```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <limits>
#include <termios.h>
#include <unistd.h>
#include <vector>
#include <algorithm>
#include <SFML/Graphics.hpp>
#include <curl/curl.h>
#include <jsoncpp/json/json.h>

using namespace std;

struct MovieDetails {
    string title;
    string director;
    string poster;
    string link;
    string year;
    string rated;  // New field for "Rated"
    string releaseDate;
};

// Base class for handling HTTP requests
class HttpRequest {
protected:
    string url;
    string response;

    // Callback function to handle the response data
    static size_t WriteCallback(void* contents, size_t size, size_t nmemb,
string* response) {
        size_t totalSize = size * nmemb;
        response->append(static_cast<char*>(contents), totalSize);
        return totalSize;
    }

public:
    HttpRequest(const string& url) : url(url) {}

    virtual void sendRequest() = 0;

    string getResponse() const {
        return response;
```

```
    }
};

// Derived class for sending HTTP GET requests
class HttpGetRequest : public HttpRequest {
public:
    using HttpRequest::HttpRequest;

    void sendRequest() override {
        CURL* curlHandle = curl_easy_init();
        if (!curlHandle) {
            throw runtime_error("Failed to initialize cURL handle.");
        }

        // Set the URL and response callback
        curl_easy_setopt(curlHandle, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curlHandle, CURLOPT_WRITEFUNCTION,
WriteCallback);
        curl_easy_setopt(curlHandle, CURLOPT_WRITEDATA, &response);

        // Send the HTTP GET request
        CURLcode res = curl_easy_perform(curlHandle);

        // Check if the request was successful
        if (res != CURLE_OK) {
            curl_easy_cleanup(curlHandle);
            throw runtime_error("Failed to retrieve movie details. Error:
" + string(curl_easy_strerror(res)));
        }

        // Clean up cURL
        curl_easy_cleanup(curlHandle);
    }
};

// Class for printing search history
class SearchHistory {
public:
    void printHistory() {
        ifstream inputFile("output.txt");
        string line;

        cout << "Search History:" << endl;
        while (getline(inputFile, line)) {
            cout << line << endl;
```

```
        }

        inputFile.close();
    }
};

// Base class for movie search
class MovieSearch {
protected:
    string movieName;

public:
    MovieSearch(const string& movieName) : movieName(movieName) {}

    virtual void searchMovieDetails() = 0;
};

// Derived class for searching movie details using HTTP GET request
class HttpMovieSearch : public MovieSearch {
public:
    using MovieSearch::MovieSearch;

    void searchMovieDetails() override {
        try {
            // URL-encode the movie name
            CURL* curl = curl_easy_init();
            if (!curl) {
                throw runtime_error("Failed to initialize cURL.");
            }

            char* encoded = curl_easy_escape(curl, movieName.c_str(),
movieName.length());
            string encodedMovieName = encoded;
            curl_free(encoded);
            curl_easy_cleanup(curl);

            // Construct the URL for searching movie details
            string url =
"http://www.omdbapi.com/?i=tt3896198&apikey=322e4456&s=" +
encodedMovieName;

            // Create an instance of HttpGetRequest and send the request
            HttpGetRequest request(url);
            request.sendRequest();
```

```
            // Store the response in a file
            ofstream outputFile("output.txt", ios::app); // Open file in
append mode
            if (outputFile.is_open()) {
                outputFile << request.getResponse() << endl;
                outputFile.close();
                cout << "Output has been saved in the file 'output.txt'"
<< endl;
            } else {
                throw runtime_error("Failed to open the output file.");
            }

            // Parse the JSON response
            Json::Value root;
            Json::Reader reader;
            if (!reader.parse(request.getResponse(), root)) {
                throw runtime_error("Failed to parse the JSON response.");
            }

            // Extract movie details
            vector<MovieDetails> movies;
        const Json::Value& searchResults = root["Search"];
        if (!searchResults.empty()) {
            for (const Json::Value& result : searchResults) {
                MovieDetails movie;
                movie.title = result["Title"].asString();
                movie.director = result["Director"].asString();
                movie.poster = result["Poster"].asString();
                movie.link = result["Link"].asString();
                movie.year = result["Year"].asString();
                movie.rated = result["Rated"].asString();
                movie.releaseDate = result["Released"].asString();
                movies.push_back(movie);
            }
            } else {
                cout << "No movies found for the given search." << endl;
            }

            // Sort the movies based on title in ascending order
            sort(movies.begin(), movies.end(), [](const MovieDetails& a,
const MovieDetails& b) {
                return a.title < b.title;
            });
               // Print the poster links in the terminal
cout << "Poster Links:" << endl;
```

```
for (const MovieDetails& movie : movies) {
    cout << "Title: " << movie.title << endl;
    cout << "Poster: " << movie.poster << endl;
    cout << "If you want to review the poster, click the link." << endl;
    cout << endl;
}

            // Display the ordered movie details in a graphical interface
            sf::RenderWindow window(sf::VideoMode(1024, 768), "Movie
Search");
            sf::Font font;
            if
(!font.loadFromFile("/usr/share/fonts/truetype/msttcorefonts/arial.ttf"))
{
                throw runtime_error("Failed to load font file.");
            }

            sf::Text titleText;
            titleText.setFont(font);
            titleText.setCharacterSize(24);
            titleText.setFillColor(sf::Color::White);
            titleText.setPosition(10.f, 10.f);

            // ...

