Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ПРИЛОЖЕНИЕ «Chess»**

БГУИР КП 1-40 01 01 110 ПЗ

Студент: гр. 851001 Листратенко И.В.

Руководитель: Шостак Е.В.

Минск 2019

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc27332588)

[1 АНАЛИЗ АНАЛОГОВ ПРОГРАММНОГО СРЕДСТВА 7](#_Toc27332589)

[1.1 Система бронирования «Aviasales» 7](#_Toc27332590)

[1.2 Веб приложение «Ticketsby» 8](#_Toc27332591)

[1.3 Веб-приложение «Skyscanner» 8](#_Toc27332592)

[2 ЦЕЛИ И ЗАДАЧИ 10](#_Toc27332593)

[2.1 Цель курсовой работы 10](#_Toc27332594)

[2.2 Задачи курсовой работы 10](#_Toc27332595)

[2.3 Необходимые ресурсы для курсовой работы 10](#_Toc27332596)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 12](#_Toc27332597)

[3.1 Интерфейс программного средства 12](#_Toc27332598)

[3.2 Загрузка и хранение списков 14](#_Toc27332599)

[3.2 Разработка регистрации и входа 16](#_Toc27332600)

[3.3 Разработка списков 17](#_Toc27332601)

[3.4 Разработка отмены последнего действия 18](#_Toc27332602)

[3.5 Структура входных и выходных данных 20](#_Toc27332603)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 22](#_Toc27332604)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 23](#_Toc27332605)

[5.1 Системные требования 23](#_Toc27332606)

[5.2 Установка программы 23](#_Toc27332607)

[5.3 Инструкция по использованию 23](#_Toc27332608)

[ЗАКЛЮЧЕНИЕ 29](#_Toc27332609)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc27332610)

[Приложение А. Исходный код программы 3](#_Toc8769804)1



# ВВЕДЕНИЕ

В последние несколько десятков лет технический прогресс изменил жизнь всех людей. Развитие технологий ускорило темп исследований в различных областях науки и позволило улучшить качество жизни. Большое влияние развитие информационных технологий оказало и на различные системы бронирования.

Глобальное развитие систем бронирования связано с ростом спроса на туристические поездки. В настоящее время можно забронировать, как номер в гостинице, железнодорожные и авиабилеты, так и билеты в театр, заказ трансфера, такси, цветов и прочих дополнительных услуг.

В наше время в сфере реализаций туристических услуг активно используется компьютерная техника. Сейчас уже трудно представить, как могло производиться бронирование отелей, железнодорожных или авиабилетов различными туристическими агентами и компаниями с учетом всевозможных скидок при отсутствии систем бронирования.

Процесс завоевания туристических агентств системами компьютерного бронирования авиакомпаний проходил быстрыми темпами и к 1982 г. почти 82% туристических агентств США были подключены к основным системам. Ведущими системами с самого начала являлись Sabre, принадлежащая "Американ эйрлайнз" и Apollo, принадлежащая "Юнайтед эйрлайнз", на долю которых в начале 80-х годов приходилось 41% и 39% рынка соответственно. Высокая экономическая эффективность использования систем компьютерного бронирования побудила их владельцев бороться за влияние на рынке туристических услуг. Для того чтобы привлечь туристических агентов, компании расширили сферу применения систем бронирования, включив в их программу широкий круг услуг (бронирование гостиничных мест, прокат автомобилей, страхование авиапассажиров, оформление билетов, заграничных паспортов и даже заказов на цветы). Кроме того, программы предоставляют агентам возможность компьютерной обработки документации и бухгалтерского учета.

Подобная компьютеризация деятельности туристических агентств существенно повысила их производительность. В настоящее время подавляющее большинство продаваемых авиабилетов на регулярные рейсы, реализуется через агентства, подключенные к крупной компьютерной системе бронирования.

Гостиницы, подключившиеся к глобальным сетям бронирования, имеют возможность внести в их банк данных информацию о самой гостинице, видах номеров, их описание и цены. Это позволяет использовать информацию десяткам тысяч агентствам по всему миру.

США являются первой страной, в которой стали внедряться компьютерные системы бронирования, и именно в США эти системы достигли наивысшего на сегодняшний день развития. Следом за США системы стали возникать в других странах. В 1987 г. были созданы два консорциума Galileo и Amadeus.

Первая национальная автоматизированная система резервирования авиационных билетов «Сирена» была разработана учеными АН СССР и специалистами Минприбора в 1972 году. Она обеспечивает минимальное время заказов и оформления авиабилетов. В 1999 году появилась система бронирования авиабилетов «Сирена-3».

Целью данного курсового проекта является разработка системы бронирования авиарейсов «MziliAvia».

# АНАЛИЗ АНАЛОГОВ ПРОГРАММНОГО СРЕДСТВА

В настоящее время существует огромное количество систем бронирования авиарейсов. В течение нескольких десятилетий появилось множество различных версий данного приложения. В данном разделе будут рассмотрены аналоги создаваемого приложения.

## Система бронирования «Aviasales»

«Aviasales» – одна из самых популярных систем бронирования рейсов для русскоязычного населения. Приложение позволяет находить самые дешевые варианты авиарейсов и подбирать варианты перелетов с учетом всех необходимых пересадок. Большим плюсом данного приложения является наличие мобильной версии. Стоит отметить удобство и простоту пользовательского интерфейса. Внешний вид данного приложения представлен на рисунке 1.1.

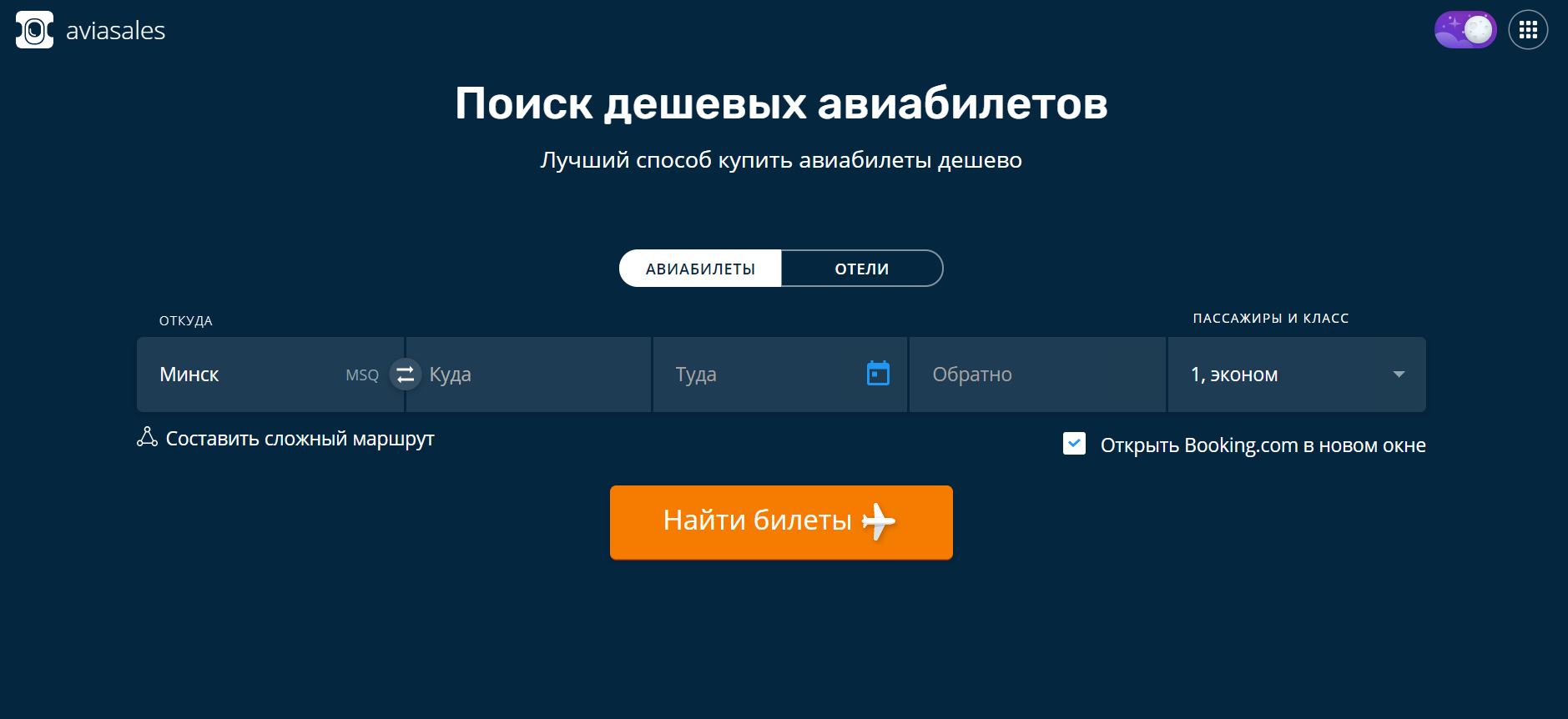


Рисунок 1.1 – Интерфейс приложения «Aviasales»

Существенным недостатком данной версии, является отсутствие десктопной версии приложения. Еще одним минусом является сотрудничество с ограниченным количеством авиакомпаний. В данном приложении у администратора отсутствует возможность редактирования авиарейсов и пользователей. Еще одной недоработкой данного приложения можно обозначить отсутствие возможности отмены последнего действия.

## 1.2 Веб приложение «Ticketsby»

«Ticketsby» – система бронирования авиарейсов, ориентированная на белорусского пользователя. Главным плюсом данного приложения является выбор билетов не только на авиарейсы, но и на железнодорожные билеты, автобусы. Предлагается широкий выбор отелей по выбранным направлениям.

Существует система лояльности для постоянных клиентов данного сервиса.

Основным минусом приложения является отсутствие десктопной и мобильной версии приложения. Как и в большинстве подобных приложений, присутствует привязка к туристическим компаниям, с которыми сотрудничает данный ресурс, из-за чего сильно ограничены возможные направления передвижения.

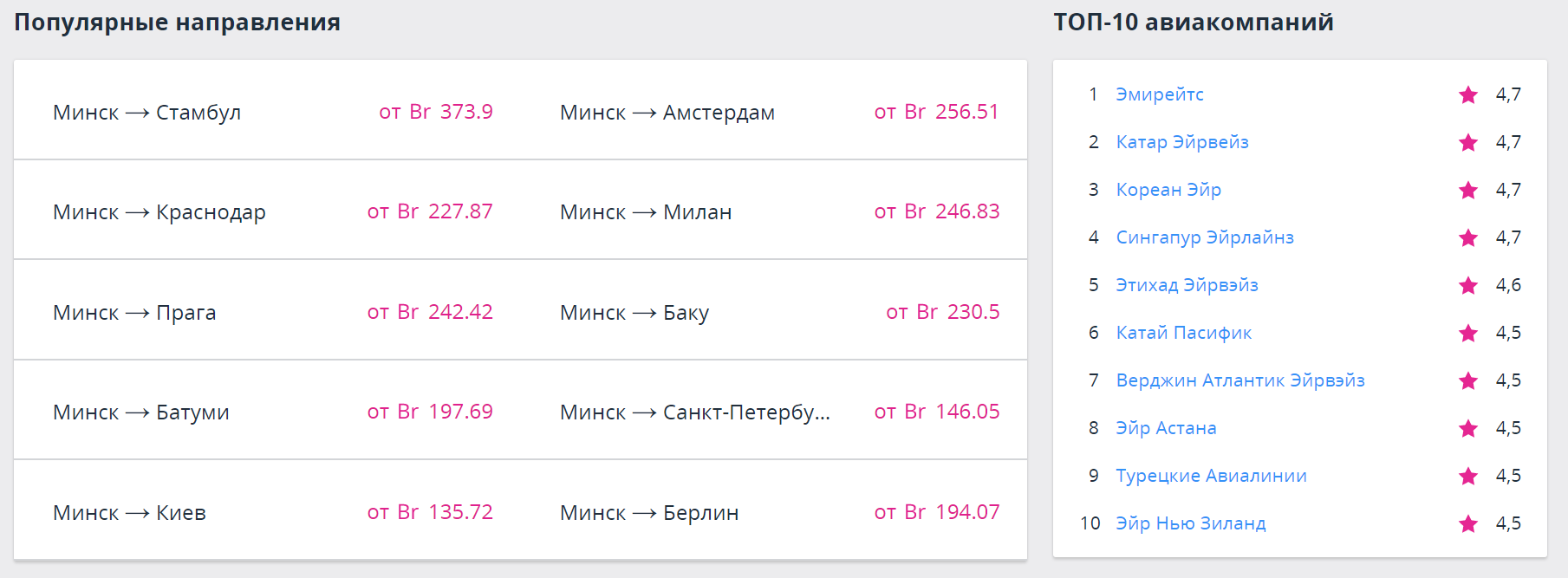
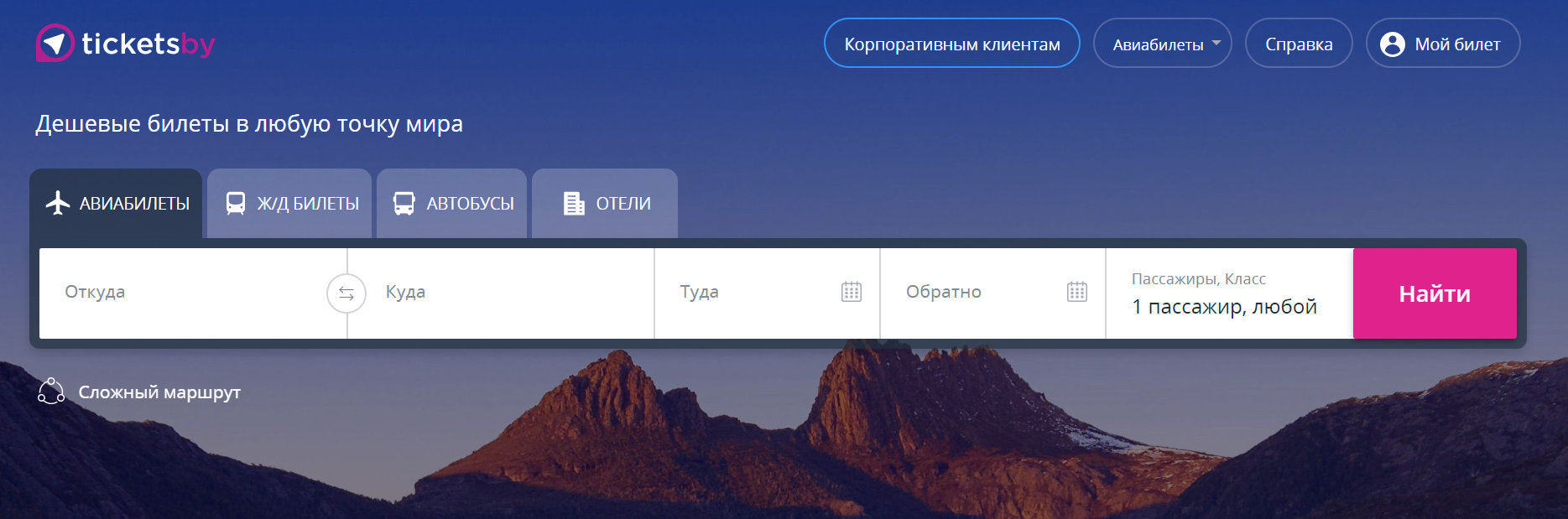


