std::vector<SelectedVenue>& selectedVenuesForEmail,
94          bool& templateExistss);
95
96          // Function to send to a custom email address
97          void emailCustomAddress(CURL* curl,
98          const std::string& senderEmail,
99          std::string& subject,
100         std::string& message,
101         const std::string& smtpServer,
102         int smtpPort,
103         std::string& attachmentName,
104         std::string& attachmentSize,
105         std::string& attachmentPath);
106
107         // Function to confirm the email before sending
108         void confirmSendEmail(CURL* curl,
109         const std::vector<SelectedVenue>& selectedVenuesForEmail,
110         const std::string& senderEmail,
111         std::string& subject,
112         std::string& message,
113         const std::string& smtpServer,
114         int smtpPort,
115         std::string& attachmentName,
116         std::string& attachmentSize,
117         std::string& attachmentPath);
118     };
119
120     #endif // MAIL_H
121
122
```

Explanation

- **Dependencies**: The header file includes dependencies related to cURL, file utilities, menu handling, error handling, console utilities, and data structures. Additionally, it utilizes the ‘ctime‘, ‘filesystem‘, ‘regex‘, and ‘thread‘ libraries.
- **Class Definition: EmailManager**:
  - **Constants**: The class defines constants for the maximum lengths of email messages and subjects, as well as the maximum size for attachments.
  - **getCurrentDateRfc2822()**: Returns the current date in the RFC 2822 format.
  - **sanitizeSubject()**: Sanitizes the subject line of an email.
  - **viewEmailSettings()**: Displays the email settings.
  - **isValidEmail()**: Checks if a given string is a valid email format.
  - **constructEmail()**: Guides the user in constructing an email with options for subject, message, and attachment.

- **viewEditEmails()**: Views the current email (if one has been created) and allows the user to edit the email.
- **sendIndividualEmail()**: Sends an individual email to a selected venue, updating email sending progress dynamically.
- **emailCustomAddress()**: Sends an individual email to a custom address based on specified configurations and updates the progress of email sending.
- **confirmSendEmail()**: Allows the user to view the email and confirm whether to send or return to the menu.

6 fileutils.cpp

Source Code

```
#include "include/fileutils.h"
#include "errorhandler.h" // Include due to circular dependency between fileutils.h and errorhandler.h

using namespace std;

namespace confPaths {
  string venuesCsvPath = "venues.csv";
  string configJsonPath = "config.json";
  string mockVenuesCsvPath = "src/test/mock_venues.csv";
  string mockConfigJsonPath = "src/test/mock_config.json";
}

string ConsoleUtils::trim(const string& str){
  const auto notSpace = [](int ch) {return !isspace(ch); };
  auto first = find_if(str.begin(), str.end(), notSpace);
  auto last = find_if(str.rbegin(), str.rend(), notSpace).base();
  return (first < last ? string(first, last) : string());
}

void ConsoleUtils::clearInputBuffer() {
  cin.clear();
  cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

void CsvReader::readCSV(vector<Venue>& venues, string& venuesCsvPath) {
  ifstream file(venuesCsvPath);
  if (!file.is_open()) {
    ErrorHandler errorHandler;
    errorHandler.handleErrorAndReturn(ErrorHandler::ErrorType::CONFIG_OPEN_ERROR, venuesCsvPath);
    return;
  }

  string line;
  getline(file, line); // Skip header

  while (getline(file, line)) {
    istringstream ss(line);
    string data;
    vector<string> rowData;
    while (getline(ss, data, ',')) {
      rowData.push_back(ConsoleUtils::trim(data));
    }

    if (rowData.size() == 6) {
      Venue venue(rowData[0], rowData[1], rowData[2], rowData[3], rowData[4], stoi(rowData[5]));
      venues.push_back(venue);
    } else {
      ErrorHandler errorHandler;
```

