Sprawozdanie Jan Naklicki

Created By

Fly-us.com

Cele projektu

Celem projektu było stworzenie programu obsługującego sprzedaż biletów lotniczych.

Szczegóły techniczne

Projekt został wykonany przy pomocy języka c++ i jego biblioteki sqlite3 pozwalającej na korzystanie z baz danych SQL.

Zmiany w trakcie projektu

Początkowo projekt miał pozwalać na edycję biletów, osób kupujących, pracowników, lotów, lotnisk, samolotów oraz bagaży. Jednak z racji podobnego charakteru tych rzeczy zdecydowałem na zmniejszenie ilość obsługiwanych rzeczy do dwóch: bilet(zawierają zarówno dane osób jak i cechy szczególne dla biletu) oraz lot.

Uruchamianie programu

Aby uruchomić program należy zainstalować bibliotekę sqlite3 i uruchomić polecenie:

```
g++ main.cpp Database.cpp Interface.cpp -lsqlite3 -o program -std=c++11 && ./program
```

Druga możliwość to uruchomić gotowy program

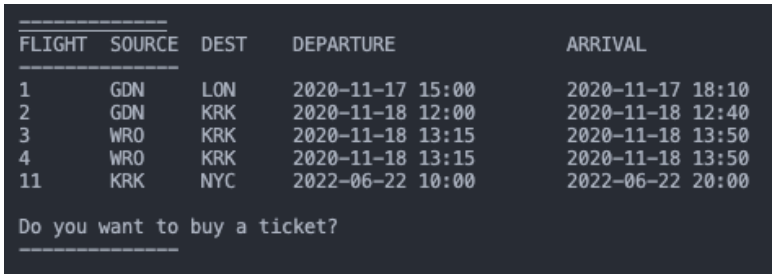
```
./program
```

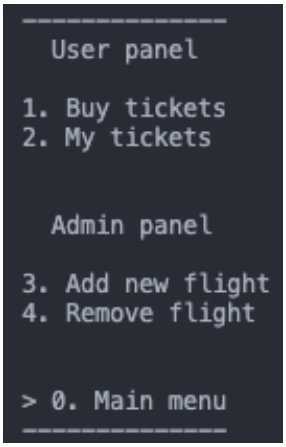
Interface

Interface graficzny to zwykłe wyświetlanie danych w konsoli w przejrzysty dla użytkownika sposób. Z racji ograniczeń bazy danych część rzeczy wyświetlana jest poza interface-m, z braku możliwości zrobienia tego w inny sposób.

W programie wyróżniamy:

- Główne menu
  - Użytkownik
    - Wyświetlanie biletów
      - kupowanie biletów
    - Kupione bilety
  - Admin
    - Dodawanie lotów
    - Usuwaniem lotów





Klasy

Interface

```
public:
    // UI elements
    void displayInterface(string content);
    void displayHeader(string content);
    void displaySpacer(string above, string below);
    // Pages
    void mainMenu();
    void searchFlight();
    void myTickets();
    void addNewFlight();
    void removeFlight();
};
```

W interface wyróżniamy dwie główne działy:

- UI
- Pages

UI zawiera funkcje pomagające wyświetlane danych. W Pages zawierają się funkcje wyświetlające konkretne widoki, w których wczytywane są dane do dalszej obsługi programu.

Database

```
class Database
{
private:
    sqlite3 *DB;
    int exit = sqlite3_open("FlyUs.sqlite", &DB);
    char *messaggeError;
    static string selectedTable;
    //Database callbacks
    static int displayFlights(void *data, int argc, char **argv, char **azColName);
    static int displayTickets(void *data, int argc, char **argv, char **azColName);

public:
    static string queryForDisplay;
    void createDatabase();
    void queryFlights();
    void queryTickets(string name, string surname);
    void insertTicket(string values);
    void insertFlight(string values);
    void removeFlight(string value);
    // Helpers
    string generateRandomSeat();
    string appendNewValue(string values, bool last, bool date);
};
```

Database zwiera funkcje wywoływane w interface, które wykonują operacje na bazie danych. Przykładowo metody z insert wykonują operacje “INSERT”, która dodaje rekord do tablicy.

