МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

**Задание**

**на курсовой проект по дисциплине**

**«Объектно-ориентированное программирование»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СТУДЕНТУ ГР. № | 4136 |  |  | П.В.Гришкина |
|  |  |  |  | инициалы, фамилия |

Тема «Разработка приложения для организации взаимодействия объектов при заданных критериях »

**Исходные данные:** Разработка системы сущностей для продажи авиабилетов

Проект должен содержать:

• анализ предметной области

• разработку классов

• разработку тестового приложения

• оформление пояснительной записки по результатам выполнения проекта

• создание презентации к проекту

Срок сдачи законченного проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст.преп. Е.О.Шумова

Дата выдачи задания 01.09.2023 г.

**Содержание**

[Введение 4](#_Toc151567973)

[Основная часть 5](#_Toc151567974)

[1. Постановка задачи 5](#_Toc151567975)

[1.1. Анализ предметной области 5](#_Toc151567976)

[1.2. Формулировка технического задания 6](#_Toc151567977)

[2. Проектирование классов 8](#_Toc151567978)

[2.1. Классы сущностей 8](#_Toc151567979)

[2.2. Управляющие классы 10](#_Toc151567980)

[2.3. Интерфейсные классы 11](#_Toc151567981)

[2.4. Используемые паттерны проектирования 11](#_Toc151567982)

[3. Разработка приложения 15](#_Toc151567983)

[3.1. Разработка интерфейса приложения 15](#_Toc151567984)

[3.2. Реализация классов 18](#_Toc151567985)

[4. Тестирование 22](#_Toc151567986)

[Заключение 37](#_Toc151567987)

[Список использованных источников 38](#_Toc151567988)

[Приложение 1. Исходный код проекта 39](#_Toc151567989)

# Введение

В современном мире, где технологии стали неотъемлемой частью нашей повседневной жизни, создание приложения для покупки авиабилетов становится не только актуальным, но и необходимым шагом для совершенствования и удобства путешествий. Авиационная индустрия стремительно развивается, и потребители ожидают возможности бронировать билеты и управлять своими путешествиями в реальном времени, всегда и везде, где бы они ни находились. В данном контексте создание приложения для покупки авиабилетов обеспечивает ряд важных преимуществ, включая удобство, быстроту, персонализированный выбор, а также снижение нагрузки на традиционные бронировочные системы авиакомпаний. В этой работе мы рассмотрим ключевые причины, почему разработка такого приложения стала неотъемлемой частью стратегии авиационных компаний, и какие преимущества она приносит как самим авиапассажирам, так и бизнесу в целом.

В приложении необходимы функции регистрации пассажиров, авиарейсов, а также продажи и возврата билетов.

Разрабатываемое приложение позволит легко выполнять все эти задачи.

# Основная часть

1. **Постановка задачи**
   1. **Анализ предметной области**

Авиакасса включает в себя следующие функции:

* добавление новых авиарейсов;
* редактирование авиарейсов;
* удаление данных авиарейсов;
* добавление новых пассажиров;
* редактирование пассажиров;
* удаление данных пассажиров;
* продажа билетов на рейс;
* возврат билетов;
* поиск авиарейсов по любому полю;
* поиск пассажиров по любому полю;
* поиск билетов по любому полю;
* фильтр от большего к меньшему и наоборот по любым данным для пассажиров;
* фильтр от большего к меньшему и наоборот по любым данным для авиарейсов;
* фильтр от большего к меньшему и наоборот по любым данным для билетов.

Можно выделить следующие сущности:

* Продажа/возврат билетов;
* Пассажиры;
* Авиарейсы.

Объекты сущности «Пассажиры» содержат информацию о пассажирах:

* фамилия, имя, отчество;
* номер паспорта;
* дата рождения.

Объекты сущности «Авиарейсы» содержат информацию о перелётах:

* номер авиарейса;
* авиакомпания;
* аэропорт прибытия;
* аэропорт отправления;
* дата и время отправления;
* количество мест всего.

Объекты сущности «Билеты» содержат информацию о билетах на авиарейс:

* номер паспорта пассажира;
* номер авиарейсов;
* номер билета.
  1. **Формулировка технического задания**
     1. **ВВЕДЕНИЕ**

Настоящее техническое задание, оформленное в соответствии с ГОСТ 19.201-78, содержит требования к информационной системе для обеспечения продажи и возврата билетов.

* + 1. **ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ**
       1. **Основание для разработки**

Основанием для разработки информационной системы является необходимость быстрого доступа для изменения хранимой информации с целью упрощения работы билетных кассиров и быстрого структурирования данных.

* + - 1. **Исполнитель и заказчик**

Заказчиком разработки, выполняемой по настоящему ТЗ, является компания Go Travel Un Limited.

Исполнителем разработки, выполняемой по настоящему ТЗ, является студентка группы 4136 П.В.Гришкина.

* + - 1. **Наименование**

Программе, разрабатываемой по настоящему ТЗ, присваивается наименование: «Информационная система авиакассы».

* + 1. **НАЗНАЧЕНИЕ РАЗРАБОТКИ**

Программа создана для продажи билетов на рейс. Пользователь может ввести данные пассажиров, авиарейсов и начать продавать или возвращать на них билеты. Информационная система предназначена для выполнения следующих действий с классами:

* создание объектов сущностей;
* заполнение полей объектов сущностей данными;
* изменение данных полей объектов;
* управление объектами классов сущностей посредствам управляющих классов;
* удаление созданных объектов классов;
* вывод данных полей объектов сущностей с использованием графического интерфейса.

1. **Проектирование классов**
   1. **Классы сущностей**

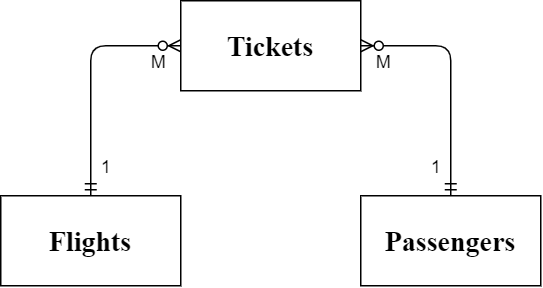


Рисунок 1 Отношения изначальных классов

По итогу, можно выделить следующие классы сущностей:

Flights – класс, объекты которого содержат данные о перелётах.