```
49            errorHandler.handleErrorAndReturn(ErrorHandler::ErrorType::INVALID_DATA_IN_CSV,
      venuesCsvPath);
50          }
51        }
52
53        file.close();
54      }
55
56      ConfigManager::ConfigManager() {}
57
58      bool ConfigManager::loadConfigSettings(bool& useSSL, bool& verifyPeer, bool& verifyHost,
      bool& verbose,
59      string& senderEmail, string& smtpUsername,
60      string& mailPass, int& smtpPort, string& smtpServer,
61      string& venuesCsvPath) {
62        // Implementation includes conditional compilation for unit testing,
63        // handles encryption and decryption of email passwords, and
64        // validates the loaded settings.
65      }
66
67      void ConfigManager::resetConfigFile() {
68        // Implementation reads the existing JSON configuration,
69        // modifies specific keys and values, and writes the updated
70        // JSON back to the file.
71      }
72
```

Explanation

- **Namespace** `confPaths`: This namespace holds configuration file paths.
- **ConsoleUtils::trim**: Function to remove leading and trailing whitespace from a given string.
- **ConsoleUtils::clearInputBuffer**: Function to clear the input buffer.
- **CsvReader::readCSV**: Function to read venue data from a CSV file.
- **ConfigManager::ConfigManager**: Default constructor for the ConfigManager class.
- **ConfigManager::loadConfigSettings**: Function to load the configuration settings. This implementation includes conditional compilation for unit testing, handles encryption and decryption of email passwords, and validates the loaded settings.
- **ConfigManager::resetConfigFile**: Function to reset the configuration file. This implementation reads the existing JSON configuration, modifies specific keys and values, and writes the updated JSON back to the file.

6.1 fileutils.h

Header File

```
1     #ifndef FILEUTILS_H
2     #define FILEUTILS_H
3
4     #include "encryption.h"
5     #include "structs.h"
6
7     #include <algorithm>
8     #include <fstream>
9     #include <limits>
10    #include <vector>
11    #include <jsoncpp/json/json.h>
12
13    class ErrorHandler;
14
```

```
15      namespace confPaths {
16        extern std::string venuesCsvPath;
17        extern std::string configJsonPath;
18        extern std::string mockVenuesCsvPath;
19        extern std::string mockConfigJsonPath;
20      }
21
22      class ConsoleUtils {
23        public:
24        static void clearInputBuffer();
25        static std::string trim(const std::string& str);
26      };
27
28      class CsvReader {
29        public:
30        static void readCSV(std::vector<Venue>& venues, std::string& venuesCsvPath);
31      };
32
33      class ConfigManager {
34        private:
35        EncryptionManager encryptionManager;
36
37        public:
38        ConfigManager();
39        bool loadConfigSettings(bool& useSSL, bool& verifyPeer, bool& verifyHost, bool& verbose,
40        std::string& senderEmail, std::string& smtpUsername,
41        std::string& mailPass, int& smtpPort, std::string& smtpServer,
42        std::string& venuesCsvPath);
43        static void resetConfigFile();
44      };
45
46      #endif // FILEUTILS_H
47
```

Explanation

- **Header Guards**: ‘FILEUTILS_H‘ is used to prevent the file from being included more than once in a single compilation.
- **Forward Declaration**: A forward declaration of ‘ErrorHandler‘ is made to resolve circular dependencies with ‘errorhandler.h‘.
- **Namespace** `confPaths`: This namespace holds string variables for configuration file paths.
- **ConsoleUtils Class**: A utility class with static functions related to console operations, such as clearing the input buffer and trimming strings.
- **CsvReader Class**: A class responsible for reading data from a CSV file.
- **ConfigManager Class**: A class responsible for managing the configuration settings. It has an instance of the ‘EncryptionManager‘ for password encryption and decryption. This class provides methods to load settings from a configuration file and reset the configuration file.

7 encryption.cpp

Source Code

```
1       #include "include/encryption.h"
2
3       using namespace std;
4
5       array<unsigned char, crypto_secretbox_KEYBYTES> globalEncryptionKey;
6       array<unsigned char, crypto_secretbox_NONCEBYTES> globalEncryptionNonce;
7
```