Рисунок 1.2 – Интерфейс веб-приложения «Ticketsby»

## 1.3 Веб-приложение «Skyscanner»

«Skyscanner» – одна из самых популярных русскоязычных систем бронирования авиарейсов. Приложение предоставляет пользователю выгодные предложения на каждый месяц и систему скидок. Присутствуем возможность забронировать туры по популярным направлениям. Кроме авиабилетов, «Skyscanner» предлагает выбор отелей, прокат автомобилей.

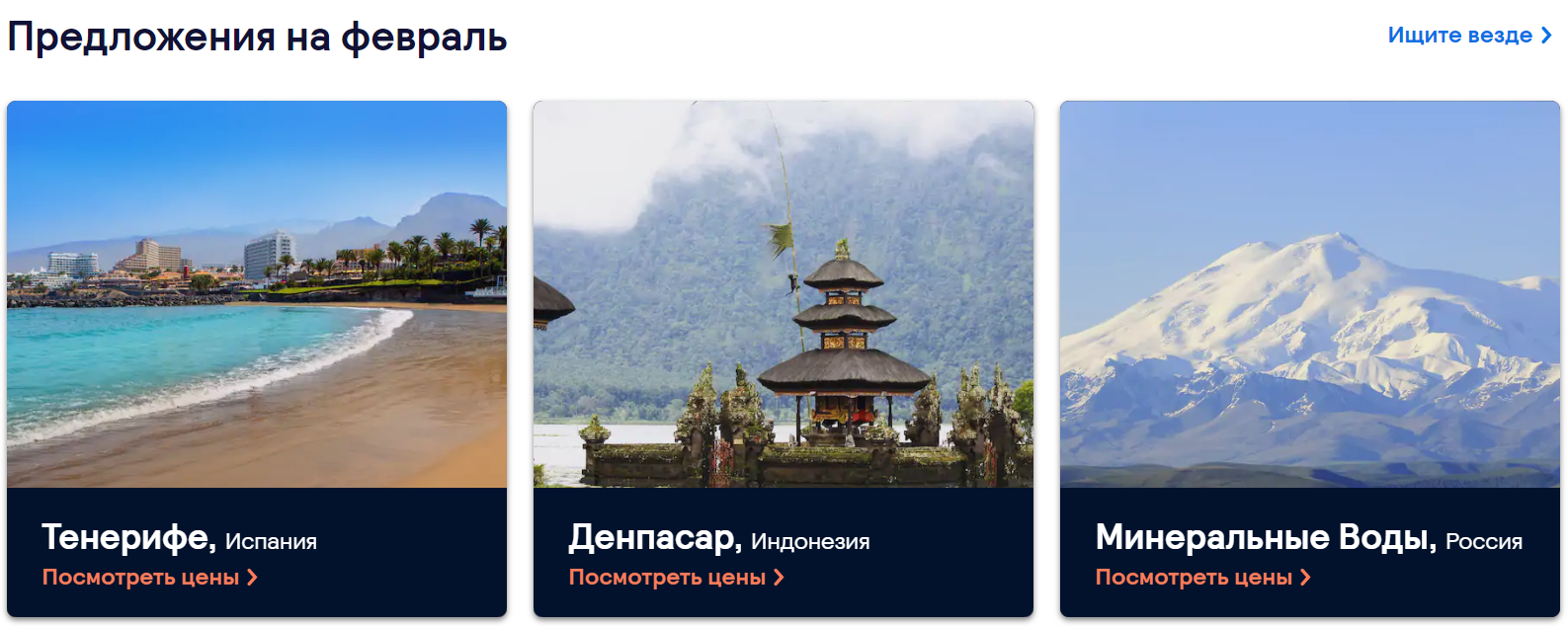
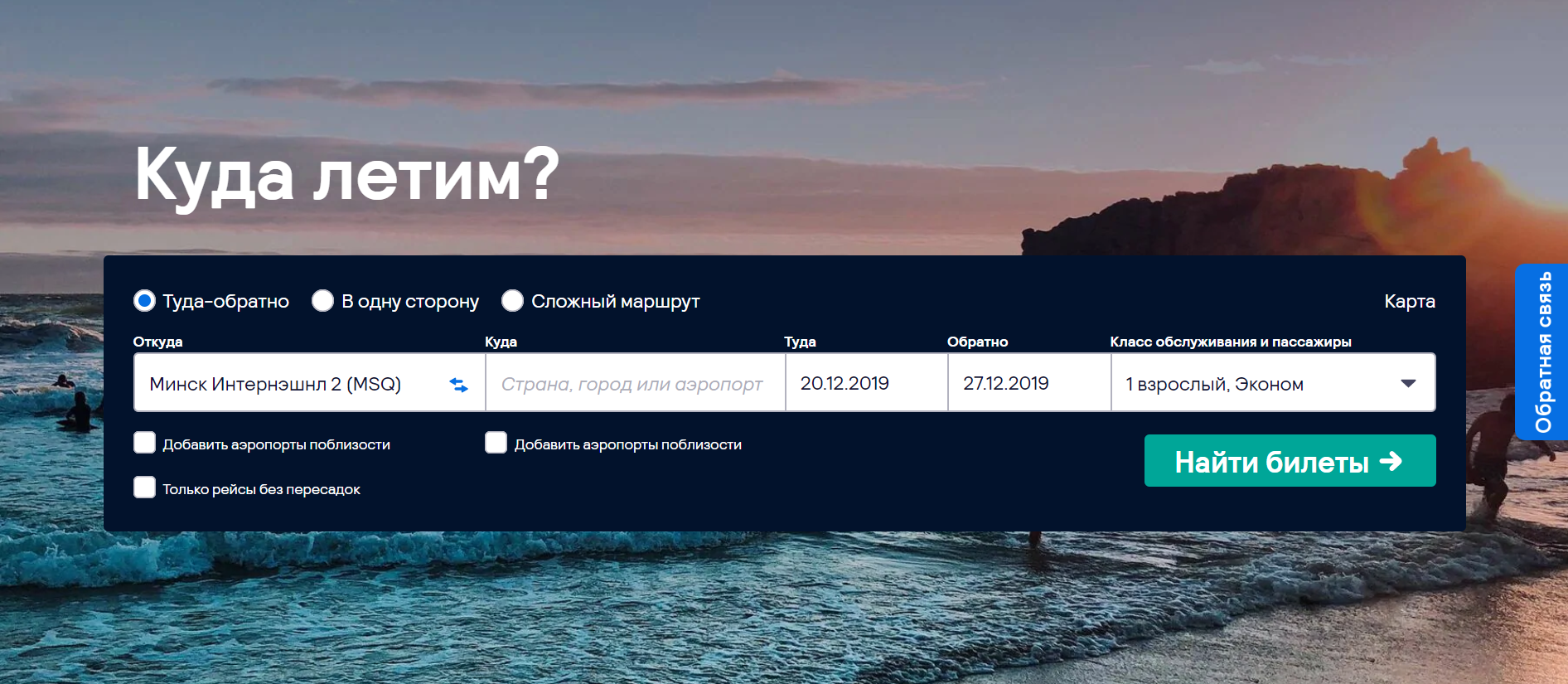


Рисунок 1.3 – Интерфейс приложения «Skyscanner»

У приложения «Skyscanner» отсутствует административная часть. Новые авиарейсы добавляются автоматически, только по предварительной договоренности с авиакомпаниями-партнерами, администратор может просматривать и редактировать только список брони. Отсутствуют десктопная и мобильная версии приложения. Данный сервис не позволяет забронировать билеты на конкретный рейс, предлагая только туры по выбранному направлению.

# ЦЕЛИ И ЗАДАЧИ

## Цель курсовой работы

Целью данной курсовой работы является создание системы бронирования авиарейсов «MziliAvia», адаптированного под запуск на современных операционных системах, в частности, версиях ОС Windows 8 и старше, на компьютерах с 4К-разрешением.

## Задачи курсовой работы

В результате сравнения аналогов программного средства и анализа предметной области в данном курсовом проекте поставлены следующие задачи:

1. разработать:
   1. систему бронирования авиарейсов «MziliAvia»;
   2. систему регистрации и проверки пользователей;
   3. алгоритм бронирования рейса;
   4. алгоритм отмены последнего действия;
2. создать:
   1. эстетичный и понятный пользователю интерфейс;
   2. динамические компоненты для хранения информации.

У пользователя есть возможность искать, бронировать рейсы, просматривать уже забронированные билеты. У администратора – просматривать и редактировать пользователей и рейсы, удалять уже существующие и добавлять новые.

## 2.3 Необходимые ресурсы для курсовой работы

Для разработки программного средства будет использоваться язык программирования C++.Это компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

**C++** широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр).

Среда разработки – Qt Creator 4.10.1. Это кроссплатформенная свободная IDE для разработки на С, С++ и QML. Использование данной среды разработки дает множество возможностей в создании оконных приложений Windows. С помощью данной среды можно разработать качественное оконное приложение, обеспечить удобные средства для ввода, редактирования, сохранения и отображения данных. Также Qt Creator предлагает широкие возможности при работе с графикой, что позволяет создать привлекательный и удобный для пользователя интерфейс.

# 3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Интерфейс программного средства

Интерфейс обеспечивает комфортное взаимодействие между пользователем и приложением. При его разработке следует учитывать эргономику и устройства, с какими будет взаимодействовать приложение. Комфорт и удобство пользователя должны быть главным критерием в построении интерфейса и создании дизайна.

Проанализировав палитру компонентов среды Qt Creator, был сделан вывод, что имеющиеся визуальные компоненты не подходят для организации работы приложения из-за отсутствия возможности детальной настройки и несоответствия стилю программы. Поэтому появилась необходимость разработать собственный способ визуального представления данных. Наиболее целесообразным стало использование QLabel (для использования изображений) и вставок CSS в строенные компоненты Qt Creator. Это позволило создать более эстетичный внешний вид игрового приложения.

Однако использование для ввода данных полей, которые отличаются от стандартных элементов управления, используемых в Windows, вносит дисбаланс во внешний вид программного средства. В связи с этим возникает идея спроектировать дизайн необходимых элементов управления таких, как кнопки вручную, с использованием графического редактора. Несмотря на высокую трудоемкость, этот способ позволяет избежать шаблонности дизайна, а также приобрести опыт разработки внешнего вида элементов управления с нуля. Примеры персонажей иллюстрированы на рисунке 3.1.



Рисунок 3.1 – Элементы интерфейса, созданные в редакторе Adobe Photoshop

Для создания собственных элементов управления используется графический редактор Adobe Photoshop, предоставляющий широкие возможности для работы с графикой.

В создаваемом приложении присутствует потребность использовать многочисленные картинки, так как необходимо создать эстетичный интерфейс. Было принято решение загрузить необходимые изображения в ресурсы проекта, так как это обеспечивает высокую скорость загрузки по сравнению с загрузкой из внешнего файла. Все изображения хранятся в ресурсном файле «Res.grc». Ниже приведена процедура загрузки из ресурсов программы в компонент QLabel.