Поля:

* Number – номер рейса;
* Airline – авиакомпания;
* Arrival – аэропорт отправления;
* Departure – аэропорт прибытия;
* Date – дата отправления;
* Total\_places – количество мест всего;
* Free\_places – количество свободных мест.

Методы:

* setFlightsModel(QAbstractTableModel \*model) – установить модель данных виджета TableView;
* handleDeleteFlightButtonClicked() – обработчик нажатия на кнопку "Удалить рейс";
* handleAddFlightButtonClicked() - обработчик нажатия на кнопку "Добавить рейс";
* handleAddFlightButtonActivated() – активатор для кнопки "Добавить рейс" при заполнении всех полей ввода;
* handleUpdateFilterParams() – обработчик изменения параметров фильтрации;
* handleChangeFlightButtonClicked() - обработчик нажатия на кнопку "Редактировать рейс".

Passengers – класс, объекты которого содержат данные о пассажирах.

Поля:

* Name – ФИО;
* Passport – номер паспорта;
* BirthDay – дата рождения.

Методы:

* setPassengersModel(QAbstractTableModel \*model) – установить модель данных виджета TableView;
* handleDeletePassengerButtonClicked() – обработчик нажатия на кнопку "Удалить пассажира";
* handleAddPassengerButtonClicked() - обработчик нажатия на кнопку "Добавить пассажира";
* handleAddPassengerButtonActivated() – активатор для кнопки "Добавить пассажира " при заполнении всех полей ввода;
* handleUpdateFilterParams() – обработчик изменения параметров фильтрации;
* handleChangePassengerButtonClicked() - обработчик нажатия на кнопку "Редактировать пассажира".

Tickets – класс, объекты которого содержат данные о билетах на авиарейс.

Поля:

* TicketNumber – номер билета;
* Passport – номер паспорта;
* Flight – номер рейса.

Методы:

* setModels(QAbstractTableModel \*ticketModel, QAbstractTableModel \*passengerModel, QAbstractTableModel \*flightModel) – установить модель данных виджета TableView;
* handleDeleteTicketButtonClicked() – обработчик нажатия на кнопку "Удалить билет";
* handleAddTicketButtonClicked() - обработчик нажатия на кнопку "Добавить билет";
* handleUpdateFilterParams() – обработчик изменения параметров фильтрации;
* handleChangeTicketButtonClicked() - обработчик нажатия на кнопку "Редактировать билет".
  1. **Управляющие классы**

Для оптимизации взаимосвязи между классами и пользователем, был реализован управляющий класс: TicketsController.

Поля:

* \_passengerModel– указатель на модель "Пассажир";
* \_ticketModel– указатель на модель "Билет";
* \_flightModel– указатель на модель "Рейс".

Методы:

* handleShowTickets() – отобразить параметры билета
* handleAddTicket(const QModelIndex &flightIndex, const QModelIndex &passengerIndex) – обработчик добавления билета;
* handleRemovePassenger(const QString &passport) - обработчик удаления пассажира;
* handleRemoveFlight(const QString &passport) - обработчик удаления рейса;
* handleRemoveTicket(const QString &passport) - обработчик удаления билета.
  1. **Интерфейсные классы**

QAbstractTableModel – интерфейс модели данных в виде таблицы.

* 1. **Используемые паттерны проектирования**

Шаблон проектирования «Фабричный метод» используется для создания прокси-моделей сортировки и фильтрации для табличных моделей данных.



Рисунок 2.1Фабричный метод

Структурный паттерн "Заместитель" используется в программе с целью реализации взаимодействия виджета с моделью данных через прокси-модель по общему интерфейсу.

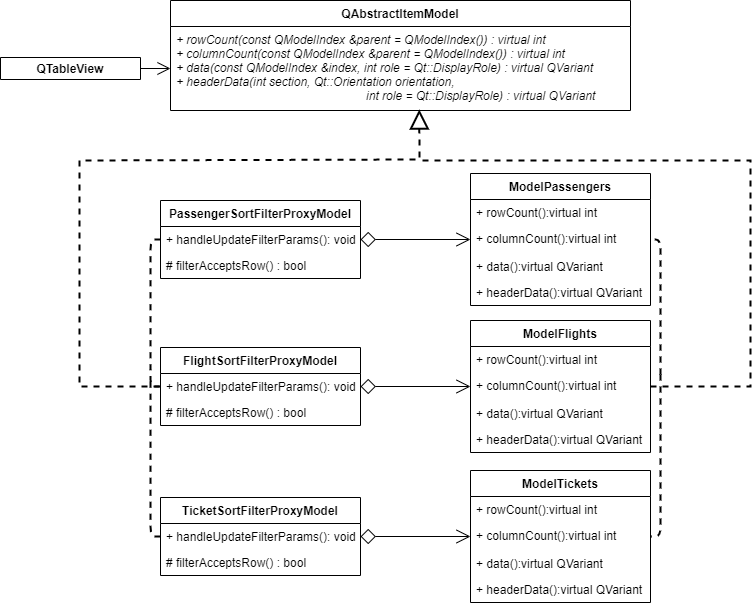


Рисунок 2.2 Заместитель

"Посредник" – поведенческий паттерн добавлен в программу для взаимодействия данных и их отображения.

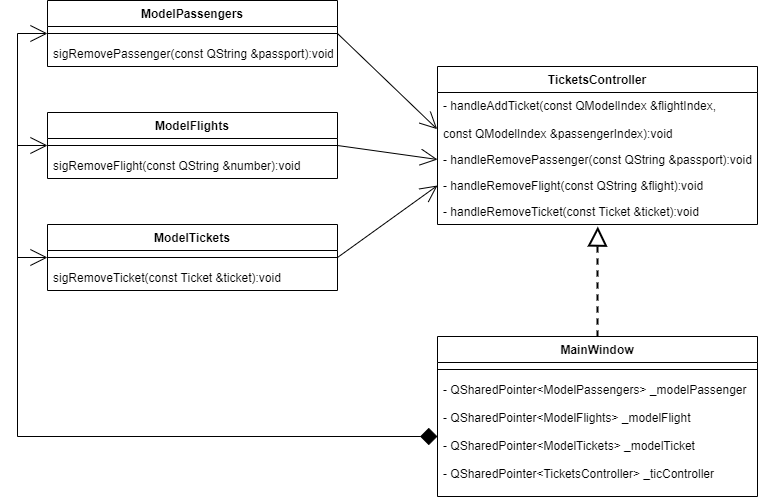


Рисунок 2.3 Контроллер (посредник)

Паттерн "Наблюдатель" применяется для взаимодействия контроллера с моделями данных, при этом контроллер знает об их существовании, но не владеет ими.