```
8     EncryptionManager::EncryptionManager() {
9       if (sodium_init() < 0) {
10        ErrorHandler errorHandler;
11        errorHandler.handleErrorAndReturn(ErrorHandler::ErrorType::LIBSODIUM_INIT_ERROR);
12        exit(EXIT_FAILURE);
13      }
14
15      randombytes_buf(globalEncryptionKey.data(), crypto_secretbox_KEYBYTES);
16      randombytes_buf(globalEncryptionNonce.data(), crypto_secretbox_NONCEBYTES);
17    }
18
19    bool EncryptionManager::encryptPassword(const string& decryptedPassword, string&
      encryptedPassword) {
20      unsigned char encryptedBuffer[crypto_secretbox_MACBYTES + decryptedPassword.size()];
21
22      if (crypto_secretbox_easy(encryptedBuffer, reinterpret_cast<const unsigned char*>(
      decryptedPassword.c_str()), decryptedPassword.size(),
23      globalEncryptionNonce.data(), globalEncryptionKey.data()) != 0) {
24        ErrorHandler errorHandler;
25        errorHandler.handleErrorAndReturn(ErrorHandler::ErrorType::
      EMAIL_PASSWORD_ENCRYPTION_ERROR);
26        return false;
27      }
28
29      string nonceStr(reinterpret_cast<const char*>(globalEncryptionNonce.data()),
      globalEncryptionNonce.size());
30      encryptedPassword = nonceStr + string(reinterpret_cast<const char*>(encryptedBuffer),
      sizeof(encryptedBuffer));
31
32      return true;
33    }
34
35    string EncryptionManager::decryptPassword(const string& encryptedPassword) {
36      const size_t NONCE_LENGTH = crypto_secretbox_NONCEBYTES;
37
38      if (encryptedPassword.size() < crypto_secretbox_MACBYTES + NONCE_LENGTH) {
39        ErrorHandler errorHandler;
40        errorHandler.handleErrorAndReturn(ErrorHandler::ErrorType::
      EMAIL_PASSWORD_ENCRYPTION_FORMAT_ERROR);
41        exit(EXIT_FAILURE);
42      }
43
44      string nonce = encryptedPassword.substr(0, NONCE_LENGTH);
45      string ciphertext = encryptedPassword.substr(NONCE_LENGTH);
46      unsigned char decryptedBuffer[ciphertext.size() - crypto_secretbox_MACBYTES];
47
48      if (crypto_secretbox_open_easy(decryptedBuffer,
49      reinterpret_cast<const unsigned char*>(ciphertext.c_str()),
50      ciphertext.size(),
51      reinterpret_cast<const unsigned char*>(nonce.c_str()),
52      globalEncryptionKey.data()) != 0) {
53        ErrorHandler errorHandler;
54        errorHandler.handleErrorAndReturn(ErrorHandler::ErrorType::
      EMAIL_PASSWORD_DECRYPTION_ERROR);
55        exit(EXIT_FAILURE);
56      }
57
58      return string(reinterpret_cast<char*>(decryptedBuffer), ciphertext.size() -
      crypto_secretbox_MACBYTES);
59    }
60
```

Explanation

- **globalEncryptionKey & globalEncryptionNonce**: Global arrays to store the encryption key and nonce.
- **EncryptionManager::EncryptionManager**: Default constructor for the EncryptionManager class. It initializes the libsodium library and generates a random encryption key and nonce.

- **EncryptionManager::encryptPassword**: Utility function that encrypts a given password using the global encryption key and nonce.
- **EncryptionManager::decryptPassword**: Utility function that decrypts an encrypted password using the global encryption key and nonce.

7.1 encryption.h

Header File

```
#ifndef ENCRYPTION_H
#define ENCRYPTION_H

#include "errorhandler.h"

#include <array>
#include <iostream>
#include <sodium.h>

extern std::array<unsigned char, crypto_secretbox_KEYBYTES> globalEncryptionKey;
extern std::array<unsigned char, crypto_secretbox_NONCEBYTES> globalEncryptionNonce;

class EncryptionManager {
  public:
  EncryptionManager();
  static bool encryptPassword(const std::string& decryptedPassword, std::string&
encryptedPassword);
  static std::string decryptPassword(const std::string& encryptedPassword);
};

#endif // ENCRYPTION_H

```

Explanation

- **globalEncryptionKey & globalEncryptionNonce**: Global arrays that store the encryption key and nonce, respectively.
- **EncryptionManager**: This class handles functionalities related to encryption.
- **EncryptionManager::EncryptionManager**: Constructor initializes the encryption key and nonce.
- **EncryptionManager::encryptPassword**: Utility function that encrypts a password. It takes in the original password to encrypt and stores the encrypted password in the given reference.
- **EncryptionManager::decryptPassword**: Utility function that decrypts a password. It takes in an encrypted password and returns the decrypted password as a string.

8 curl.cpp

Source Code

```
#include "include/curl.h"

using namespace std;

// CurlHandleWrapper Class Implementation
// ...