// Объявление объекта типа QLabel

QLabel ImageLabel;

// Объявление объекта типа QPixmap и загрузка изображениия

QPixmap picture( ":/images/planeS.png" );

// Установка изображения в QLabel

ImageLabel.setPixmap( picture );

// Установка размеров QLabel под размер изображения

ImageLabel.resize(picture.size());

// Вывод QLabel на экран

ImageLabel.show();

Анимация изображений происходит с помощью константно заданной матрицы, в которой хранится траектория перемещения (координаты x и y). Объекты двигаются по одной траектории, различие достигается за счет изменения координаты «y», различного для каждого объекта, и начального индекса. Благодаря этому, изображения двигаются на разной высоте и в каждый момент времени находятся в различных точках траектории. Анимированные объекты продемонстрированы на рисунке 3.2.

Рисунок 3.2 – Анимированные изображения

Перемещение изображения происходит по срабатыванию таймера. Для качественной отрисовки и плавного движения анимации опытным путем было установлено время срабатывания таймера (140мс). На рисунке 3.3 представлена блок-схема процедуры срабатывания таймера.



Рисунок 3.3 – Блок-схема процедуры «TimeFunc»

## 3.2 Загрузка и хранение списков

Для хранения и создания списков рейсов, пользователей и брони используется шаблонный двусвязный список. Данная структура хранения данных была выбрана из-за ряда преимуществ перед динамическими и статическими массивами. Для связных списков операции вставки и удаления принадлежат к классу O(1). Независимо от текущего элемента спуска и его емкости, для вставки или удаления элемента всегда требуется одно и то же время. Основным недостатком связных списков является то, что получение доступа к их элементам принадлежит к классу О(n). Однако была произведена перегрузка оператора «квадратные скобки», то позволяет обращаться к элементам списка так же, как к элементам массива. Шаблонный список был выбран для того, чтобы не создавать три различных структуры. В данном проекте над списком будет определено только три операции: удаление, добавление и поиск объекта. Ниже приведена структура шаблонного списка:

class DoublyLinkedList

{

//поля размера и указателей на голову и следующий элемент

int Size;

Node<T> \*head;

Node<T> \*tail;

}

На основе шаблонного списка строятся 3 основных: список пользователей, всех рейсов и забронированных. В связи с этим, необходимо реализовать достаточную основу для дальнейшей работы со списками. Для главного списка создан класс «DoublyLinkedList». Дня него перегружены операторы «<<», «>>», «==», «!=», «[ ]», что значительно упрощает работу со списком (например запись и чтение из файла, обращение к конкретному элементу, сравнение). Ниже приведен код некоторых функций, определенных над шаблонным списком:

public:

// конструкторы класса

DoublyLinkedList();

DoublyLinkedList(const DoublyLinkedList &);

DoublyLinkedList(DoublyLinkedList &&);

~DoublyLinkedList();

// функция возвращающая размер

int size() const;

// функция проверяющая список на пустоту

bool empty()const;

// процедура удаляющая список

void delete\_all();

// процедура удаляющая звено списка

void delete\_node(Node <T>\*);

// процедура вставки звена

void insert\_node(const T& val, int index);

// процедура поиска по списку

\_Iterator<T> search(const T& date);

// процедуры снятия первого и последнего элемента списка

void pop\_back();

void pop\_front();

// процедуры добавления в начало и конец списка

void push\_front(const T & val);

void push\_back(const T & val);

// счетчики, указывающие на начало и конец списка для

// поиска

\_Iterator<T> begin();

\_Iterator<T> end();

## 3.2 Разработка регистрации и входа

При запуске приложения пользователю необходимо авторизоваться. Для этого необходимо ввести Логин и Пароль. Возможна регистрация пользователя. Окно, появляющееся при запуске приложения продемонстрированно на рисунке 3.4.

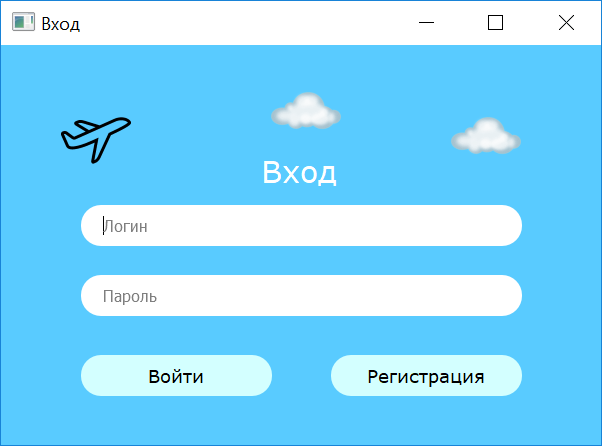


Рисунок 3.4 – Демонстрация окна входа

Основные задачи данного этапа разработки:

1. разработать алгоритм проверки пользователя для входа;
2. разработать алгоритм для регистрации пользователя;
3. разработать систему обработок ошибок при входе.

Процедуры, реализующие эти задачи вынесены в класс «User», от которого наследуются классы «Admin» и «Reservation». В данном классе присутствует три основных поля для хранения информации ниже приведены поля данного класса:

// поле для хранения имени

std::string name = "";

// поле для хранения пароля

std::string password = "";

//поле, определяющее режим доспупа

// 1 – администратор, 0 – обычный пользователь

bool access\_mode = false;

Ниже приведен листинг процедур и функций определенных над объектами класса User:

1. void SaveUser() – процедура сохраняющая пользователя в файл;
2. bool GetMode() – функция возвращающая режим доступа объекта;
3. bool CheckUser() – функция проверяющая существование объекта;
4. bool CheckUserForReg() – функция проверяющая возможна ли регистрация;
5. string GetName() – функция возвращающая имя пользователя;
6. string GetPassword() – функция возвращающая пароль пользователя;
7. void SetName(string \_name) – функция устанавливающая имя пользователя;
8. void SetPassword(string \_password) – функция устанавливающая пароль пользователя;
9. void SetMode(string \_password) – функция устанавливающая режим доступа пользователя;
10. bool operator !=(user & obj) – перегрузка операции «!=»;
11. friend bool operator == (const user user\_1, const user user\_2) – перегрузка операции «==»;
12. inline friend ofstream& operator<<(ofstream& out, user& us) – перегрузка операции «<<» для записи в файл.

## 3.3 Разработка списков

Для удобства вся информация вся многократно используемая информация была вынесена в отдельный класс «Data». В нем реализованы основные операции над списками (удаление, добавление, запись и чтение из файла). Ниже приведен код удаления выбранного элемента списка:

void UserEditingFrame::on\_DeleteUser\_clicked()

{

// создание объекта класса Data

Data TempUserList;

// инициализация списка пользователей

TempUserList.InitializateUserList();

// вычисление индекса удаляемого объекта

int Index = TempUserList.GetUserIndex(NameEdit.text());

// если пользователь найден

if (Index != -1){

// удаляем пользователя из списка

TempUserList.UserList.remove(Index);

// открываем поток для записи в файл

ofstream Users;

Users.open("Users.bin", std::ios::out);

for ( int i = 0; i < UserList.size()- 1; i++){

// записываем в файл с помощью перегруженного оператора

// «<<»

Users << TempUserList.UserList[i].GetName() << " ";

Users << TempUserList.UserList[i].GetPassword() << " ";

Users << TempUserList.UserList[i].GetMode() << " ";

}

Users.close();

// обновляем таблицу пользователей

RefreshTable();

}

}

Перегрузка операторов, это всего лишь более удобный способ вызова функций, что значительно упрощает работу с объектами класса и сокращает количество кода. Компилятор сам определяет какую функцию выбрать в зависимости от сигнатуры. Ниже приведен код перегрузки оператора «==»:

bool operator == (const user user\_1, const user user\_2){

// сравниваем имена пользователей

if (user\_1.name == user\_2.name){

// сравниваем пароли

if (user\_1.password != user\_2.password){

// если пароль не сошелся

return false;

}

}else{

return false;

}

// если сошелся пароль и имя объекты класса «user»

// равны

return true;

}

На основе данных списков реализованы процедуры бронирования, редактирования, добавления и удаления авиарейсов. На основе двух стеков реализована процедура отмены последнего действия, блок-схема которой представлена на рисунке 3.5. При редактировании авиарейсов, после завершения очередного действия на первый стек помещается индекс измененного элемента и код совершенной операции (2 – удаление, 1 – добавление, 0 – редактирование). На второй стек помещается объект до изменения. Данные процедуры позволяют совершать отмену последнего действия неограниченное количество раз (в программе ограничено 50). Если при попытке отмены последнего действия стеки оказались пустыми, пользователь увидит соответствующее сообщение об ошибке.



Рисунок 3.5 – Блок-схема процедуры отмены последнего действия

## 3.4 Структура входных и выходных данных

Входными данными для программного обеспечения персонального компьютера являются 3 бинарных файла. Один из файлов хранит в себе данные о пользователях, зарегистрированных в информационный системе и имеет структуру, представленную на рисунке 3.6.

Файл user.bin

|  |  |
| --- | --- |
| Login | Password |
| User1 | Password1 |

Рисунок 3.6- Структура файла *user.bin*

Следующий файл дает возможность узнать основную информацию о рейсах. На рисунке 3.7 приведена его структура.

Файл flight.bin

|  |  |  |  |
| --- | --- | --- | --- |
| flight\_number  (номер рейса) | departure\_place  (место вылета) | arrival\_place  (место прибытия) | Number\_of\_seats (время полета) |
| SU9812 | Минск | Киев | 200 |
| flight\_days  (день перелета) | departure\_time  (время вылета) | arrival\_time  (время прибытия) | price  (цена) |
| 30.05.2019 | 19:00 | 20:30 | 150 |

Рисунок 3.7 – Структура файла flight.bin

В третьем файле хранится информация об уже забронированных билетах. Информация между этим и вторым файлом, в котором хранится информация обо всех авиарейсах, связывается с помощью номера рейса. Ниже приведена его структура (рисунок 3.8).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| user  (Данные о пользователе) | flight\_number  (номер рейса) | seats  (количество  мест) | f\_class  (класс перелета) | result\_cost  (итоговая стоимость) |
| Gerus123 | SU9812 | 1 | 3 | 1400 |

Файл booked\_flight.bin

Рисунок 3.8 – Структура файла booked\_flight.bin

Выходными данными программного обеспечения персонального компьютера являются текстовые файлы, которые содержат измененную информацию о пользователях и авиарейсах.

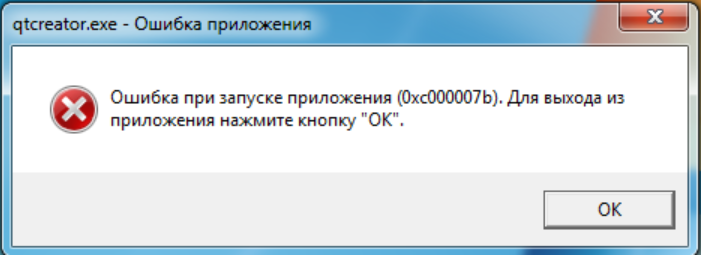
На данном этапе проектирование и разработка основных функций игрового приложения закончены. Все основные функции реализованы.

# 4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Большинство ошибок, возникших в результате написания программного средства, были связаны с недочетами проектирования ПО и исправлены на стадии разработки. Однако в процессе тестирования первоначальной версии продукта был выявлен ряд случаев некорректной работы программы.

Одной из основных проблем стала проблема с кодировкой файлов при работе с русским языком. Так как на русские буквы требуется по два байта, чтение в стандартную строку приводило к искажению данных и делало их нечитабельными. Проблема была решена заменой классического типа данных «string» на «wstring», которая подходит для чтения как английского, так и русского языка.

Еще одной сложностью стала загрузка изображений из ресурсов программы. Она была быстро устранена путем замены формата файла. Сообщение о данной ошибке продемонстрированно на рисунке 4.1.

  
Рисунок 4.1 – Ошибка, появляющаяся при работе с ресурсами

Остальные случаи возможного некорректного поведения программы были обнаружены на стадии проектирования и исправлены на стадии разработки.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данном разделе приводится вся необходимая информация, которая поможет пользователю в работе с приложением «MziliAvia».

## 5.1 Системные требования

Минимальные рекомендумые системные требования приложения «MziliAvia» для персонального компьютера:

1. операционная система – 64-разрядная версия Windows 8 и выше;
2. процессор – Intel Core i5;
3. операционная память – 512 мегабайт;
4. место на диске – 50 мегабайт.

## 5.2 Установка программы

Для установки приложения следует запустить файл «Setup\_MziliAvia.exe» (рисунок 5.1) и выполнить все инструкции программы.

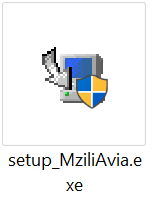


Рисунок 5.1 – Иконка установщика

## 5.3 Инструкция по использованию

**5.3.1** Запуск программы

Для запуска программы необходимо открыть исполняемый файл «MziliAvia.exe».

**5.3.2** Начало работы

При запуске программы можно увидеть следующий интерфейс, продемонстрированный на рисунке 5.1. Перемещение указателя для выбора режима выполняется курсором мыши. Для выбора пункта меню необходимо нажать левую кнопку мыши. Указателем для выбора режима игры служит анимация изменения цвета кнопок с указаниями. Приведённые ниже пункты меню выполняют следующие функции:

1. Войти – открывает окно с выбором игрока;
2. Регистрация – регистрирует пользователя и запускает стандартное пользовательское окно.