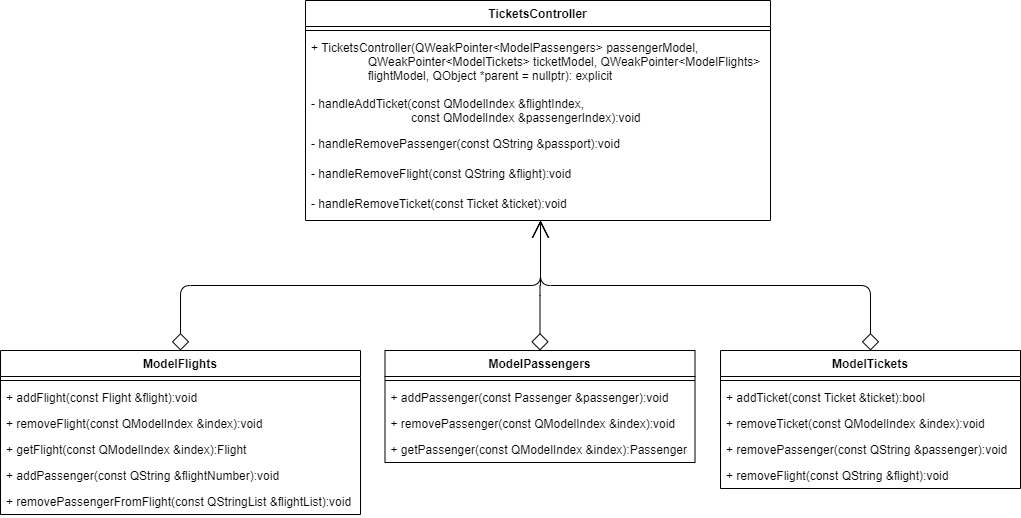


Рисунок 2.4 Наблюдатель

Архитектурный паттерн "MVC" упорядочивает взаимодействие с данными и их отображения.

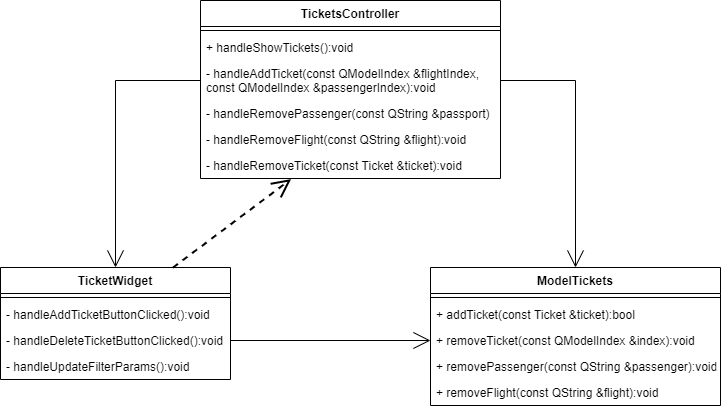


Рисунок 2.5 MVC

А также не явно выделены: "Цепочка обязанностей" – поведенческий, "Компоновщик" – структурный, "Итератор" – поведенческий.