// Progress callback to show progress for email sending
int CurlHandleWrapper::progressCallback(...){...}
```

```
10
11      // Callback function to read data for sending
12      size_t CurlHandleWrapper::readCallback(...){...}
13
14      // Function to set SSL options
15      void CurlHandleWrapper::setSSLOptions(...){...}
16
17      // Constructor
18      CurlHandleWrapper::CurlHandleWrapper() {...}
19
20      // Destructor
21      CurlHandleWrapper::~CurlHandleWrapper() {...}
22
23      // Get the underlying cURL handle
24      CURL* CurlHandleWrapper::get() const {...}
25
26      // Global cURL initialization
27      void CurlHandleWrapper::init() {...}
28
29      // Global cURL cleanup
30      void CurlHandleWrapper::cleanup() {...}
31
32      // Setter for emailBeingSent
33      void CurlHandleWrapper::setEmailBeingSent(const string& email) {...}
34
35      // Getter for emailBeingSent
36      string CurlHandleWrapper::getEmailBeingSent() const {...}
37
38      // Clear the email being sent
39      void CurlHandleWrapper::clearEmailBeingSent() {...}
40
41      // Setup a cURL handle
42      CURL* setupCurlHandle(...) {...}
43
```

Explanation

- **CurlHandleWrapper Class**: This class provides a wrapper around the cURL handle to manage its behavior and settings.
- **progressCallback()**: A callback to show email sending progress.
- **readCallback()**: A callback function to read the data being sent.
- **setSSLOptions()**: Configures SSL options for the cURL session.
- **Constructor and Destructor**: Manages the lifecycle of the cURL handle.
- **get()**: Returns the underlying cURL handle.
- **init() and cleanup()**: Global initialization and cleanup functions for cURL.
- **setEmailBeingSent(), getEmailBeingSent(), and clearEmailBeingSent()**: Manage the email currently being sent.
- **setupCurlHandle()**: Configures a cURL handle with various settings.

8.1 curl.h

Header File

```
1       #ifndef CURL_H
2       #define CURL_H
3
4       #include "errorhandler.h"
5       #include <algorithm>
```

```
6       #include <cstring>
7       #include <iostream>
8       #include <mutex>
9       #include <curl/curl.h>
10
11      class CurlHandleWrapper {
12        public:
13        CurlHandleWrapper(const CurlHandleWrapper&) = delete;
14        CurlHandleWrapper& operator=(const CurlHandleWrapper&) = delete;
15
16        static CurlHandleWrapper& getInstance();
17        CURL* get() const;
18        static void init();
19        static void cleanup();
20        inline double getProgress() const;
21        int progressCallback(void* /*clientp*/, double dltotal, double dlnow, double /*ultotal*/
      , double /*ulnow*/);
22        void setEmailBeingSent(const std::string& email);
23        std::string getEmailBeingSent() const;
24        void clearEmailBeingSent();
25        void setSSLOptions(bool useSSL = true, bool verifyPeer = true, bool verifyHost = true);
26        static size_t readCallback(void* ptr, size_t size, size_t nmemb, void* userp);
27
28        private:
29        CurlHandleWrapper();
30        ~CurlHandleWrapper();
31        CURL* curl;
32        double progress{};
33        std::string emailBeingSent;
34        mutable std::mutex mtx;
35      };
36
37      CURL* setupCurlHandle(CurlHandleWrapper &curlWrapper, bool useSSL, bool verifyPeer, bool
      verifyHost, bool verbose,
38      const std::string& senderEmail, const std::string& smtpUsername, std::string&
      mailPassDecrypted, int smtpPort, const std::string& smtpServer);
39
40      #endif // CURL_H
41
```

Explanation

- **Dependencies**: The header file includes the ‘errorhandler.h‘ for error handling, standard libraries, the cURL library, and utilities for multi-threading.
- **Class Definition: CurlHandleWrapper**:
  - This class provides a Singleton pattern to ensure a single instance of the cURL handle.
  - The copy constructor and assignment operator are deleted to prevent copying.
  - **getInstance()**: Provides access to the single instance of this class.
  - **get()**: Returns the cURL handle.
  - **init() and cleanup()**: Static methods for initializing and cleaning up the cURL library.
  - **getProgress()**: Inline method to get the progress of the ongoing cURL operation.
  - **progressCallback()**: Callback to update the progress of ongoing operations.
  - **setEmailBeingSent(), getEmailBeingSent(), and clearEmailBeingSent()**: Manage the email currently being sent.
  - **setSSLOptions()**: Configures SSL options for the cURL session.
  - **readCallback()**: Callback function to read the email payload.
  - **setupCurlHandle()**: A utility function to set up and configure a cURL handle with the given options.

9 errorhandler.cpp

Source Code