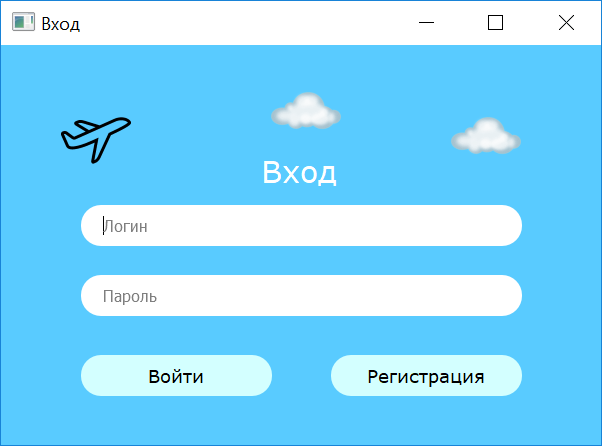


Рисунок 5.2 – Главное меню приложения

Запуск приложения осуществляется нажатием левой кнопки мыши по кнопке «Войти». В результате, в зависимости от режима доступа пользователя, он попадет на форму пользователя (рисунок 5.3) или на форму администратора (рисунок 5.4).

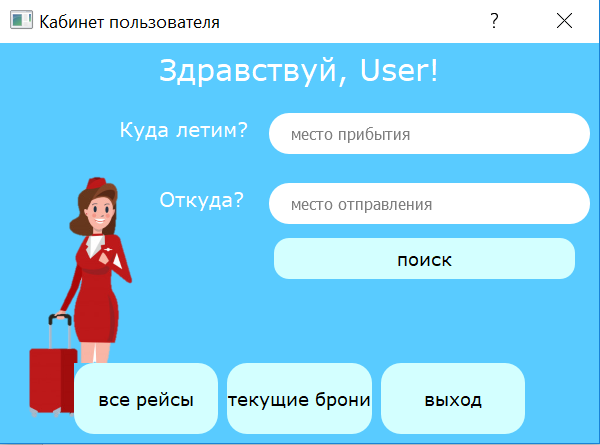


Рисунок 5.3 – Стандартное окно пользователя

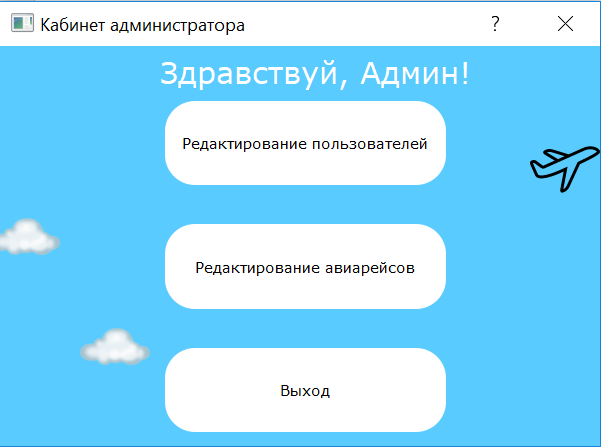


Рисунок 5.4 – Окно администратора

**5.3.3** Функциональность пользователя

Пользователь может забронировать рейс, просмотреть все рейсы, искать по рейсу и просматривать текущие брони. Приведённые ниже пункты меню выполняют следующие функции:

1. Поиск – осуществляет поиск рейса по выбранным направлениям;
2. Текущие брони – открывает окно для просмотра текущих броней;
3. Все рейсы – открывает окно для просмотра всех рейсов;
4. Выход – выход пользователя на форму авторизации.

Окна просмотра рейсов и брони практически идентичны и представлены ниже на рисунке 5.5.

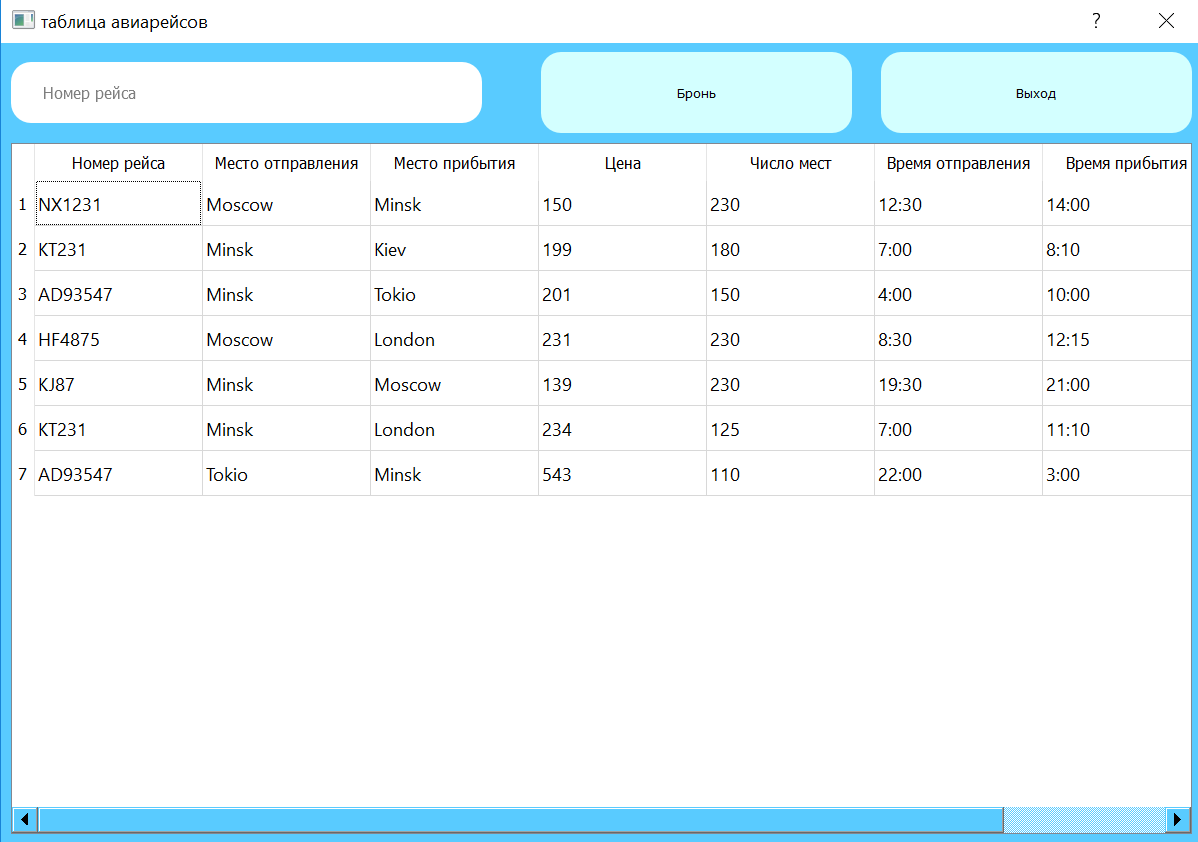


Рисунок 5.5 – Окно поиска рейсов и просмотра всех рейсов

На рисунке 5.6 продемонстрированно окно просмотра брони. При нажатии кнопки «Удалить», пользователь удаляет существующую бронь. Кнопка «Выход» вернет в стандартное окно пользователя.

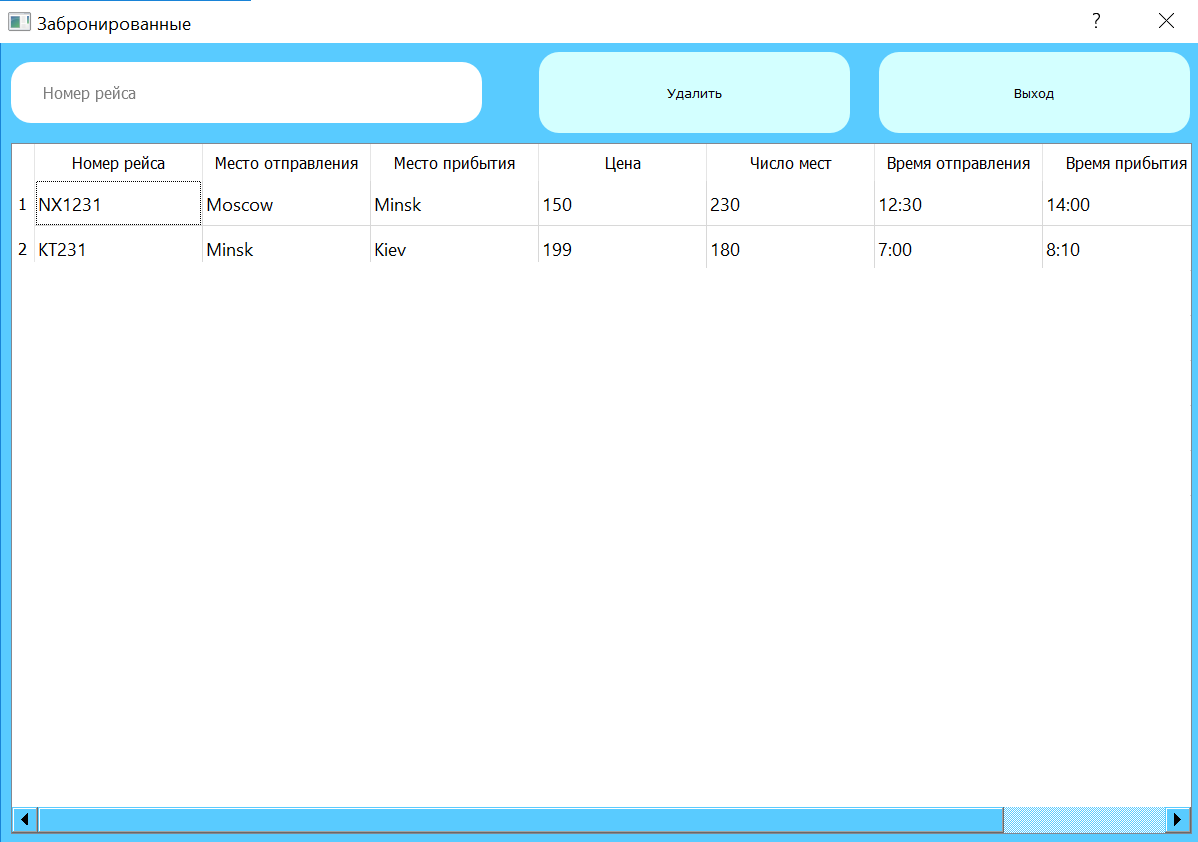


Рисунок 5.6 – Окно просмотра брони

**5.3.4** Функциональность Администратора

Администратор может просматривать и редактировать списки пользователей и рейсов, удалять и добавлять перелеты. Приведённые ниже пункты меню выполняют следующие функции:

При нажатии на кнопку редактирование пользователей появляется новое окно, на котором находится таблица с пользователями (рисунок 5.7).

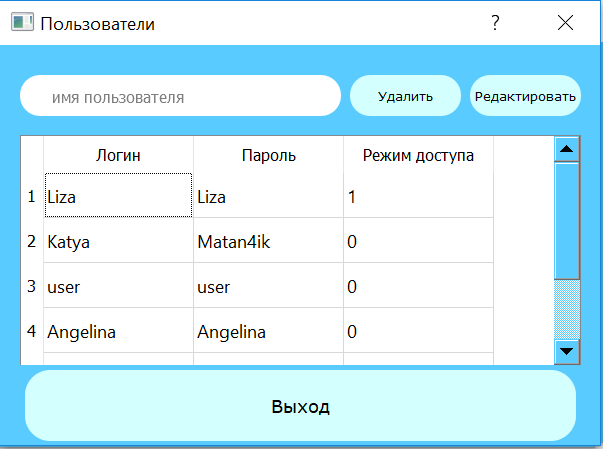


Рисунок 5.7 – Окно просмотра пользователей

Для редактирования отдельного пользователя необходимо ввести его имя в поле «Имя пользователя» и нажать на кнопку «Редактировать». Далее появится окно для редактирования (рисунок 5.8).

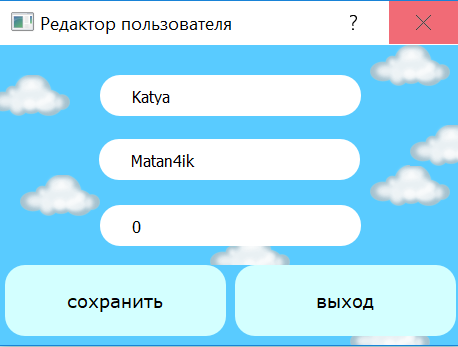


Рисунок 5.8 – Окно редактирования пользователя

Для того чтобы сохранить изменения, необходимо нажать кнопку «Сохранить». Для выхода необходимо нажать кнопку «Выход».

Для редактирования авиарейсов, необходимо нажать на кнопку «Редактирование авиарейсов», после чего откроется соответствующее окно, изображенное на рисунке 5.9.

