AKADEMIA GÓRNICZO-HUTNICZA

Dokumentacja do projektu

Cinema booking system

z przedmiotu

Języki programowania obiektowego

Elektronika i Telekomunikacja PL, III rok

Andrzej Deja
Sebastian Krajewski

poniedziałek, 17:50

prowadzący: Rafał Frączek

27.01.2020

1. Opis projektu

Projekt implementuje klasy systemu rezerwacji biletów w kinie. Zapewnia on możliwość rezerwacji przez użytkowników, jak i pozwala dodawać i edytować: sale, filmy i seanse; w trybie administratora. Program wykorzystuje interfejs konsoli, co nie jest realistyczne. Aplikacja webowa z API i zewnętrzną bazą danych byłaby lepszym rozwiązaniem, jednak nie było to celem projektu.

2. Project description

The project implements cinema ticket booking system classes. It provides the option of booking by users, and allows you to add and edit: rooms, movies and screenings in administrator mode. The program uses the console interface, which is not realistic. A web application with an API and an external database would be a better solution, but it was not the goal of the project.

3. Instrukcja użytkownika

Wszystkie menu są opisane wewnątrz programu. Wszystkie operacje wejścia użytkownika są wykonywane przez std::cin, więc każdy wybór trzeba zatwierdzić enterem. Dotyczy to również poleceń "Kliknij/Naciśnij <znak>". Aby wybrać opcje z listy należy podać numer przy danej opcji.

Do projektu dołączone są przykładowe pliki tekstowe: movie.txt, order.txt, room.txt, track.txt, user.txt. Nie jest wymaganym korzystanie z ich zawartości, jednak program przy uruchomieniu potrzebuje ich obecności w folderze z plikiem .exe. Można usunąć ich zawartość.

Aby wejść w tryb administratora jako "numer" użytkownika należy podać słowo "admin", a jako hasło liczbę 1234. Przykładowy plik users.txt zawiera również użytkownika o numerze 1234 i haśle 1234.

Opcje menu użytkownika:

a - dodaj zamówienie

b - wyświetl zamówienia

exit - wyjdź

Opcje menu administratora:

a - dodaj film

b - zmień tytuł filmu

c - dodaj sale

d - edytuj sale

e - dodaj seans

f - edytuj seans

g - wyświetl filmy

h - wyświetl sale

i - wyświetl seanse

j - wyświetl zamówienia

save - zapisz stan

exit - wyjdź z trybu administratora

shutdown - zakończ program

4. Kompilacja

Projekt był kompilowany w Visual Studio 2015 (tj. kompilator Microsoft Visual C++ 14.0), przy standardowych ustawieniach, oprócz opcji dotyczącej ostrzeżeń. Został ustawiony poziom 5, czyli wszystkie.

W tych warunkach projekt zwraca ostrzeżenia o kodach:

C4514: unreferenced inline function has been removed (w plikach bobliotek) - nieużyte funkcje dostępne w bibliotece

C4710: function not inline (w pliku main.cpp) - wynika z decyzji kompilatora o nieużyciu przez niego funkcji jako inline

C4820: 'x' bytes padding added after data member (w plikach nagłówkowych klas) - wynika z wymiaru obiektu klasy lub ustawienia zmiennych klasy.

5. Pliki źródłowe

Projekt składa się z następujących plików źródłowych:

- *user.h*, user.*cpp* – deklaracja oraz implementacja klasy `User`,
- *movie.h*, movie.*cpp* – deklaracja oraz implementacja klasy `Movie`,
- *order.h*, order.*cpp* – deklaracja oraz implementacja klasy `Order`,
- *room.h*, room.*cpp* – deklaracja oraz implementacja klasy `Room`,
- *seat.h*, seat.*cpp* – deklaracja oraz implementacja klasy `Seat`,
- *track.h*, track.*cpp* – deklaracja oraz implementacja klasy `Track`,
- main.cpp - zawiera zmienne globalne, funkcje zapisu i odczytu z plików, funkcje obsługi uzytkowników i administratora, oraz logikę odpowiedzialną za ich wywoływanie i kontrolę dostępu.

6. Zależności

Brak, tylko STD i STL.

7. Opis klas

W projekcie utworzono następujące klasy:

- `Movie` – reprezentuje film.
  - `std::string Movie::getTitle()` – zwraca tytuł filmu,
  - `void Movie::modifyTitle(std::string)` – modyfikuje tytuł filmu.
- `Order` – reprezentuje zamówienie.
  - `int Order::getOrder()` – zwraca orderID,
  - `int Order::getTickets()` – zwraca ilość biletów,

- `int Order::getTrack()` – zwraca trackID,
- `int Order::getUser()` – zwraca userID,
- `void Order::setOrder(int)` – ustawia orderID,
- `void Order::setTickets(int)` – ustawia ilość biletów,
- `void Order::setTrack(int)` – ustawia trackID,
- `void Order::setUser(int)` – ustawia userID,
- `void Order::incTickets()` – inkrementuje ilość biletów,
- `void Order::summarize()` – wyświetla informacje o zamówieniu w jednej linii.