```
#include "include/errorhandler.h"
#include "include/fileutils.h"

using namespace std;

std::mutex ErrorHandler::outputMutex;

bool ErrorHandler::handleCurlError(CURLcode res) {
  if (res != CURLE_OK) {
    cerr << "CURL Error: " << curl_easy_strerror(res) << endl;
    handleErrorAndReturn(ErrorType::LIBCURL_ERROR);
    return false;
  }
  return true;
}

void ErrorHandler::handleErrorAndReturn(ErrorType error) {
  handleErrorAndReturn(error, "");
}

void ErrorHandler::handleErrorAndReturn(ErrorType error, const string& extraInfo) {
  std::lock_guard<std::mutex> lock(outputMutex);
  switch (error) {
    // ... (All the error handling cases go here)
  }
}

```

Explanation

- **Dependencies**: This source file imports the error handling and file utilities header files for proper functioning.
- **Mutex Initialization**: A mutex is defined for thread safety. It ensures that multiple threads don't concurrently access the error handling function, which could lead to interleaved error messages.
- **handleCurlError()**: This function checks for any cURL-related errors. If one occurs, it displays a relevant error message and returns 'false'. Otherwise, it returns 'true'.
- **handleErrorAndReturn() (Overloaded)**: Handles various error types and displays appropriate error messages. One version accepts only the error type, while the overloaded version accepts an error type and additional information for a more detailed error message.
- **Error Handling**: A comprehensive switch-case block is used to handle different types of errors, each providing a specific error message to help in debugging and user feedback.

9.1 errorhandler.h

Header File

```
#ifndef ERRORHANDLER_H
#define ERRORHANDLER_H

#include <curl/curl.h>
#include <iostream>
#include <mutex>

class CsvReader;
class ConfigManager;
```

```
10
11      class ErrorHandler {
12        private:
13        static std::mutex outputMutex;
14        public:
15        enum class ErrorType {
16          // ... (All the error types go here)
17        };
18
19        void handleErrorAndReturn(ErrorType error);
20        void handleErrorAndReturn(ErrorType error, const std::string& extraInfo);
21        bool handleCurlError(CURLcode res);
22      };
23
24      #endif // ERRORHANDLER_H
25
```

Explanation

- **Dependencies**: The header file includes dependencies related to cURL, input/output streams, and mutex handling for thread safety.
- **Forward Declarations**: Forward declarations for ‘CsvReader‘ and ‘ConfigManager‘ are present due to circular dependencies between ‘fileutils.h‘ and ‘errorhandler.h‘.
- **Class Definition: ErrorHandler**:
  - **Mutex**: A static mutex ‘outputMutex‘ is defined to ensure thread-safe output when handling errors.
  - **ErrorType Enumeration**: Represents different types of errors that the system can encounter. This provides a clear structure to identify and handle different error scenarios.
  - **handleErrorAndReturn()**: Handles various types of errors and displays appropriate error messages. There are two versions of this method: one accepts only the error type, while the overloaded version also takes additional information to provide a more detailed error message.
  - **handleCurlError()**: Checks for any cURL-related errors and provides a feedback mechanism.

10 test_venuesender.cpp

Source File

