```
#include <iostream>

#include <fstream>

#include <string>

#include <limits>

#include <vector>

#include <algorithm>

#include <curl/curl.h>

#include <jsoncpp/json/json.h>

#include <boost/asio.hpp>

#include <boost/bind.hpp>



using namespace std;

using boost::asio::ip::tcp;



struct MovieDetails {

    string title;

    string director;

    string poster;
    string link;

    string year;

};



class HttpRequest {

protected:

    string url;

    string response;



    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;

    }

};


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.");

        }


        curl_easy_setopt(curlHandle, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curlHandle, CURLOPT_WRITEFUNCTION, WriteCallback);

        curl_easy_setopt(curlHandle, CURLOPT_WRITEDATA, &response);



        CURLcode res = curl_easy_perform(curlHandle);



        if (res != CURLE_OK) {

            curl_easy_cleanup(curlHandle);

            throw runtime_error("Failed to retrieve movie details. Error: " + string(curl_easy_strerror(res)));

        }



        curl_easy_cleanup(curlHandle);

    }

};



class MovieSearch {

protected:

    string movieName;
```

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

    virtual void searchMovieDetails() = 0;
};

class HttpMovieSearch : public MovieSearch {
private:
    boost::asio::ip::tcp::socket socket;

public:
    HttpMovieSearch(boost::asio::ip::tcp::socket socket) : MovieSearch(""), socket(std::move(socket)) {}
    using MovieSearch::MovieSearch;

    void searchMovieDetails() override {
        try {
            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);


        string url = "http://www.omdbapi.com/?i=tt3896198&apikey=322e4456&s=" +
encodedMovieName;


        HttpGetRequest request(url);
        request.sendRequest();


        ofstream outputFile("output.txt", ios::app);
        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.");
}


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


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();

        movies.push_back(movie);

    }

} else {

    cout << "No movies found for the given search." << endl;

}



sort(movies.begin(), movies.end(), [](const MovieDetails& a, const MovieDetails& b) {

    return a.title < b.title;

});



cout << "Ordered Movie Details:" << endl;

for (const MovieDetails& movie : movies) {

    cout << "Title: " << movie.title << endl;

    cout << "Director: " << movie.director << endl;

    cout << "Poster: " << movie.poster << endl;

    cout << "Link: " << movie.link << endl;
```

```
                cout << "Year: " << movie.year << endl;

                cout << "------------------------" << endl;

            }

        } catch (const exception& e) {

            cerr << "Error: " << e.what() << endl;

            throw;

        }

    }

};



void printSearchHistory() {

    ifstream inputFile("output.txt");

    string line;



    cout << "Search History:" << endl;

    while (getline(inputFile, line)) {

        cout << line << endl;

    }
    inputFile.close();
}

class WebServer {
private:
    boost::asio::io_context ioContext;
    tcp::acceptor acceptor;

public:
    WebServer() : acceptor(ioContext, tcp::endpoint(tcp::v4(), 8888)) {}


    void start() {
        accept();
        ioContext.run();
    }
```

```
private:

    void accept() {

        acceptor.async_accept(

            [this](boost::system::error_code ec, boost::asio::ip::tcp::socket socket) {

                if (!ec) {

                    std::make_shared<HttpMovieSearch>(std::move(socket))->searchMovieDetails();

                }

                accept();

            });

    }

};



int main() {

    WebServer server;

    server.start();

    return 0;

}
```