            while (window.isOpen()) {
                sf::Event event;
                while (window.pollEvent(event)) {
                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                window.clear(sf::Color::Black);

                float yPos = 10.f;
                for (const MovieDetails& movie : movies) {
                    sf::Text titleText;
                    titleText.setFont(font);
                    titleText.setCharacterSize(14);
                    titleText.setFillColor(sf::Color::Red);
                    titleText.setPosition(10.f, yPos);

                    titleText.setString("Title: " + movie.title);
                    window.draw(titleText);
```

```
                    yPos += 30.f;

                    sf::Text directorText;
                    directorText.setFont(font);
                    directorText.setCharacterSize(12);
                    directorText.setFillColor(sf::Color::White);
                    directorText.setPosition(10.f, yPos);

                    directorText.setString("Director: " + movie.director);
                    window.draw(directorText);
                    yPos += 25.f;

                    sf::Text posterText;
                    posterText.setFont(font);
                    posterText.setCharacterSize(8);
                    posterText.setFillColor(sf::Color::White);
                    posterText.setPosition(10.f, yPos);

                    posterText.setString("Poster: " + movie.poster);
                    window.draw(posterText);
                    yPos += 25.f;

                    sf::Text linkText;
                    linkText.setFont(font);
                    linkText.setCharacterSize(12);
                    linkText.setFillColor(sf::Color::White);
                    linkText.setPosition(10.f, yPos);

                    linkText.setString("Link: " + movie.link);
                    window.draw(linkText);
                    yPos += 25.f;

                    sf::Text yearText;
                    yearText.setFont(font);
                    yearText.setCharacterSize(12);
                    yearText.setFillColor(sf::Color::White);
                    yearText.setPosition(10.f, yPos);

                    yearText.setString("Year: " + movie.year);
                    window.draw(yearText);
                    yPos += 25.f;

                    sf::Text releaseText;
                    yearText.setFont(font);
                    yearText.setCharacterSize(12);
```

```
                    yearText.setFillColor(sf::Color::White);
                    yearText.setPosition(10.f, yPos);

                    yearText.setString("Release date: " +
movie.releaseDate);
                    window.draw(releaseText);
                    yPos += 25.f;




                    sf::RectangleShape separator(sf::Vector2f(1000.f,
1.f));
                    separator.setPosition(10.f, yPos);
                    separator.setFillColor(sf::Color::White);
                    window.draw(separator);
                    yPos += 10.f;
                }

                window.display();
            }
        } catch (const exception& ex) {
            cout << "Error: " << ex.what() << endl;
        }
    }


};

// Class to handle searching movies by genre
class MovieSearchByGenre  {
private:
    string genreName;


public:
    MovieSearchByGenre(const string& genreName) : genreName(genreName) {}

    void searchMoviesByGenre() {
        cout << "Searching movies of genre: " << genreName << endl;

        // Create an instance of HttpMovieSearch and search for movies of
the selected genre
        HttpMovieSearch movieSearch(genreName);
        movieSearch.searchMovieDetails();
```

```
    }
};

// Class to handle searching movies by actor name
class MovieSearchByActor {
private:
    string actorName;

public:
    MovieSearchByActor(const string& actorName) : actorName(actorName) {}


    void searchMoviesByActor() {
        cout << "Searching movies of actor: " << actorName << endl;

        // Create an instance of HttpMovieSearch and search for movies of
the selected actor
        HttpMovieSearch movieSearch(actorName);
        movieSearch.searchMovieDetails();

    }
};

// Class to handle the movie search application
class MovieSearchApp {