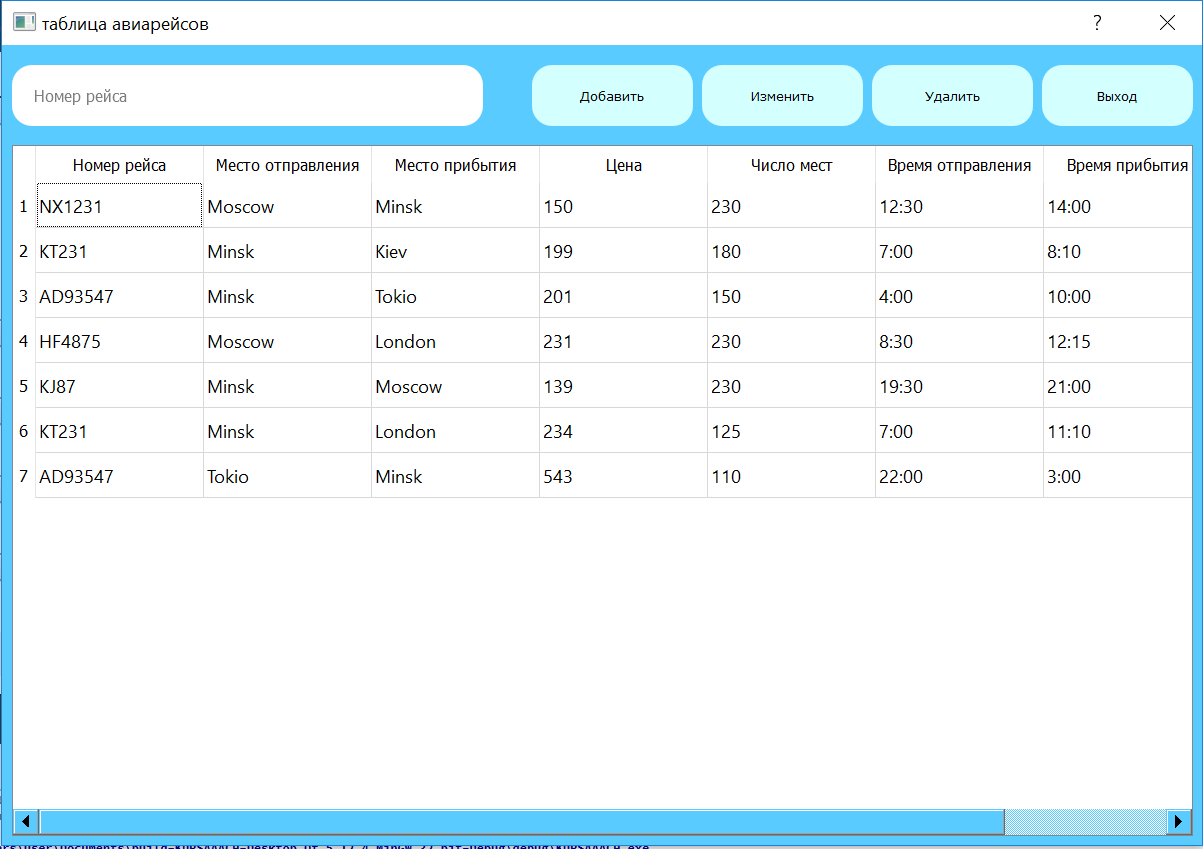


Рисунок 5.9 – Окно просмотра авиарейсов

Если необходимо изменить рейс, нужно ввести его номер в поле «Номер рейса», далее нажать на кнопку «Изменить». После этого откроется новое заполненное окно для редактирования. Аналогичное, но пустое окно, откроется при создании нового рейса (рисунок 5.10).

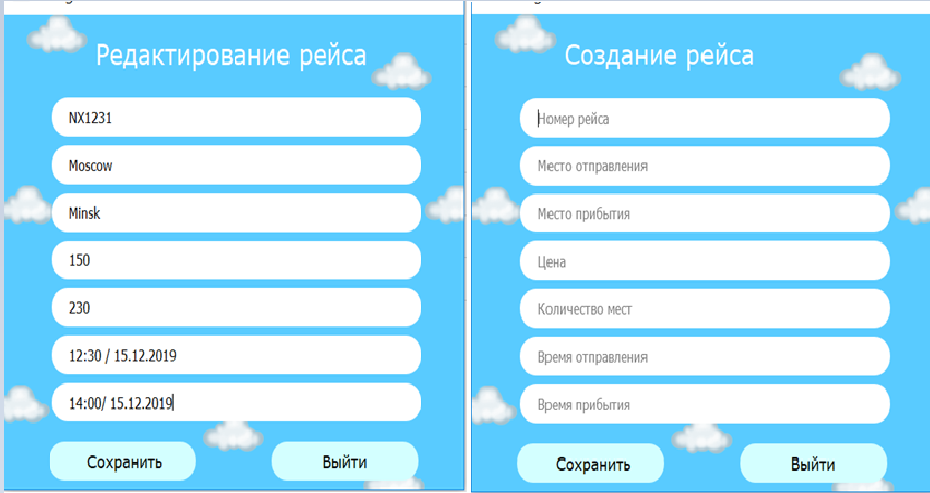


Рисунок 5.10 – редактирования и добавления рейса

При нажатии кнопки «Выход» пользователь попадет в стандартное окно администратора.

**ЗАКЛЮЧЕНИЕ**

В рамках данного курсового проекта было разработана система бронирования авиарейсов «MziliAvia», в котором просто и удобно для пользователя реализована пользовательская и административная части. В данном приложении были реализованы следующие функции:

1. проверки входа и регистрации пользователя;
2. добавления, удаления, редактирования авиарейсов и пользователей;
3. просмотра текущих авиарейсов и их бронирования;
4. отмены последнего действия.

Для успешного выполнения всех поставленных целей потребовалось изучить объектно-ориентированные возможности языка С++. Подробно познакомиться с многими компонентами Qt Creator и их возможностями.

Существует много возможностей для дальнейшего улучшения приложения. Можно добавить новые алгоритмы для поиска авиарейсов. Например, алгоритм Дейкстры. Также возможно изменение интерфейса (анимирование некоторых его частей). Ещё одним вариантом развития является адаптация проекта для запуска на устройствах с низкой разрешающей способностью экрана. Приложение также можно адаптировать под различные платформы.

«MziliAvia» позволит быстро забронировать билет на необходимый рейс для пользователя, комфортную и понятную управляющую часть для администратора.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Серебряная, Л.В. Марина, И.М. Структуры и алгоритмы обработки данных: учебно-метод. пособие для студ. спец. «Программное обеспечение информационных технологий» всех форм обуч. / Л. В. Серебряная, И. М. Марина. – Минск: БГУИР, 2012. – 49 с. ил.;
2. Шилдт, Г. С++. Полное руководство: учеб. Пособие. – Вильямс, 2017.
3. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003.
4. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003.
5. C++ справочник (электронный ресурс). – Электронные данные. – Режим доступа: http://cplusplus.com

# ПРИЛОЖЕНИЕ А. Исходный код программы

**\_algorithm.h**

#pragma once

#include "\_Iterator.h"

using namespace std;

template <class T>

class **\_Algorithm**

{

public:

\_Iterator<T> **search**(\_Iterator<T> begin, T& inf);

DoublyLinkedList<T> **searchAll**(\_Iterator<T> begin, T& inf);

};

**\_iterator.h**

#pragma once

#include <iostream>

#include "doublylinkedlist.h"

//#include "DoublyLinkedList.cpp"

using namespace std;

//template <class T>

//class DoublyLinkedList;

template <class T>

class **\_Iterator**

{

protected:

friend class DoublyLinkedList<T>;

Node<T>\* current;

public:

\_Iterator<T>& operator ++();

\_Iterator<T>& operator --();

\_Iterator<T>& operator ++(int);

\_Iterator<T>& operator --(int);

//bool operator >(const \_Iterator<T>);

//bool operator <(const \_Iterator<T>);

bool operator ==(const \_Iterator& it) const;

bool operator !=(const \_Iterator& it) const;

**\_Iterator**();

**\_Iterator**(Node<T> \*c);

~**\_Iterator**();

T& operator\*();

};

**additionalinformation.h**

#ifndef ADDITIONALINFORMATION\_H

#define ADDITIONALINFORMATION\_H

#include "flight.h"

#include <cstdlib>

class **AdditionalInformation** : public Flight{

string number = "";

string price = "";

string numOfSeats = "";

string timeOfDeparture = "";

string timeOfArrival = "";

public:

**AdditionalInformation**();

**AdditionalInformation**(string \_num, string \_dep, string \_arr, string \_price, string \_seats, string \_depTime, string \_arrTime)

{

Flight(\_dep, \_arr);

this->number = \_num;

this->price = \_price;

this->numOfSeats = \_seats;

this->timeOfDeparture = \_depTime;

this->timeOfArrival = \_arrTime;

}

void **SetNum**(string \_num)

{

this->number=\_num;

}

void **SetPrice**(string \_price)

{

this->price = \_price;

}

void **SetNumOfSeats**(string \_num)

{

this->numOfSeats = \_num;

}

void **SetTimeOfDep**(string \_depTime)

{

this->timeOfDeparture=\_depTime;

}

void **SetTimeOfArr**(string \_arrTime)

{

this->timeOfArrival=\_arrTime;

}

string **GetNum**()

{

return this->number;

}

string **GetPrice**()

{

return this->price;

}

string **GetNumOfSeats**()

{

return this->numOfSeats;

}

string **GetTimeOfDep**()

{

return this->timeOfDeparture;

}

string **GetTimeOfArr**()

{

return this->timeOfArrival;

}

inline friend ofstream& operator<<(ofstream& out, AdditionalInformation& inf)

{

out << static\_cast<Flight&>(inf);

out << inf.number << " ";

out << inf.price << " ";

out << inf.numOfSeats << " ";

out << inf.timeOfDeparture << " ";

out << inf.timeOfArrival << " ";

return out;

}

};

#endif // ADDITIONALINFORMATION\_H

**admin.h**

#ifndef ADMIN\_H

#define ADMIN\_H

#include<iostream>

#include<fstream>

using namespace std;

class **Admin** : public WorkWithFile

{

int rang;

public:

**Admin**();

bool **CheckMegaAdmin**(int,int);

};

#endif // ADMIN\_H

**adminframe.h**

#ifndef ADMINFRAME\_H

#define ADMINFRAME\_H

#include <QDialog>

#include <QTimer>

namespace **Ui** {

class **adminFrame**;

}

class **adminFrame** : public QDialog

{

Q\_OBJECT

public:

explicit **adminFrame**(QWidget \*parent = nullptr);

~***adminFrame***();

private slots:

void **TimePlane\_1**();

void **on\_pushButton\_clicked**();

void **on\_pushButton\_3\_clicked**();

void **on\_EditFlights\_clicked**();

private:

Ui::adminFrame \*ui;

int Index\_1 = 0, Index\_2 = 30, Index\_3 = 11, Index\_4 = 41, Index\_5 = 20;

QTimer\* Timer\_1;

};

#endif // ADMINFRAME\_H

**dataunit.h**

#ifndef DATAUNIT\_H

#define DATAUNIT\_H

#include "doublylinkedlist.h"

#include "additionalinformation.h"

#include "user.h"

#include "reserved.h"

#include <QMessageBox>

class **searchbtn**;

class **DataUnit**

{

public: DoublyLinkedList<user> UserList;

DoublyLinkedList<AdditionalInformation> FlightList;

// DoublyLinkedList<Reserved> ReservedList;

public:

**DataUnit**();

friend class searchbtn;

//DoublyLinkedList<user> GetUserList();

DoublyLinkedList<user>\* **GetUserList**(){

return &this->UserList;

}

DoublyLinkedList<AdditionalInformation>\* **GetFlighList**(){

return &this->FlightList;

}

void **InitializateUserList**();

void **InitializateFlightList**();

int **GetUserIndex**(string Name);

};

#endif // DATAUNIT\_H

**doublylinkedlist.h**

#pragma once

#include <iostream>

#include "Node.h"

//#include "\_Iterator.h"

//#include "\_Iterator.cpp"

//#include "\_Algorithm.h"

//#include "\_Algorithm.cpp"

using namespace std;

template <class T>

class **\_Iterator**;

template <class T>

class **DoublyLinkedList**

{

int Size;

Node<T> \*head;

Node<T> \*tail;

void **create**();

public:

**DoublyLinkedList**();

**DoublyLinkedList**(const DoublyLinkedList &);

**DoublyLinkedList**(DoublyLinkedList &&);

~**DoublyLinkedList**();

int **size**() const;

bool **empty**()const;

void **delete\_all**();

void **delete\_node**(Node <T>\*);

void **insert\_node**(const T& val, int index);

\_Iterator<T> **search**(const T& date);

void **remove**(const T&);

void **pop\_back**();

void **pop\_front**();

void **push\_front**(const T & val);

void **push\_back**(const T & val);

const T& **front**();

const T& **back**();

\_Iterator<T> **begin**();

\_Iterator<T> **end**();

inline friend std::ostream& operator<<(std::ostream& os, const DoublyLinkedList<T>& l)

{

Node<T>\* element = l.head->next;

for (; element != l.tail; element = element->next) os << element->inf << ' ';

return os;

}

inline friend bool operator ==(const DoublyLinkedList<T>& list1, const DoublyLinkedList<T>& list2)

{

if (list1.size() == list2.size())

{

Node<T>\* element1 = list1.head->next;

Node<T>\* element2 = list2.head->next;

for (; element1 != list1.tail; element1 = element1->next, element2 = element2->next)