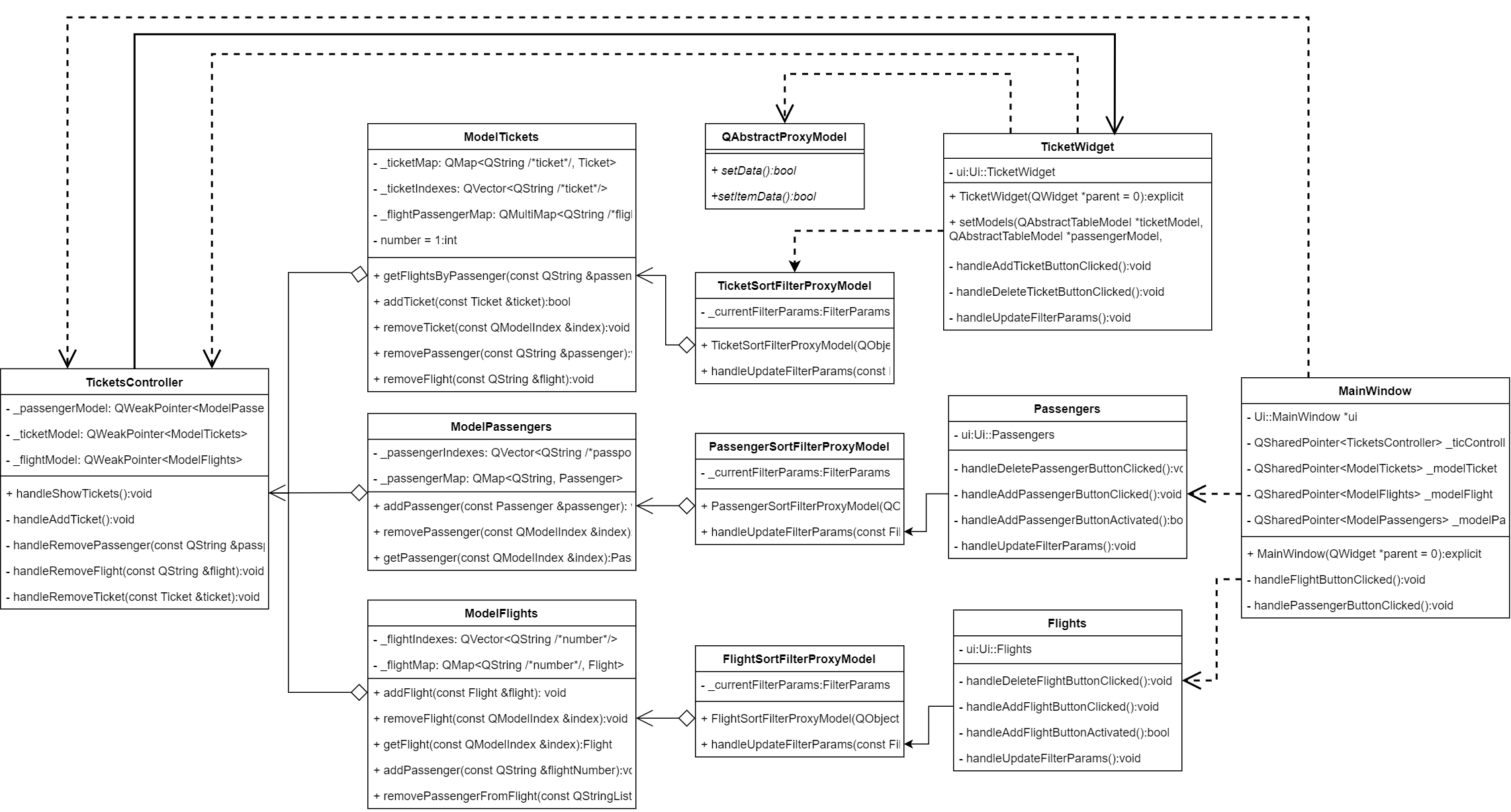


Рисунок 2.6 Окончательная диаграмма классов

1. **Разработка приложения** 
   1. **Разработка интерфейса приложения**

Интерфейса приложения начался с разработки главного меню, которое открывается при запуске программы. На нём представлены три активные кнопки, при нажатии на которые должен происходить переход в другие формы с таблицами, а также два изображения.

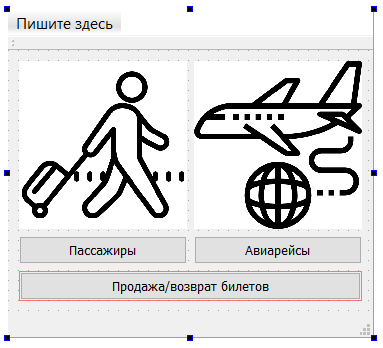


Рисунок 3.1.1 Оконная форма главного меню

После нажатия на крайнюю левую кнопку "Пассажиры" должен происходить переход пользователя в форму с данными о пассажирах. На ней показаны 2 label поля с соответствующими им lineEdit объектами. В каждом из которых присутствует проверка на валидность данных: ФИО можно вводить латиницей/кириллицей с использованием дефиса разным регистром, в таблицу оно в итоге занесётся корректно (первая буква является заглавной, остальные - строчные); номер паспорта состоит из 4-х цифр и через пробел 10- ти чисел серии. Около label с датой рождения находится поле с dateEdit, значения в котором принадлежат отрезку от 01.01.1890 до 20.11.2023. Далее расположены три кнопки для взаимодействия с записями таблицы: добавление, удаление и редактирование. Единственный checkBox служит для фильтрации данных по заполненным line- и date- Edit. В самом низу была добавлена tableView, в которой при запуске программы отобразятся три шаблонных пассажира. Изначально все кнопки, кроме "Добавить пассажира", являются активными. Как только пользователь корректно заполняет все данные в полях, на кнопку становится можно нажать.

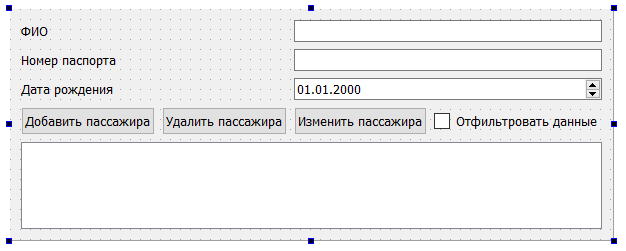


Рисунок 3.1.2 Оконная форма пассажира

После нажатия на крайнюю правую кнопку "Авиарейсы" должен происходить переход пользователя в форму с данными об авиаперелётах. На ней показаны 5 label полей с соответствующими им lineEdit объектами. В каждом из которых присутствует проверка на валидность данных: номер авиарейса состоит из двух заглавных букв латиницы и записанного через пробел четырёхзначного числа; названия авиакомпании, аэропортов прибытия/отправления – латиница или кириллица любого регистра с возможным использованием дефиса; количество мест в самолёте это натуральное число, исключая специальные символы. Около label с датой отправления находится поле с dateTimeEdit, значения в котором не должны быть в прошлом. Далее расположены три кнопки для взаимодействия с записями таблицы: добавление, удаление и редактирование. Единственный checkBox служит для фильтрации данных по заполненным line- и dateTime- Edit. В самом низу была добавлена tableView, в которой при запуске программы отобразятся три шаблонных авиарейса. Изначально все кнопки, кроме "Добавить авиарейс", являются активными. Как только пользователь корректно заполняет все данные в полях, на кнопку становится можно нажать.

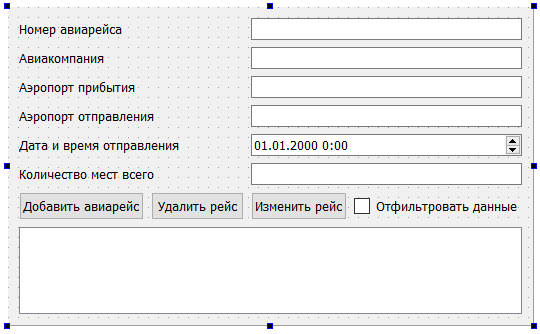


Рисунок 3.1.3 Оконная форма авиарейсов

После нажатия на центральную кнопку "Продажа/возврат билетов" должен происходить переход пользователя в форму с данными о билетах на рейсы. На ней показаны 3 label поля с соответствующими им lineEdit объектами. В каждом из которых присутствует проверка на валидность данных: номер билета состоит из цифр, номера паспорта и авиарейса аналогичны предыдущим формам. В середине присутствуют две кнопки для взаимодействия трёх таблиц: продажа и возврат билетов. Единственный checkBox служит для фильтрации данных по заполненным lineEdit. Слева представлены две tableView, на основе которых заполняется новая таблица справа, в которой при запуске программы отобразятся три шаблонных билета на рейсы. Все кнопки являются активными с начала работы программы.

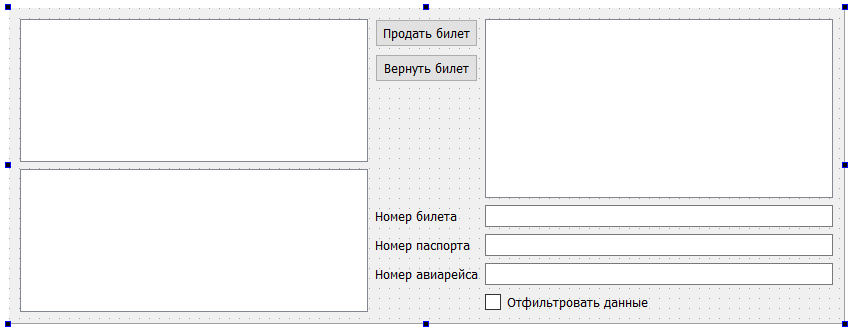


Рисунок 3.1.4 Оконная форма билетов

* 1. **Реализация классов**

Объект Пассажир реализуется в программе с помощью класса Passenger, который хранит основные данные о пассажире: ФИО, паспорт и дата рождения.

struct Passenger

{

Passenger() = default;

QString name;

QString passport;

QDate birthday;

};

Для хранения объектов данного класса, мы создаём класс ModelPassengers, который реализует операции с пассажирами (добавление, удаление, редактирование).