public:
    void run() {
        cout << "---------------------------------Welcome to the Movie
Search Application!---------------------------------" << endl;

        while (true) {
            cout << endl;
            cout << "Menu:" << endl;
            cout << "1. Search movies by name only" << endl;
            cout << "2. Search movies by genre" << endl;
            cout << "3. Search movies by actor name" << endl;
            cout << "4. View search history" << endl;
            cout << "5. Exit" << endl;
            cout << "Enter your choice: ";

            int choice;
            cin >> choice;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
            switch (choice) {
                case 1: {
                    cout << "Enter the name of the movie: ";
                    string movieName;
                    getline(cin, movieName);

                    cout << "Searching movie: " << movieName << endl;

                    // Create an instance of HttpMovieSearch and search
for the movie details
                    HttpMovieSearch movieSearch(movieName);
                    movieSearch.searchMovieDetails();


                    break;
                }
                case 2:
                    searchByGenre();
                    break;
                case 3:
                    searchByActorName();
                    break;
                case 4:
                    printSearchHistory();
                    break;
                case 5:
                    cout << "Thank you for using the Movie Search
Application!" << endl;
                    return;
                default:
                    cout << "Invalid choice. Please try again." << endl;
            }



        }
    }

    void searchByGenre() {
        cout << "Available Genres:" << endl;
        cout << "1. Animated" << endl;
        cout << "2. Drama" << endl;
        cout << "3. Action" << endl;
        cout << "Enter the number corresponding to the genre: ";
```

```
        int genre;
        cin >> genre;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        string genreName;
        switch (genre) {
            case 1:
                genreName = "animated";
                break;
            case 2:
                genreName = "drama";
                break;
            case 3:
                genreName = "action";
                break;
            default:
                cout << "Invalid genre. Please try again." << endl;
                return;
        }

        // Create an instance of MovieSearchByGenre and search for movies
of the selected genre
        MovieSearchByGenre searchByGenre(genreName);
        searchByGenre.searchMoviesByGenre();
    }

    void searchByActorName() {
        cout << "Enter the name of the actor: ";
        string actorName;
        getline(cin, actorName);

        // Create an instance of MovieSearchByActor and search for movies
of the selected actor
        MovieSearchByActor searchByActor(actorName);
        searchByActor.searchMoviesByActor();
    }

    void printSearchHistory() {
        SearchHistory history;
        history.printHistory();
    }
};
```

```
const string PASSWORD = "saim";




void SetEcho(bool enable) {
    struct termios tty;
    tcgetattr(STDIN_FILENO, &tty);

    if (enable) {
        tty.c_lflag |= ECHO;
    } else {
        tty.c_lflag &= ~ECHO;
    }

    tcsetattr(STDIN_FILENO, TCSANOW, &tty);
}
int main() {


    string enteredPassword;



    std::cout << "Enter the password to unlock the application: ";
    SetEcho(false);  // Disable echoing

    char ch;
    while ((ch = getchar()) != '\n') {  // Capture each character until
Enter (newline) is pressed
        enteredPassword.push_back(ch);
    }
    std::cout << std::endl;

    SetEcho(true);  // Enable echoing

    if (enteredPassword == PASSWORD) {
        cout << "Access granted! The application is now unlocked." <<
endl;

    // Initialize cURL
    curl_global_init(CURL_GLOBAL_DEFAULT);

    // Create an instance of the MovieSearchApp and run the application
    MovieSearchApp app;
```

```
    app.run();

    // Clean up cURL
    curl_global_cleanup();
    }
    else {
        cout << "Access denied! Incorrect password." << std::endl;

    }
    return 0;
}
```