{

if (element1->inf != element2->inf)

return false;

}

return true;

}

else

return false;

}

inline friend bool operator !=(const DoublyLinkedList<T>& list1, const DoublyLinkedList<T>& list2)

{

return !(list1 == list2);

}

T& operator [](int);

friend class \_Iterator<T>;

};

**editflight.h**

#ifndef EDITFLIGHT\_H

#define EDITFLIGHT\_H

#include <QDialog>

namespace **Ui** {

class **EditFlight**;

}

class **EditFlight** : public QDialog

{

Q\_OBJECT

public:

explicit **EditFlight**(QWidget \*parent = nullptr, int Mode = 1);

~***EditFlight***();

private slots:

void **on\_LoginBTN\_clicked**();

private:

Ui::EditFlight \*ui;

int Index = 0;

};

#endif // EDITFLIGHT\_H

**editflightframe.h**

#ifndef EDITFLIGHTSFORM\_H

#define EDITFLIGHTSFORM\_H

#include <QDialog>

#include "dataunit.h"

#include "usereditingframe.h"

#include "string"

namespace Ui {

class EditFlightsForm;

}

class EditFlightsForm : public QDialog

{

Q\_OBJECT

public:

explicit EditFlightsForm(QWidget \*parent = nullptr);

~EditFlightsForm();

string GetMode();

void RefrathTable();

private slots:

void on\_ExitBTN\_clicked();

void on\_EditUser\_clicked(bool checked);

void on\_EditUser\_clicked();

void on\_ExitBTN\_2\_clicked();

void on\_DeleteUser\_clicked();

private:

Ui::EditFlightsForm \*ui;

};

#endif // EDITFLIGHTSFORM\_H

**flight.h**

#ifndef FLIGHT\_H

#define FLIGHT\_H

#include <iostream>

#include <string>

#include <list>

#include <fstream>

#include <iterator>

using namespace std;

class **Flight**

{

// string number = "";

string departurePoint = "";

string placeOfArrival = "";

// int price = 0;

public:

**Flight**();

**Flight**(string \_dep, string \_arr)

{

// this->number = \_num;

this->departurePoint = \_dep;

this->placeOfArrival = \_arr;

// this->price = \_price;

}

// void SetNum(string \_num)

// {

// this->number=\_num;

// }

void **SetDeparture**(string \_dep)

{

this->departurePoint=\_dep;

}

void **SetArrieve**(string \_arr)

{

this->placeOfArrival=\_arr;

}

string **GetDeparture**()

{

return this->departurePoint;

}

string **GetArrieve**()

{

return this->placeOfArrival;

}

// void SetPrice(string \_price)

// {

// this->price=stoi(\_price);

// }

inline friend ofstream& operator<<(ofstream& out, Flight& fl)

{

// out << fl.number << " ";

out << fl.departurePoint << " ";

out << fl.placeOfArrival << " ";

// out << fl.price << " ";

return out;

}

};

#endif // FLIGHT\_H

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTimer>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **TimeFunc**();

void **on\_RegBTN\_clicked**();

void **on\_LoginBTN\_clicked**();

private:

Ui::MainWindow \*ui;

int Index = 0, Index\_1 = 39, Index\_2 = 21;

QTimer\* Timer;

};

#endif // MAINWINDOW\_H

**Node.h**

#ifndef NODE\_H

#define NODE\_H

#include <iostream>

using namespace std;

template <class T>

struct **Node**

{

T inf;

Node\* next;

Node\* prev;

**Node**(const T& d = T{}, Node\* p = nullptr, Node\* n = nullptr) : inf{ d }, prev{ p }, next{ n } {}

**Node**(T&& d, Node\* p = nullptr, Node\* n = nullptr) : inf{ move(d) }, prev{ p }, next{ n } {}

};

#endif // NODE\_H

**reserved.h**

#ifndef RESERVED\_H

#define RESERVED\_H

#include "user.h"

#include "additionalinformation.h"

class **Reserved**: public user, public AdditionalInformation

{

int TypeOfSeats = 0;

int NumOfReservedSeats = 0;

public:

**Reserved**();

};

#endif // RESERVED\_H

**search.h**

#ifndef SEARCHBTN\_H

#define SEARCHBTN\_H

#include "user.h"

#include "dataunit.h"

#include <QDialog>

namespace Ui {

class SearchBTN;

}

class SearchBTN : public QDialog

{

Q\_OBJECT

public:

explicit SearchBTN(QWidget \*parent = nullptr, int UserIndex = 0);

~SearchBTN();

DataUnit all;

int Index;

private slots:

void on\_pushButton\_5\_clicked();

private:

Ui::SearchBTN \*ui;

};

#endif // SEARCHBTN\_H

**standartuserframe.h**

#ifndef STANDARTUSERFRAME\_H

#define STANDARTUSERFRAME\_H

#include <QDialog>

using namespace std;

namespace Ui {

class StandartUserFrame;

}

class StandartUserFrame : public QDialog

{

Q\_OBJECT

string SUserName;

public:

explicit StandartUserFrame(QWidget \*parent = nullptr, string UserName = "");

~StandartUserFrame();

private slots:

void on\_pushButton\_5\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_clicked();

private:

Ui::StandartUserFrame \*ui;

};

#endif // STANDARTUSERFRAME\_H

**user.h**

#ifndef USER\_H

#define USER\_H

#include <iostream>

#include <fstream>

#include <string>

#include "workwithfile.h"

using namespace std;

class **user**{ //public WorkWithFile{

std::string name = "";

std::string password = "";

bool access\_mode = false;

public:

**user**();

user operator = (user & us)

{

this->name = us.name;

this->password = us.password;

this->access\_mode = us.access\_mode;

return \*this;

}

**user** (std::string login, std::string password);

bool **SaveUser**();

bool **GetMode**();

bool **CheckUser**();

bool **CheckUserForReg**();

string **GetName**(){

return name;

}

string **GetPassword**(){

return password;

}

void **SetName**(string \_name){

this->name=\_name;

}

void **SetPassword**(string \_password){

this->password=\_password;

}

void **SetMode**(bool \_mode){

this->access\_mode=\_mode;

}

bool operator !=(user & obj)

{

bool m;

return m = (name!=obj.name);

}

friend bool operator == (const user user\_1, const user user\_2);

friend bool operator != (const user user\_1, const user user\_2);

inline friend ofstream& operator<<(ofstream& out, user& us)

{

out << us.name << " ";

out << us.password << " ";

if(us.access\_mode)

out << "1 ";

else

out << "0 ";

return out;

}

};

#endif // USER\_H

**usereditingframe.h**

#ifndef USEREDITINGFRAME\_H

#define USEREDITINGFRAME\_H

#include <QDialog>

#include "doublylinkedlist.h"

#include "doublylinkedlist.cpp"

#include "user.h"

#include "additionalinformation.h"

namespace **Ui** {

class **UserEditingFrame**;

}

class **UserEditingFrame** : public QDialog

{

Q\_OBJECT

DoublyLinkedList<user> UserList;

DoublyLinkedList<AdditionalInformation> FlightList;

public:

explicit **UserEditingFrame**(QWidget \*parent = nullptr);

~***UserEditingFrame***();

void **InitializateUserList**();

void **InitializateFlightList**();

void **RefreshTable**();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_3\_clicked**();

void **on\_EditUser\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_DeleteUser\_clicked**();

private:

Ui::UserEditingFrame \*ui;

};

#endif // USEREDITINGFRAME\_H

**workwithfile.h**

#ifndef WORKWITHFILE\_H

#define WORKWITHFILE\_H

#include <string>

#include <iostream>

#include <fstream>

using namespace std;

class **WorkWithFile**

{

public:

**WorkWithFile**();

bool **SaveFile**(std::string filename, string name,string password,bool access\_mode);

};

#endif // WORKWITHFILE\_H

**\_algorithm.cpp**

#include "\_algorithm.h"

template<class T>

\_Iterator<T> \_Algorithm<T>::**search**(\_Iterator<T> begin, T & inf)

{

bool flag = 0;

\_Iterator<T> it1;

it1 = begin;

while (it1 != nullptr && !flag)

{

if (inf != \*it1)

{

it1++;

}

else

{

flag = 1;

}

}

if (!flag)

{

return nullptr;

}

else return it1;

}

template<class T>

DoublyLinkedList<T> \_Algorithm<T>::**searchAll**(\_Iterator<T> begin, T & inf)

{

DoublyLinkedList<T> finded;

\_Iterator<T> it1, it2;

it1 = begin;

do

{

if (inf != \*it1)

{

it1++;

}

else

{

finded.push\_back(\*it1);

it1++;

}

} while (it1 != nullptr);

return finded;

}

**\_iterator.cpp**

#include <iostream>

#include "\_Iterator.h"

template <class T>

\_Iterator<T>::**\_Iterator**()

{

}

template<class T>

\_Iterator<T>& \_Iterator<T>::operator++()

{

this->current = this->current->next;

return \*this;

}

template<class T>

\_Iterator<T>& \_Iterator<T>::operator--()

{

this->current = this->current->prev;

return \*this;

}

template<class T>

\_Iterator<T>& \_Iterator<T>::operator++(int)

{

auto cpy = \*this;

current = current->next;

return cpy;

}

template<class T>

\_Iterator<T>& \_Iterator<T>::operator--(int)

{

auto cpy = \*this;

current = current->prev;

return cpy;

}

//template<class T>

//bool \_Iterator<T>::operator>(const \_Iterator<T> element)

//{

// Node<T>\* pointer = element.current;

// for (; pointer != this->current && pointer->next; pointer = pointer->next);

//

// if (pointer->next)

// {

// return false;

// }

// else return true;

//}

template <typename T>

bool \_Iterator<T>:: operator ==(const \_Iterator& it) const //перегрузка оператора сравнения

{

return(current == it.current);

}

template <typename T>

bool \_Iterator<T>:: operator !=(const \_Iterator& it) const //перегрузка оператора сравнения

{

return(current != it.current);

}

template<class T>

\_Iterator<T>::**\_Iterator**(Node<T>\* c)

{

current = c;

}

template <class T>

\_Iterator<T>::~**\_Iterator**()

{

}

template<class T>

T & \_Iterator<T>::operator\*()

{

return current->inf;

}

**adminframe.cpp**

#include "adminframe.h"

#include "ui\_adminframe.h"

#include "usereditingframe.h"

#include "editflightsform.h"

#include <QTimer>

#include "dataunit.h"

adminFrame::**adminFrame**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::adminFrame)

{

ui->setupUi(this);

Timer\_1 = new QTimer(this);

Timer\_1->start(140);

connect(Timer\_1, SIGNAL(timeout()), this, SLOT(TimePlane\_1()));

}

void adminFrame::**TimePlane\_1**()

{

const int sizeX = 81, sizeY = 61, cloud\_size\_x = 71, cloud\_size\_y = 61;

int size = 64;

int PosArr[65][2] = { {-50, 90}, {-40, 88}, {-30, 85}, {-20, 83}, {-10, 80}, {0, 78}, {10, 75}, {20, 73}, {30, 70}, {40, 68},

{50, 65}, {60, 62}, {70, 60}, {80, 58}, {90, 55}, {100, 52}, {110, 50}, {120, 48}, {130, 45}, {140, 43},

{150, 40}, {160, 38}, {170, 35}, {180, 35}, {190, 35}, {200, 35}, {210, 35}, {220, 33}, {230, 35}, {240, 35},

{250, 35}, {260, 35}, {270, 35}, {280, 37}, {290, 40}, {300, 43}, {310, 45}, {320, 48}, {330, 50}, {340, 52},

{350, 55}, {360, 58}, {370, 60}, {380, 62}, {390, 65}, {400, 68}, {410, 70}, {420, 73}, {430, 75}, {440, 78},

{450, 80}, {460, 83}, {470, 85}, {480, 88}, {490, 90}, {500, 90}, {510, 90}, {520, 90}, {530, 90}, {540, 90},

{550, 90}, {560, 90}, {570, 90}, {580, 90}, {590, 90}};

ui->plane\_1->setGeometry(PosArr[Index\_1][0], PosArr[Index\_1][1], sizeX, sizeY);

ui->cloud\_1->setGeometry(PosArr[Index\_2][0], PosArr[Index\_2][1], cloud\_size\_x, cloud\_size\_y);

ui->plane\_2->setGeometry(PosArr[Index\_4][0], PosArr[Index\_3][1] + 100, sizeX, sizeY);

ui->cloud\_2->setGeometry(PosArr[Index\_3][0], PosArr[Index\_4][1] + 120, cloud\_size\_x, cloud\_size\_y);

ui->cloud\_3->setGeometry(PosArr[Index\_5][0], PosArr[Index\_4][1] + 230, cloud\_size\_x, cloud\_size\_y);

Index\_1++;

Index\_2++;

Index\_3++;

Index\_4++;

Index\_5++;

if (Index\_1 > size){

Index\_1 = 0;

}

if (Index\_2 > size){

Index\_2 = 0;

}

if (Index\_3 > size){

Index\_3 = 0;

}

if (Index\_4 > size){

Index\_4 = 0;

}

if (Index\_5 > size){

Index\_5 = 0;

}

}

adminFrame::~***adminFrame***()

{

delete ui;

}

void adminFrame::**on\_pushButton\_clicked**()