Этот класс наследуется от QAbstractTableModel. Взаимодействие между отображением и данными осуществляется с помощью интерфейса наследуемого от QAbstractTableModel. Описание интерфейса на примере пассажиров:

Возвращает количество строк

virtual int rowCount(const QModelIndex &parent = QModelIndex()) const override;

Возвращает количество столбцов

virtual int columnCount(const QModelIndex &parent = QModelIndex()) const override;

Возвращает данные по определенному индексу

virtual QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const override;

Возвращает заголовок по определенному индексу

virtual QVariant headerData(int section, Qt::Orientation orientation,

int role = Qt::DisplayRole) const override;

Каждый объект имеет свои свойства, конструктор и реализацию наследуемых методов.

Объекты класса ModelPassengers имеют такие поля:

Индексы пассажиров, соответствующие номерам строк в отображаемой таблице

QVector<QString /\*passport\*/> \_passengerIndexes;

Ассоциативный контейнер: ключом является паспорт, значением – структура данных пассажира.

QMap<QString, Passenger> \_passengerMap;

Архитектура взаимодействия классов Flight, Ticket и моделей ModelFlights, ModelTickets реализована аналогично, за исключением ассоциативного контейнера QMultiMap в ModelTickets (в нём одному ключу номера рейса может соответствовать несколько значений пассажиров).

У классов Model… имеются методы для взаимодействия с хранящимися в контейнере объектами:

• Добавление элемента в контейнер;

• Удаление элемента из контейнера;

• Редактирование элемента в контейнере;

• Возвращение объекта по индексу.

Взаимодействие между пользователем и бизнес-логикой описывается в классе MainWindow. В данном классе создаётся объект главного окна приложения и объекты классов моделей, которые играют роль контейнеров для соответствующих типов объектов и контролер, обеспечивающий взаимодействие между моделями данных.

\_modelPassenger = QSharedPointer<ModelPassengers>::create();

\_modelFlight = QSharedPointer<ModelFlights>::create();

\_modelTicket = QSharedPointer<ModelTickets>::create();

\_ticController = QSharedPointer<TicketsController>

::create(\_modelPassenger.toWeakRef(),

\_modelTicket.toWeakRef(),\_modelFlight.toWeakRef());

Далее идёт описание методов, вызываемых пользователем при взаимодействии с главным окном приложения. Описываемые методы:

void handlePassengerButtonClicked() - обработчик нажатия кнопки "Пассажиры", отображает виджет Passengers для взаимодействия с данными пассажиров.

void handleFlightButtonClicked()- обработчик нажатия кнопки "Авиарейсы", отображает виджет Flights для взаимодействия с данными авиарейсов.

Нажатие на кнопку "Продажа/возврат билет" обрабатывается TicketsController в методе void handleShowTickets()

Для тестирования работы приложения были созданы данные по умолчанию, заполненные в конструкторе моделей.

\_flightMap.insert("FI 1111",{

"FI 1111",

"S7",

"LED",

"DME",

QDateTime(QDate(2024, 7, 23), QTime(8, 30, 0)),

3,

3

});

\_flightIndexes.append("FI 1111");

\_passengerMap.insert("4370 111111",{

"Ivanov Ivan Ivanovich",

"4370 111111",

QDate (2000,12,01)

});

\_passengerIndexes.append("4370 111111");

Ticket ticket;

ticket.flight = "FI 1111";

ticket.passport = "4370 111111";

if (ticModRef->addTicket(ticket))

flightModRef->addPassenger(ticket.flight);

# Тестирование

После запуска программы перед пользователем предстает оконная форма (рис.3.3.1).

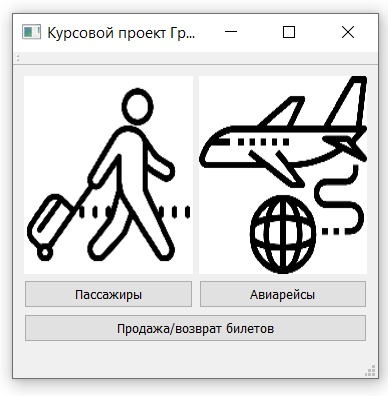


Рисунок 3.3.1 Главное окно

После нажатия на кнопку "Пассажиры" происходит переход в следующую форму (рис. 3.3.2).

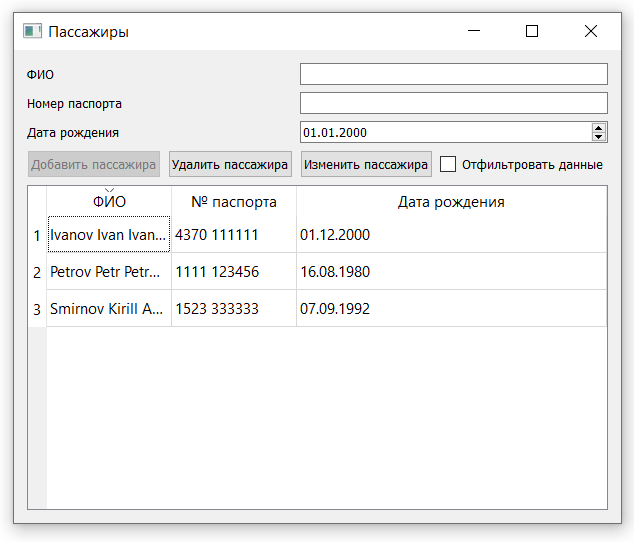


Рисунок 3.3.2 Окно с пассажирами

Изначально кнопка "Добавить" неактивна, но после ввода всех данных о пассажире на неё можно нажать (рис. 3.3.3). Пример добавления пассажира и его мгновенное отображение в таблице после нажатия на кнопку "Добавить" (рис. 3.3.4).