```
1       // This file is used for testing DO NOT DELETE
2
3       #define CATCH_CONFIG_RUNNER
4       // ... (various includes)
5
6       using namespace std;
7
8       class CinGuard {
9         // ... (definition of CinGuard class)
10      };
11
12      class CoutGuard {
13        // ... (definition of CoutGuard class)
14      };
15
16      // Test groups and test cases follow...
17
18      int main( int argc, char* argv[] ) {
19        // ... (main function for running tests)
20      }
21
```

Explanation

- **Purpose**: This source file is dedicated to unit testing. It is important to note that deleting or altering this file could impact the quality assurance process for the software.
- **Dependencies**: The file includes dependencies to header files associated with various components of the system. It also uses the Catch2 testing framework, as indicated by the `CATCH_CONFIG_RUNNER` preprocessor directive.
- **Utility Classes**: ‘CinGuard‘ and ‘CoutGuard‘ classes are utility classes that help in redirecting the standard input and output streams. These are especially useful in testing scenarios where you want to simulate user input or capture output for verification.
- **Test Groups**: Multiple test groups are present in this source file, each focusing on a specific component or functionality of the system. These groups include ‘ConsoleUtils‘, ‘CsvReader‘, ‘ConfigManager‘, ‘MenuManager‘, ‘EmailManager‘, ‘VenueFilter‘, ‘VenueUtilities‘, and ‘EncryptionManager‘.
- **Test Cases**: Each test group contains various test cases, which are units of testing that ascertain the correctness of specific functionalities. For instance, the ‘ConsoleUtils‘ test group has a test case for verifying the ‘trim‘ function’s behavior.
- **Main Function**: At the end of the source file, a main function is defined to run all the test cases using the Catch2 framework. This main function also handles cleanup tasks after all tests have been executed.

11 venues.csv

This file represents a CSV (Comma-Separated Values) database of music venues. Each line in the CSV file corresponds to a single venue and its attributes.

Columns

The file is organized into six columns:

1. **Venue Name**: The name of the venue.
2. **Email**: The contact email address of the venue.
3. **Genre**: The primary genre of music played at the venue.
4. **State**: The state in which the venue is located.
5. **City**: The city in which the venue is located.
6. **Capacity**: The maximum number of people the venue can accommodate.

Data

Venue Name	Email	Genre	State	City	Capacity
Venue1	venue1@mock.com	all	AL	Daphne	100
Venue2	venue2@mock.com	all	UT	Provo	300

Explanation

- The file begins with a header row that lists the names of the columns.
- Each subsequent row provides details about a specific venue, with values separated by commas.
- For example, the ”Alabama Music Box” venue is located in Mobile, Alabama, can accommodate up to 700 people, and plays music of all genres.

12 config.json

This file contains the configuration settings related to the email sending functionality and paths to other resources.

Attributes

1. **email_pass_encrypted**: A boolean that indicates whether the email password is encrypted. (Do not change this value manually.)
2. **email_password**: The password of the sender's email account.
3. **sender_email**: The email address used to send the emails.
4. **smtp_port**: The port number used for the SMTP server connection.
5. **smtp_server**: The SMTP server address.
6. **smtp_username**: The username used for SMTP authentication.
7. **useSSL**: A boolean that indicates whether SSL should be used for the connection.
8. **venues_csv_path**: The path to the CSV file containing venue information.
9. **verbose**: A boolean that indicates whether to enable verbose logging.
10. **verifyHost**: A boolean that indicates whether to verify the host during the SMTP connection.
11. **verifyPeer**: A boolean that indicates whether to verify the peer during the SMTP connection.

Data Overview

Attribute	Example Value
email_pass_encrypted	false
email_password	”enter_email_password”
sender_email	”enter_your_sender_email”
smtp_port	587
smtp_server	”enter_your_smtp_server”
smtp_username	”enter_your_smtp_username”
useSSL	false
venues_csv_path	”venues.csv”
verbose	false
verifyHost	false
verifyPeer	false

Explanation

The ‘config.json‘ file is a configuration file that contains various settings related to the email sending process and paths to required resources. It's crucial to ensure the correct values are set for the email sending to work properly. Do not change the ‘email_pass_encrypted‘ manually as it is controlled by the application.