{

UserEditingFrame window;

window.setModal(true);

window.*exec*();

}

void adminFrame::**on\_EditFlights\_clicked**()

{

EditFlightsForm window;

window.setModal(true);

window.*exec*();

}

**dataunit.cpp**

#include "dataunit.h"

#include <QMessageBox>

DataUnit::**DataUnit**()

{

}

void DataUnit::**InitializateUserList**(){

user tempUser;

string temp;

ifstream Users;

Users.open("Users.txt", ios::in);

//Flights.open("Flights.txt", ios::in);

if (!Users.is\_open()){

//QMessageBox::critical(this, "Ошибка", "Файл не может быть открыт");

} else {

while (!Users.eof()){

Users>>temp;

tempUser.SetName(temp);

Users>>temp;

tempUser.SetPassword(temp);

Users>>temp;

if(temp=="1")

tempUser.SetMode(true);

else

tempUser.SetMode(false);

this->UserList.push\_back(tempUser);

}

}

Users.close();

}

int DataUnit::**GetUserIndex**(string Name){

for (int i = 0; i < this->UserList.size() - 1; i++){

if (this->UserList[i].GetName() == Name){

return i;

}

}

return -1;

}

void DataUnit::**InitializateFlightList**()

{

AdditionalInformation tempInf;

string temp;

ifstream Flights, AdditionalInf;

Flights.open("Flights.txt", ios::in);

AdditionalInf.open("AddInf.txt", ios::in);

if (!Flights.is\_open()||!AdditionalInf.is\_open()){

// QMessageBox::critical(this, "Ошибка", "Файл не может быть открыт");

} else {

while (!Flights.eof()){

Flights>>temp;

tempInf.SetNum(temp);

Flights>>temp;

tempInf.SetDeparture(temp);

Flights>>temp;

tempInf.SetArrieve(temp);

Flights>>temp;

tempInf.SetPrice(temp);

AdditionalInf>>temp;

tempInf.SetNumOfSeats(temp);

AdditionalInf>>temp;

tempInf.SetTimeOfDep(temp);

AdditionalInf>>temp;

tempInf.SetTimeOfArr(temp);

this->FlightList.push\_back(tempInf);

}

}

Flights.close();

AdditionalInf.close();

}

**doublylinkedlist.cpp**

#include <iostream>

#include "DoublyLinkedList.h"

//using namespace std;

template <class T>

DoublyLinkedList<T>::**DoublyLinkedList**()

{

create();

}

template <class T>

DoublyLinkedList<T>::**DoublyLinkedList**(const DoublyLinkedList& list) //copy

{

create();

for (Node<T>\* itr = list.head->next; itr != list.tail; itr = itr->next)

push\_back(itr->inf);

}

template<class T>

DoublyLinkedList<T>::**DoublyLinkedList**(DoublyLinkedList<T>&& list) : Size(list.Size), head{ list.head }, tail{ list.tail } //moving

{

list.Size = 0; list.head = nullptr; list.tail = nullptr;

}

template <class T>

DoublyLinkedList<T>::~**DoublyLinkedList**()

{

delete\_all();

delete head;

delete tail;

}

template <class T>

void DoublyLinkedList<T>::**create**()

{

Size = 0;

head = new Node<T>();

tail = new Node<T>();

head->next = tail;

tail->prev = head;

}

template <class T>

int DoublyLinkedList<T>::**size**() const

{

return Size;

}

template <class T>

bool DoublyLinkedList<T>::**empty**() const

{

if (Size == 0)

return 1;

else

return 0;

}

template <class T>

void DoublyLinkedList<T>::**delete\_all**()

{

while (!empty()) {

pop\_back();

}

}

template <class T>

void DoublyLinkedList<T>::**pop\_back**()

{

delete\_node(tail->prev);

}

template <class T>

void DoublyLinkedList<T>::**pop\_front**()

{

delete\_node(head->next);

}

template <class T>

void DoublyLinkedList<T>::**delete\_node**(Node<T>\* d)

{

d->prev->next = d->next;

d->next->prev = d->prev;

--Size;

delete d;

}

template <class T>

void DoublyLinkedList<T>::**push\_back**(const T& n)

{

insert\_node(n, Size - 1);

}

template <class T>

void DoublyLinkedList<T>::**push\_front**(const T& n)

{

if (this->Size != 0)

insert\_node(n, 0);

else

insert\_node(n, Size - 1);

}

template <class T>

void DoublyLinkedList<T>::**insert\_node**(const T& val, int index)

{

if (index >= Size) return;

Node<T>\* element = head->next;

for (int i = 0; i <= index; ++i, element = element->next);

Node<T>\* insertingElement = new Node<T>(val);

element->prev->next = insertingElement;

insertingElement->prev = element->prev;

insertingElement->next = element;

element->prev = insertingElement;

++Size;

}

template <class T>

\_Iterator<T> DoublyLinkedList<T>::**search**(const T& date)

{

for (auto ths = begin(); ths != end(); ths++)

{

if (\*ths==date)

{

return ths;

}

}

return \_Iterator<T> (nullptr);

}

template <class T>

const T& DoublyLinkedList<T>::**front**()

{

return head->next->inf;

}

template <class T>

const T& DoublyLinkedList<T>::**back**()

{

return tail->prev->inf;

}

template<class T>

\_Iterator<T> DoublyLinkedList<T>::**begin**()

{

\_Iterator<T> i{ head->next };

return i;

}

template<class T>

\_Iterator<T> DoublyLinkedList<T>::**end**()

{

\_Iterator<T> i = {tail->prev};

return i;

}

template <class T>

void DoublyLinkedList<T>::**remove**(const T& data)

{

Node<T>\* search = head->next;

for (; search != tail; search = search->next)

{

if (search->inf == data)

{

Node<T>\* next = search->next;

delete\_node(search);

search = next;

}

}

}

template <class T>

T& DoublyLinkedList<T>:: operator [](int number)

{

Node<T>\* element = head->next;

for (int i = 0; i < number; element = element->next, i++);

return element->inf;

}

**editflight.cpp**

#include "editflight.h"

#include "ui\_editflight.h"

#include <stdlib.h>

#include "editflightsform.h"

#include "dataunit.h"

#include "additionalinformation.h"

EditFlight::EditFlight(QWidget \*parent, int Mode) :

QDialog(parent),

ui(new Ui::EditFlight)

{

DataUnit List;

List.InitializateFlightList();

ui->setupUi(this);

EditFlightsForm window;

Index = Mode;

if (Mode != -1)

{

QString qstr = QString::fromStdString(List.FlightList[Mode].GetNum());

ui->LoginEdit\_2->setText(qstr);

qstr = QString::fromStdString(List.FlightList[Mode].GetDeparture());

ui->LoginEdit\_6->setText(qstr);

qstr = QString::fromStdString(List.FlightList[Mode].GetArrieve());

ui->LoginEdit\_5->setText(qstr);

qstr = QString::fromStdString(List.FlightList[Mode].GetPrice());

ui->LoginEdit\_7->setText(qstr);

qstr = QString::fromStdString(List.FlightList[Mode].GetNumOfSeats());

ui->LoginEdit\_8->setText(qstr);

qstr = QString::fromStdString(List.FlightList[Mode].GetTimeOfDep());

ui->LoginEdit\_4->setText(qstr);

qstr = QString::fromStdString(List.FlightList[Mode].GetTimeOfArr());

ui->LoginEdit\_3->setText(qstr);

}

}

EditFlight::~EditFlight()

{

delete ui;

}

void EditFlight::on\_LoginBTN\_clicked()

{

DataUnit List;

List.InitializateFlightList();

if (Index != -1){

List.FlightList[Index].SetNum(ui->LoginEdit\_2->text().toStdString());

List.FlightList[Index].SetDeparture(ui->LoginEdit\_6->text().toStdString());

List.FlightList[Index].SetArrieve(ui->LoginEdit\_5->text().toStdString());

List.FlightList[Index].SetPrice(ui->LoginEdit\_7->text().toStdString());

List.FlightList[Index].SetNumOfSeats(ui->LoginEdit\_8->text().toStdString());

List.FlightList[Index].SetTimeOfDep(ui->LoginEdit\_4->text().toStdString());

List.FlightList[Index].SetTimeOfArr(ui->LoginEdit\_3->text().toStdString());

}else {

AdditionalInformation tempInf;

tempInf.SetNum(ui->LoginEdit\_2->text().toStdString());

tempInf.SetDeparture(ui->LoginEdit\_6->text().toStdString());

tempInf.SetArrieve(ui->LoginEdit\_5->text().toStdString());

tempInf.SetPrice(ui->LoginEdit\_7->text().toStdString());

tempInf.SetNumOfSeats(ui->LoginEdit\_8->text().toStdString());

tempInf.SetTimeOfDep(ui->LoginEdit\_4->text().toStdString());

tempInf.SetTimeOfArr(ui->LoginEdit\_3->text().toStdString());

List.FlightList.push\_back(tempInf);

}

ofstream Flights, AdditionalInf;

Flights.open("Flights.txt", std::ios::out);

AdditionalInf.open("AddInf.txt", std::ios::out);

for (int i = 0; i < List.FlightList.size(); i++){

Flights << List.FlightList[i].GetNum() << " ";

Flights << List.FlightList[i].GetDeparture() << " ";

Flights << List.FlightList[i].GetArrieve() << " ";

if (i != List.FlightList.size() - 1){

Flights << List.FlightList[i].GetPrice() << " ";

} else {

Flights << List.FlightList[i].GetPrice();

}

AdditionalInf << List.FlightList[i].GetNumOfSeats() << " ";

AdditionalInf << List.FlightList[i].GetTimeOfDep() << " ";

if (i != List.FlightList.size() - 1){

AdditionalInf << List.FlightList[i].GetTimeOfArr() << " ";

} else {

AdditionalInf << List.FlightList[i].GetTimeOfArr();

}

}

Flights.close();

AdditionalInf.close();

ui->LoginBTN\_3->click();

}

**deditflightframe.cpp**

#include "editflightsform.h"

#include "ui\_editflightsform.h"

#include "editflight.h"

#include "dataunit.h"

EditFlightsForm::**EditFlightsForm**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::EditFlightsForm)

{

DataUnit all;

ui->setupUi(this);

ui->FlightsTable->setColumnCount(7);

for (int i = 0; i < 7; i++)

ui->FlightsTable->setColumnWidth(i, 168);

QStringList table\_names;

table\_names << "Номер рейса" << "Место отправления" << "Место прибытия"<<"Цена"<<"Число мест"<<"Время отправления"<<"Время прибытия";

ui->FlightsTable->setHorizontalHeaderLabels(table\_names);

ui->mod->hide();

RefrathTable();

//\*\*\*\*\*\*\*\*

all.InitializateFlightList();

// ofstream Flights;

// Flights.open("Flightss.txt", std::ios::out);

// Flights << all.GetFlighList();

// Flights.close();

//\*\*\*\*\*\*\*\*\*\*\*\*

}

EditFlightsForm::~***EditFlightsForm***()

{

delete ui;

}

string EditFlightsForm::**GetMode**(){

return ui->mod->text().toStdString();

}

void EditFlightsForm::**on\_ExitBTN\_clicked**()

{

}

void EditFlightsForm::**RefrathTable**(){

DataUnit List;

List.InitializateFlightList();

ui->FlightsTable->setRowCount(List.FlightList.size());

QTableWidgetItem \*item = new QTableWidgetItem();

for (int j = 0; j < List.FlightList.size(); j++){

int i = 0;

QString qstr = QString::fromStdString(List.FlightList[j].GetNum());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetDeparture());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetArrieve());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetPrice());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetNumOfSeats());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetTimeOfDep());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetTimeOfArr());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

}

List.FlightList.delete\_all();

}

void EditFlightsForm::**on\_EditUser\_clicked**()

{

DataUnit List;

List.InitializateFlightList();

int Index = -1;

for (int i = 0; i < List.FlightList.size(); i++ ){

if (List.FlightList[i].GetNum() == ui->PasswordEdit->text().toStdString()){

Index =i; break;

}

}

if (Index != -1){

EditFlight window(nullptr, Index);

window.setModal(true);

window.*exec*();

RefrathTable();

}

}

void EditFlightsForm::**on\_ExitBTN\_2\_clicked**()

{

EditFlight window(nullptr, -1);

window.setModal(true);

window.*exec*();

RefrathTable();

}

void EditFlightsForm::**on\_DeleteUser\_clicked**()

{

DataUnit List;

List.InitializateFlightList();

int Index = -1;

for (int i = 0; i < List.FlightList.size(); i++ ){

if (List.FlightList[i].GetNum() == ui->PasswordEdit->text().toStdString()){

Index =i; break;

}

}

AdditionalInformation temp = List.FlightList[Index];

// List.FlightList.remove(temp);

ofstream Flights, AdditionalInf;

Flights.open("Flights.txt", std::ios::out);

AdditionalInf.open("AddInf.txt", std::ios::out);

for (int i = 0; i < List.FlightList.size(); i++){

Flights << List.FlightList[i].GetNum() << " ";

Flights << List.FlightList[i].GetDeparture() << " ";

Flights << List.FlightList[i].GetArrieve() << " ";

if (i != List.FlightList.size() - 1){

Flights << List.FlightList[i].GetPrice() << " ";

} else {

Flights << List.FlightList[i].GetPrice();