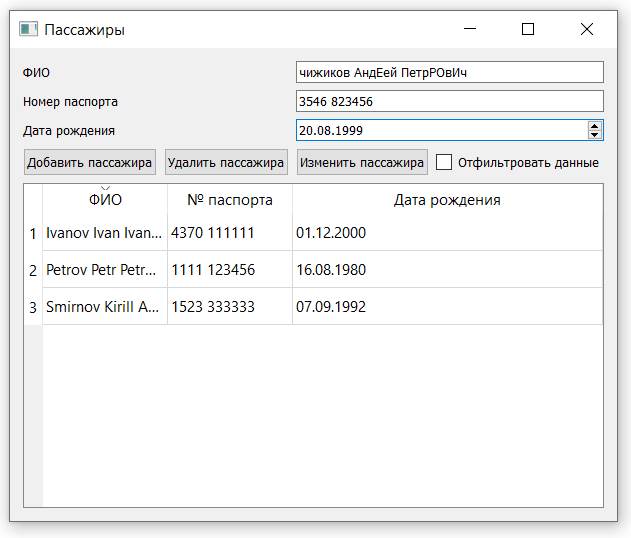


Рисунок 3.3.3 Заполнение данных

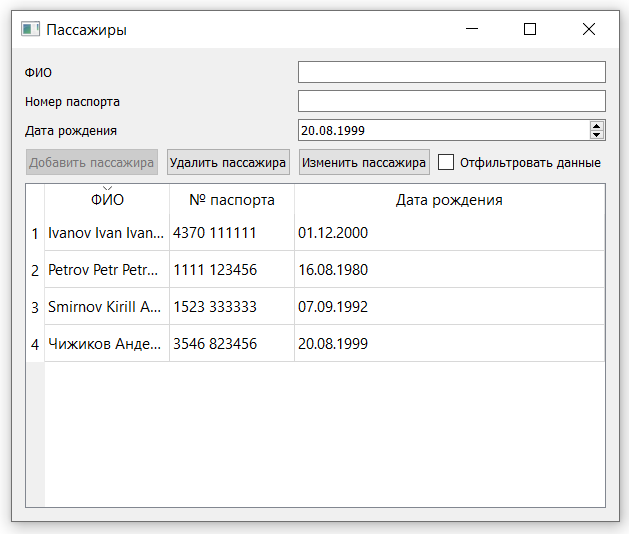


Рисунок 3.3.4 Успешное добавление

При попытке удалить пассажира, у которого куплен билет на рейс, будет выведена ошибка (рис. 3.3.5, 3.3.6).

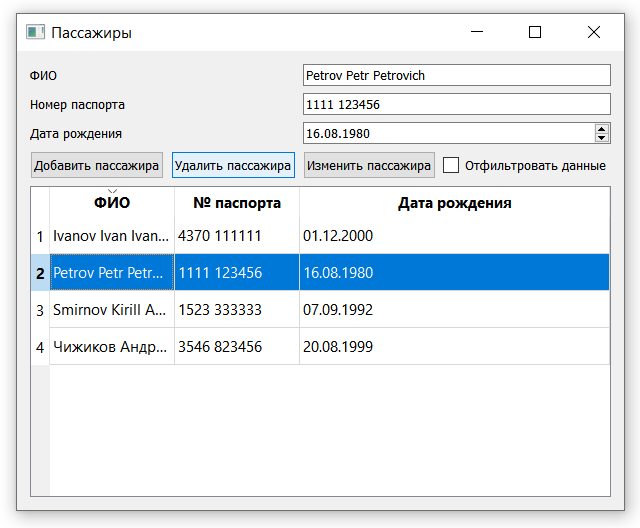


Рисунок 3.3.5 Удаление

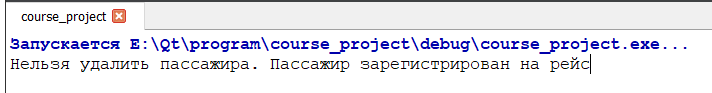


Рисунок 3.3.6 Вывод ошибки

Успешное удаление пассажира (рис. 3.3.7, 3.3.8).

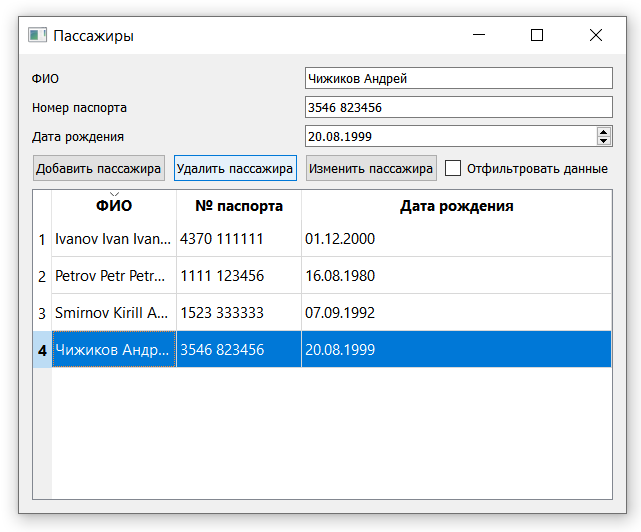


Рисунок 3.3.7 Выбор пассажира

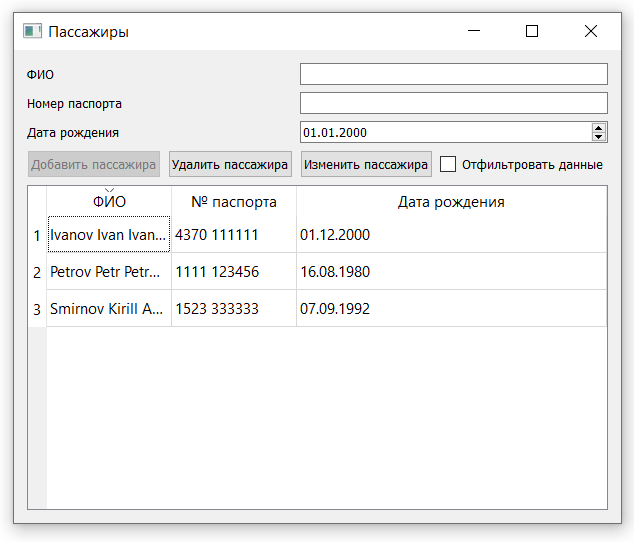


Рисунок 3.3.8 Пассажир удален

Для того чтобы изменить данные пассажира, пользователю надо выбрать его в таблице, после чего изменить данные и нажать на кнопку "Изменить пассажира" (рис. 3.3.9, 3.3.10). Данные паспорта менять запрещено, если пользователь попытается, то будет вызвана соответствующая ошибка (рис. 3.3.10, 3.3.11).