Są też funkcje pomocnicze takie jak appendNewValue która, tworzy string, który zostanie wklejony np. do funkcji insert tickets.

Callbacks:

- **displayFlights()** - funkcja wywoływana jako callback po wykonaniu zapytania do bazy danych, odebrane dane wyświetla w formie tabeli
- **displayTickets()** - działa tak samo jak displayFlights tylko wyświetla bilety

Obsługa bazy danych:

- **createDatabase()** - tworzy pustą bazę danych
- zapytania wykorzystujące SELECT:
  - **queryFlights()** - funkcja wyświetla wszystkie rekordy w tabel FLIGHT
  - **queryTickets() -** funkcja wybiera wszystkie rekordy z połączonych ze sobą tabel FLIGHT i TICKET gdzie podane imię i nazwisko w tabeli pasują do podanych na wejściu.
- zapytania wykorzystujące INSERT:
  - **insertTicket()** - funkcja wstawia rekord do bazy danych z informacjami o kupującym zebranymi w warstwie interface
  - **insertFlight()** - funkcja tworzy lot, który później może zostać wykorzystany do zakupu biletów
- zapytania wykorzystujące DELETE:
  - **removeFlight()** - funkcja usuwa lot o podanym przez użytkownika ID

Helpers:

- generateRandomSeat() - funkcja losuje liczbę z przedziału 1-800, która jest później wykorzystywana jako siedzenie do zakupionego biletu
- appendNewValue() - funkcja pomocnicza, która z podanych wartości tworzy format danych, które są wymagane w kwerendach SQL.

Baza danych

W bazie znajdują się dwie tabele:

Flight:

Reset Filters Records: 5 Search 5 re

	ID	AIRPORT_START_ID	AIRPORT_DESTINATION_ID	TAKEOFF_DATE	ARRIVAL_DATE
	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	GDN	LON	2020-11-17 15:00	2020-11-17 18:10
2	2	GDN	KRK	2020-11-18 12:00	2020-11-18 12:40
3	3	WRO	KRK	2020-11-18 13:15	2020-11-18 13:50
4	4	WRO	KRK	2020-11-18 13:15	2020-11-18 13:50
5	11	KRK	NYC	2022-06-22 10:00	2022-06-22 20:00

Każdy rekord przedstawia lot, w którego skład wchodzą:

- ID
- AIRPORT_START_ID - id lotniska, z którego startuje samolot
- AIRPORT_DESTINATION_ID - id lotniska docelowego
- TAKEOFF_DATE - data i godzina wylotu
- ARRIVAL_DATE - data i godzina lądowania

Ticket:

	ID	FLIGHT_ID	NAME	SURNAME	SEAT	CLASS
	Search column...	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	1	JANEK	NAKLICKI	446	1
2	2	3	Jan	Krak	695	2
3	3	4	Kamil	Krzak	799	2
4	4	2	Kasia	Szpak	782	2
5	9	1	Kamil	Szpak	98	2
6	10	1	Marek	Szpak	338	1
7	11	3	Jan	Szpak	777	2
8	12	3	Andrzej	Kowal	474	2
9	13	2	Jan	Naklicki	484	2

Każdy rekord przedstawia bilet, w którego skład wchodzą:

- ID
- FLIGHT_ID - obcy klucz lotu, po którym możemy łączyć tablę
- NAME - imię pasażera
- SURNAME - nazwisko pasażera
- SEAT - siedzenie w samolocie
- CLASS - klasa przelotu

Podsumowanie

Projekt jest przykładem tego, że nawet w małym projekcie warto jest używać większych bibliotek, które pomagają przy tworzeniu programów. Sqllite3 okazał się być bardzo wygodną rzeczą, która bardzo uprościła pracę z danymi, co pozwoliło na szybkie

rozwijanie programu o dodatkowe funkcje, nie zmuszając do zamian w warstwie danych.