- `Room` – reprezentuje salę kinową.
  - `short Room::getColumns()` – zwraca liczbę miejsc w rzędzie,
  - `short Room::getRows()` – zwraca liczbę rzędów w sali,
  - `std::string Room::getName()` – zwraca nazwę sali,
  - `void Room::setColumns(short)` – ustawia liczbę miejsc w rzędzie,
  - `void Room::setRows(short)` – ustawia liczbę rzędów w sali,
  - `void Room::setName(std::string)` – ustawia nazwę sali,
  - `void Room::summarize()` – wyświetla informacje o sali w jednej linii.
- `Seat` – reprezentuje miejsce podczas seansu.
  - `int Seat::getUID()` – zwraca userID,
  - `void Seat::book(int)` – rezerwuje miejsce dla użytkownika,
  - `bool Seat::is_booked()` – zwraca czy miejsce jest zarezerwowane,
  - `void Seat::cancel()` – usuwa rezerwacje.
- `Track` – reprezentuje seans.
  - `int Track::getID()` – zwraca track_id,
  - `int Track::getMovie()` – zwraca movie_id,
  - `int Track::getRoom()` – zwraca room_id,
  - `char Track::getSeat(int)` – zwraca 'X' jeśli miejsce o danym numerze jest zarezerwowane, a w przeciwnym razie 'O',
  - `int Track::getSeatUID(int)` – zwraca userID, które rezerwuje miejsce,
  - `std::tm Track::getTime()` – zwraca czas w postaci struktury tm,
  - `std::string Track::getTimeStr()` – zwraca czas w postaci std::string do wyświetlenia,
  - `void Track::setMovie(int)` – ustawia movie_id,
  - `void Track::setRoom(int)` – ustawia room_id,
  - `void Track::setTime(std::tm)` – podmienia strukturę tm na daną w argumencie,
  - `void Track::bookSeat(int, int)` – rezerwuje miejsce o danym numerze użytkownikowi o danym ID,

- `void Track::clearSeats()` – czyści rezerwacje z wszystkich miejsc, nie usuwa rezerwacji z kont użytkowników,
- `void Track::summarize()` – wyświetla informacje o seansie w jednej linii.

- `User` – reprezentuje użytkownika.
  - `int User::get_ID()` – zwraca id użytkownika,
  - `int User::get_pass()` – zwraca hasło, aby można było, zapisać je do pliku, dla celów projektu. W gotowym produkcie powinno zostać zamienione na hash z solą i nie być przechowywane jako int.
  - `bool User::match(int)` – zwraca true, gdy hasło jest równe danemu intowi. Bezpieczniejsze niż przekazywanie wartości do porównania.

Wszystkie klasy zawierają odpowiednie konstruktory i destruktory.

8. Zasoby

W projekcie wykorzystywane są następujące pliki zasobów:

- `room.txt` – plik zawierający o salach kinowych. Struktura pliku:
  - druga linia: nazwa sali,
  - trzecia linia: ilość miejsc w rzędzie,
  - czwarta linia: ilość rzędów w sali,
  - w następnych liniach dane kolejnych sal w powyższej kolejności.
- `track.txt` – plik zawierający dane o seansach. Struktura pliku:
  - druga linia: id seansu,
  - trzecia linia: id filmu,
  - czwarta linia: id sali kinowej,
  - piata linia: czas w formacie HHMMddmmYYYY
  - w następnych liniach są zapisywane dane o rezerwacjach na seans:

    w pierwszej linii X lub O zależenie czy miejsce jest zarezerwowane

    w drugiej linii numer klienta (0 gdy miejsce nie jest zarezerwowane)

    ten schemat powtarza się dla każdego miejsca w sali
  - następnie cały schemat jest powtarzany na kolejnych liniach dla kolejnych seansów, oczywiście zachowując kolejność.
- `movie.txt` – plik zawierający o filmach. Struktura pliku:
  - druga linia: tytuł filmu,
  - w następnych liniach dane kolejnych filmów.
- `user.txt` – plik zawierający dane klientów. Struktura pliku:
  - druga linia: id użytkownika,
  - trzecia linia: hasło użytkownika,
  - w następnych liniach dane kolejnych klientów w powyższej kolejności.

- `order.txt` – plik zawierający dane o zamówieniach. Struktura pliku:
  - druga linia: id zamówienia,
  - trzecia linia: id użytkownika,
  - czwarta linia: id seansu,
  - piąta linia: ilość zamówionych biletów,
  - w następnych liniach dane kolejnych zamówień w powyższej kolejności.

Pierwsza linia w każdym pliku jest pusta.

9. Dalszy rozwój i ulepszenia

Można:

Poprawić kontrolę danych wprowadzanych przez użytkownika.

Podzielić kod na mniejsze funkcje i przenieść część kodu z pliku main.cpp na model klasowy.

Dodać więcej opcji dla administratora dotyczących kontroli kont i zamówień.

Dodać możliwość zmiany hasła, przechowywać hasła jako hashe z solą.

Kolejne zmiany polegałyby na przejściu na model z użyciem baz danych, a następnie podział na aplikacje webową z API do bazy danych i klientem, najlepiej w postaci strony internetowej, wysyłającym zapytania.

10. Inne

brak