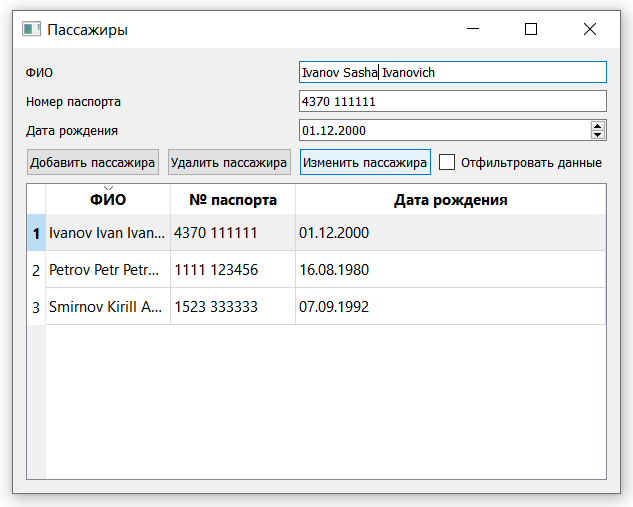


Рисунок 3.3.9 Выбор изменяемого пассажира

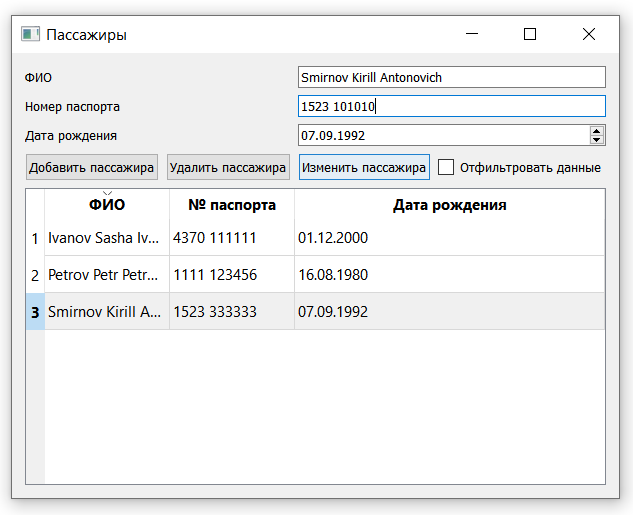


Рисунок 3.3.10 Измененный пассажир, замена паспорта



Рисунок 3.3.11 Ошибка из-за замены паспорта

Фильтрация данных происходит после проставления галочки в checkbox-е и заполнения интересующих для поиска полей (фильтрация происходит по ФИО и номеру паспорта) (рис.3.3.12).

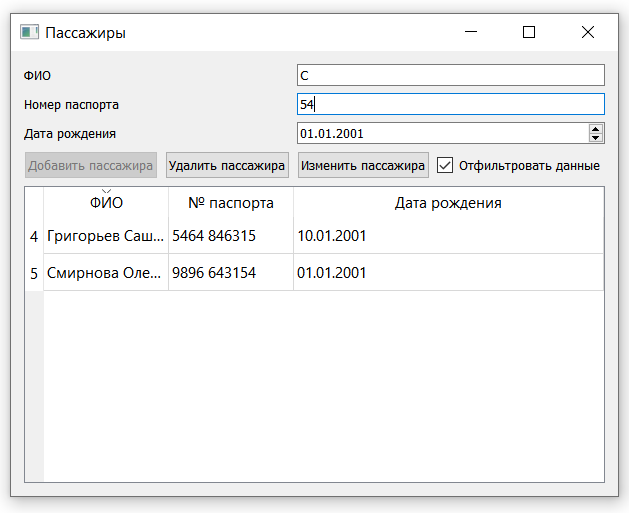


Рисунок 3.3.12 Фильтрация данных

При нажатии на "Авиарейсы" в главном окне происходит переход в оконную форму с данными об авиаперелётах.

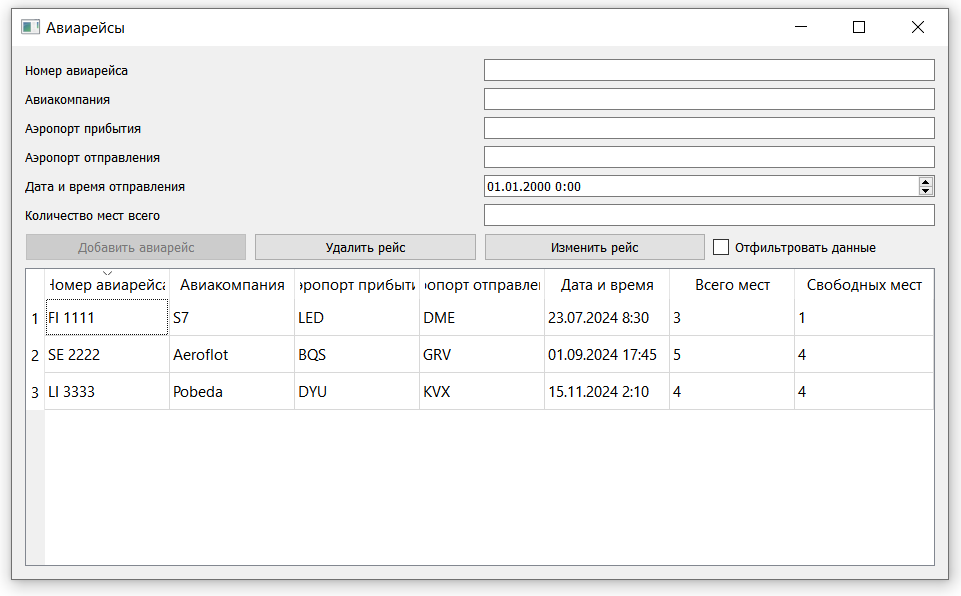


Рисунок 3.3.13 Оконная форма с авиарейсами

Изначально кнопка "Добавить" неактивна, но после ввода всех данных о рейсе на неё можно нажать (рис. 3.3.14). Пример добавления авиарейса и его мгновенное отображение в таблице после нажатия на кнопку "Добавить" (рис. 3.3.15).