}

AdditionalInf << List.FlightList[i].GetNumOfSeats() << " ";

AdditionalInf << List.FlightList[i].GetTimeOfDep() << " ";

if (i != List.FlightList.size() - 1){

AdditionalInf << List.FlightList[i].GetTimeOfArr() << " ";

} else {

AdditionalInf << List.FlightList[i].GetTimeOfArr();

}

}

Flights.close();

AdditionalInf.close();

RefrathTable();

}

**Deditflightframe\_1.cpp**

#include "editflightsform\_1.h"

#include "ui\_editflightsform\_1.h"

#include "dataunit.h"

EditFlightsForm\_1::EditFlightsForm\_1(QWidget \*parent, int Mode, string Depart, string Arrive, string UserName) :

QDialog(parent),

ui(new Ui::EditFlightsForm\_1)

{

ui->setupUi(this);

SUserName = UserName;

ui->FlightsTable->setColumnCount(7);

for (int i = 0; i < 7; i++)

ui->FlightsTable->setColumnWidth(i, 168);

QStringList table\_names;

table\_names << "Номер рейса" << "Место отправления" << "Место прибытия"<<"Цена"<<"Число мест"<<"Время отправления"<<"Время прибытия";

ui->FlightsTable->setHorizontalHeaderLabels(table\_names);

if (Mode == 0){

List.InitializateFlightList();

} else {

DataUnit TempList;

TempList.InitializateFlightList();

for (int i = 0; i < TempList.FlightList.size(); i++){

if ((TempList.FlightList[i].GetDeparture() == Depart) && (TempList.FlightList[i].GetArrieve() == Arrive)){

List.FlightList.push\_back(TempList.FlightList[i]);

}

}

}

RefreshTable();

}

void EditFlightsForm\_1::RefreshTable(){

ui->FlightsTable->setRowCount(List.FlightList.size());

QTableWidgetItem \*item = new QTableWidgetItem();

for (int j = 0; j < List.FlightList.size(); j++){

int i = 0;

QString qstr = QString::fromStdString(List.FlightList[j].GetNum());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetDeparture());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetArrieve());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetPrice());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetNumOfSeats());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetTimeOfDep());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

qstr = QString::fromStdString(List.FlightList[j].GetTimeOfArr());

item->setText(qstr);

ui->FlightsTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

}

List.FlightList.delete\_all();

}

EditFlightsForm\_1::~EditFlightsForm\_1()

{

delete ui;

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QLabel>

#include <QPropertyAnimation>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "adminframe.h"

#include "standartuserframe.h"

#include "user.h"

#include <QMessageBox>

#include <QTimer>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

Timer = new QTimer(this);

Timer->start(140);

connect(Timer, SIGNAL(timeout()), this, SLOT(TimeFunc()));

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**TimeFunc**()

{

const int sizeX = 81, sizeY = 61, cloud\_size\_x = 71, cloud\_size\_y = 61;;

int size = 64;

int PosArr[65][2] = { {-50, 90}, {-40, 88}, {-30, 85}, {-20, 83}, {-10, 80}, {0, 78}, {10, 75}, {20, 73}, {30, 70}, {40, 68},

{50, 65}, {60, 62}, {70, 60}, {80, 58}, {90, 55}, {100, 52}, {110, 50}, {120, 48}, {130, 45}, {140, 43},

{150, 40}, {160, 38}, {170, 35}, {180, 35}, {190, 35}, {200, 35}, {210, 35}, {220, 33}, {230, 35}, {240, 35},

{250, 35}, {260, 35}, {270, 35}, {280, 37}, {290, 40}, {300, 43}, {310, 45}, {320, 48}, {330, 50}, {340, 52},

{350, 55}, {360, 58}, {370, 60}, {380, 62}, {390, 65}, {400, 68}, {410, 70}, {420, 73}, {430, 75}, {440, 78},

{450, 80}, {460, 83}, {470, 85}, {480, 88}, {490, 90}, {500, 90}, {510, 90}, {520, 90}, {530, 90}, {540, 90},

{550, 90}, {560, 90}, {570, 90}, {580, 90}, {590, 90}};

ui->planeP->setGeometry(PosArr[Index][0], PosArr[Index][1], sizeX, sizeY);

ui->cloud\_1->setGeometry(PosArr[Index\_2][0], PosArr[Index\_2][1], cloud\_size\_x, cloud\_size\_y);

ui->cloud\_2->setGeometry(PosArr[Index\_1][0], PosArr[Index\_1][1] - 20, cloud\_size\_x, cloud\_size\_y);

Index++;

Index\_1++;

Index\_2++;

if (Index > size){

Index = 0;

}

if (Index\_1 > size){

Index\_1 = 0;

}

if (Index\_2 > size){

Index\_2 = 0;

}

}

void MainWindow::**on\_RegBTN\_clicked**()

{

if ((ui->LoginEdit->text().length() == 0) || (ui->PasswordEdit->text().length() == 0)){

QMessageBox::critical(this, "Ошибка", "Заполните все поля");

} else {

user User = user(ui->LoginEdit->text().toStdString(), ui->PasswordEdit->text().toStdString());

if (!User.CheckUserForReg()){

if (!User.SaveUser()){

QMessageBox::critical(this, "Ошибка", "файл");

}else{

StandartUserFrame window;

window.setModal(true);

window.*exec*();

}

} else {

QMessageBox::critical(this, "Ошибка", "Данный пользователь уже существует!");

}

}

}

void MainWindow::**on\_LoginBTN\_clicked**()

{

if ((ui->LoginEdit->text().length() == 0) || (ui->PasswordEdit->text().length() == 0)){

QMessageBox::critical(this, "Ошибка", "Заполните все поля");

} else {

user User = user(ui->LoginEdit->text().toStdString(), ui->PasswordEdit->text().toStdString());

if (User.CheckUser()){

ui->LoginEdit->setText("");

ui->PasswordEdit->setText("");

if (User.GetMode()){

adminFrame window;

window.setModal(true);

window.*exec*();

} else {

StandartUserFrame window;

window.setModal(true);

window.*exec*();

}

} else {

QMessageBox::critical(this, "Ошибка", "Пользователь не найден!");

}

}

}

**search.cpp**

#include "searchbtn.h"

#include "ui\_searchbtn.h"

#include "user.h"

#include "usereditingframe.h"

#include <fstream>

#include <string>

#include <QMessageBox>

#include "user.h"

#include "additionalinformation.h"

#include "dataunit.h"

#include "doublylinkedlist.h"

SearchBTN::**SearchBTN**(QWidget \*parent, int UserIndex) :

QDialog(parent),

ui(new Ui::SearchBTN)

{

ui->setupUi(this);

all.InitializateUserList();

Index = UserIndex;

QString qstr = QString::fromStdString(all.UserList[UserIndex].GetName());

ui->PasswordEdit->setText(qstr);

qstr = QString::fromStdString(all.UserList[UserIndex].GetPassword());

ui->PasswordEdit\_2->setText(qstr);

qstr = QString::number(all.UserList[UserIndex].GetMode());

ui->PasswordEdit\_3->setText(qstr);

}

SearchBTN::~***SearchBTN***()

{

delete ui;

}

void SearchBTN::**on\_pushButton\_5\_clicked**()

{

if ((ui->PasswordEdit->text() != "") && (ui->PasswordEdit\_2->text() != "") && (ui->PasswordEdit\_3->text() != "")

&& ((ui->PasswordEdit\_3->text() == "1") || (ui->PasswordEdit\_3->text() == "0"))){

all.UserList[Index].SetName(ui->PasswordEdit->text().toStdString());

all.UserList[Index].SetPassword(ui->PasswordEdit\_2->text().toStdString());

all.UserList[Index].SetMode(ui->PasswordEdit\_3->text().toInt());

ofstream Users;

Users.open("Users.txt", std::ios::out);

for ( int i = 0; i < all.UserList.size() - 1; i++){

Users << all.UserList[i].GetName() << " ";

Users << all.UserList[i].GetPassword() << " ";

Users << all.UserList[i].GetMode() << " ";

}

Users.close();

ui->pushButton\_3->click();

}

}

**useredditting.cpp**

#include "usereditingframe.h"

#include "ui\_usereditingframe.h"

#include "searchbtn.h"

#include <iostream>

#include <fstream>

#include <string>

#include <QMessageBox>

#include "user.h"

#include "additionalinformation.h"

#include "\_iterator.h"

#include "doublylinkedlist.h"

#include "\_algorithm.h"

using namespace std;

UserEditingFrame::UserEditingFrame(QWidget \*parent) :

QDialog(parent),

ui(new Ui::UserEditingFrame)

{

//заполнение листа из файла и запись его в файл

InitializateUserList();

// ofstream Users;

// Users.open("Userss.txt", std::ios::out);

// Users << this->UserList;

// Users.close();

ui->setupUi(this);

RefreshTable();

}

UserEditingFrame::~UserEditingFrame()

{

delete ui;

}

void UserEditingFrame::InitializateUserList()

{

user tempUser;

string temp;

ifstream Users;

Users.open("Users.txt", ios::in);

//Flights.open("Flights.txt", ios::in);

if (!Users.is\_open()){

QMessageBox::critical(this, "Ошибка", "Файл не может быть открыт");

} else {

while (!Users.eof()){

Users>>temp;

tempUser.SetName(temp);

Users>>temp;

tempUser.SetPassword(temp);

Users>>temp;

if(temp=="1")

tempUser.SetMode(true);

else

tempUser.SetMode(false);

this->UserList.push\_back(tempUser);

}

}

Users.close();

}

void UserEditingFrame::RefreshTable(){

ui->UsersTable->setColumnCount(3);

QStringList table\_names;

table\_names << "Логин" << "Пароль" << "Режим доступа";

ui->UsersTable->setHorizontalHeaderLabels(table\_names);

ifstream InputFile;

InputFile.open("Users.txt", std::ios::in);

int j = 0;

if (InputFile.is\_open()){

int Temp = 0;

while (!InputFile.eof()){

ui->UsersTable->setRowCount(j + 1);

int i = 0;

QTableWidgetItem \*item = new QTableWidgetItem();

string temp;

InputFile >>temp;

QString qstr = QString::fromStdString(temp);

item->setText(qstr);

ui->UsersTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

InputFile >>temp; // ui->UsersTable->setItem(j,i, new QTableWidgetItem(tr("%2").arg(temp))); i++;

qstr = QString::fromStdString(temp);

item->setText(qstr);

ui->UsersTable->setItem(j, i, new QTableWidgetItem(qstr)); i++;

InputFile >> Temp; ui->UsersTable->setItem(j, i, new QTableWidgetItem(tr("%1").arg(Temp))); i++;

i++; j++;

}

ui->UsersTable->setRowCount(j - 1);

}

}

void UserEditingFrame::InitializateFlightList()

{

AdditionalInformation tempInf;

string temp;

ifstream Flights, AdditionalInf;

Flights.open("Flights.txt", ios::in);

AdditionalInf.open("AddInf.txt", ios::in);

if (!Flights.is\_open()||!AdditionalInf.is\_open()){

// QMessageBox::critical(this, "Ошибка", "Файл не может быть открыт");

} else {

while (!Flights.eof()){

Flights>>temp;

tempInf.SetNum(temp);

Flights>>temp;

tempInf.SetDeparture(temp);

Flights>>temp;

tempInf.SetArrieve(temp);

Flights>>temp;

tempInf.SetPrice(temp);

AdditionalInf>>temp;

tempInf.SetNumOfSeats(temp);

AdditionalInf>>temp;

tempInf.SetTimeOfDep(temp);

AdditionalInf>>temp;

tempInf.SetTimeOfArr(temp);

this->FlightList.push\_back(tempInf);

}

}

Flights.close();

AdditionalInf.close();

}

void UserEditingFrame::on\_EditUser\_clicked()

{

DataUnit UserList;

UserList.InitializateUserList();

int Index = UserList.GetUserIndex(ui->NameEdit->text().toStdString());

if (Index != -1){

SearchBTN window(nullptr, Index);

window.setModal(true);

window.exec();

RefreshTable();

}

}

void UserEditingFrame::on\_DeleteUser\_clicked()

{

DataUnit TempUserList;

TempUserList.InitializateUserList();

int Index = TempUserList.GetUserIndex(ui->NameEdit->text().toStdString());

if (Index != -1){

user temp = TempUserList.UserList[Index];

TempUserList.UserList.remove(temp);

ofstream Users;

Users.open("Users.txt", std::ios::out);

for ( int i = 0; i < TempUserList.UserList.size() - 1; i++){

Users << TempUserList.UserList[i].GetName() << " ";

Users << TempUserList.UserList[i].GetPassword() << " ";

Users << TempUserList.UserList[i].GetMode() << " ";

}

Users.close();

RefreshTable();

}

}

**workwithfile.cpp**

#include "workwithfile.h"

WorkWithFile::**WorkWithFile**()

{

}

bool WorkWithFile:: **SaveFile**(std::string filename, string name,string password,bool access\_mode){

bool result = true;

ofstream outputFile;

outputFile.open(filename, ios::app);

if (!outputFile.is\_open()){

result = false;

}else{

outputFile<<name<<" ";

outputFile<<password<<" ";

outputFile<<access\_mode<<" ";

outputFile.close();

}

return result;

}