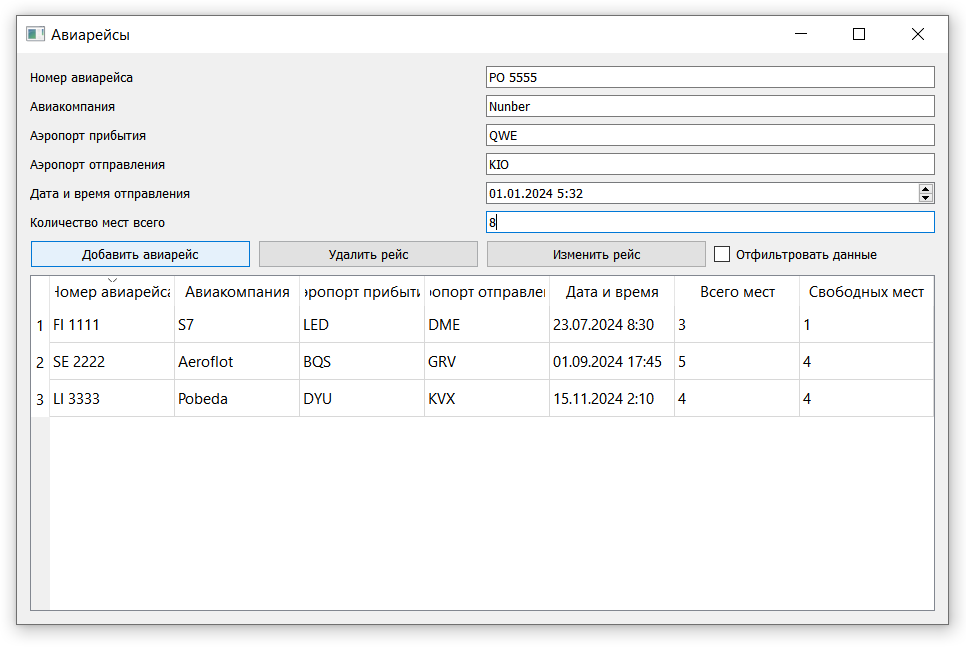


Рисунок 3.3.14 Добавление рейса

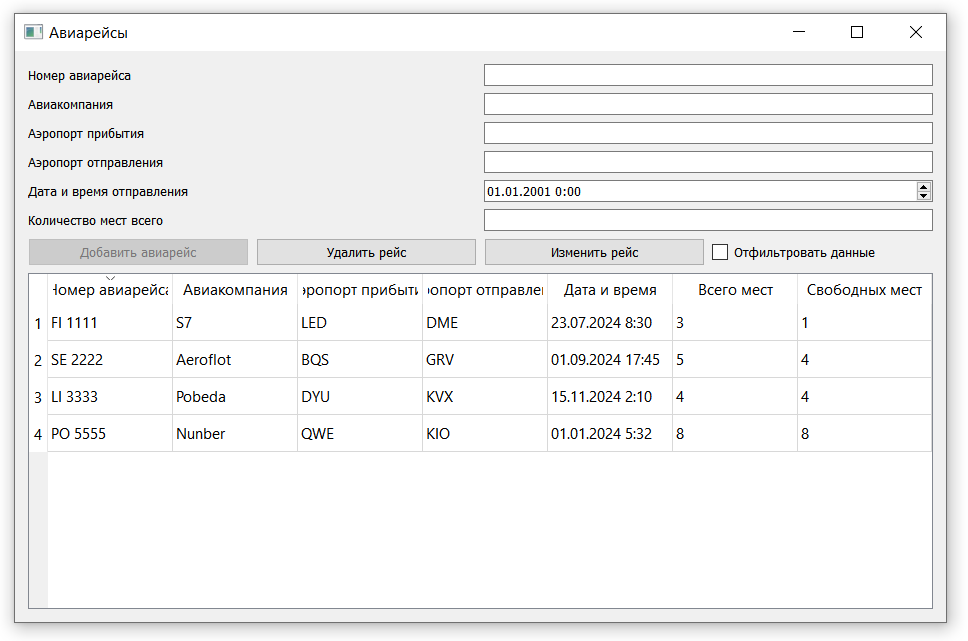


Рисунок 3.3.15 Новый рейс добавлен

Для того чтобы удалить рейс нужно выделить его в таблице и нажать на кнопку "Удалить". Успешное удаление продемонстрировано на рис. 3.3.16, 3.3.18)

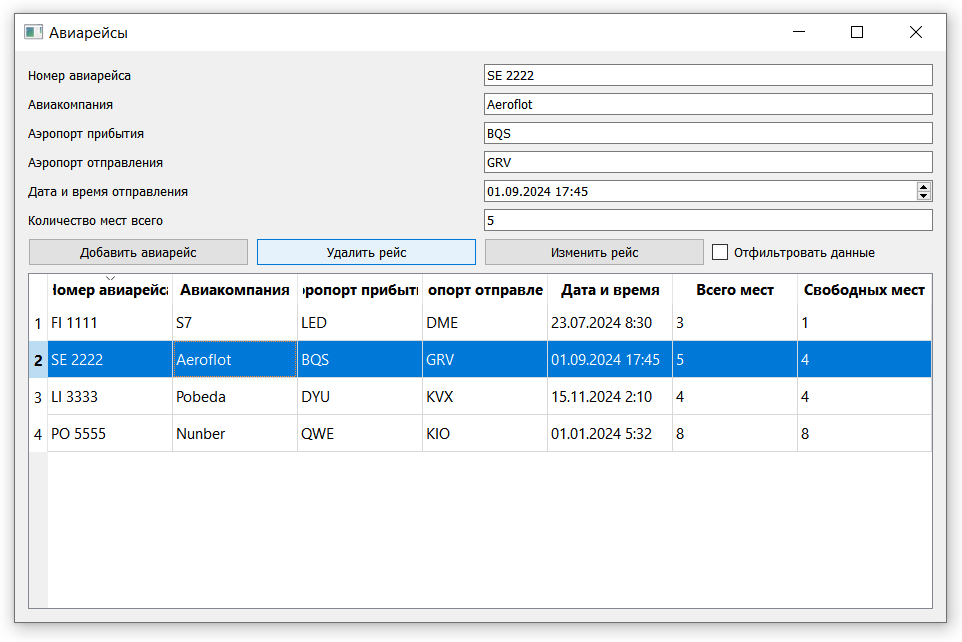


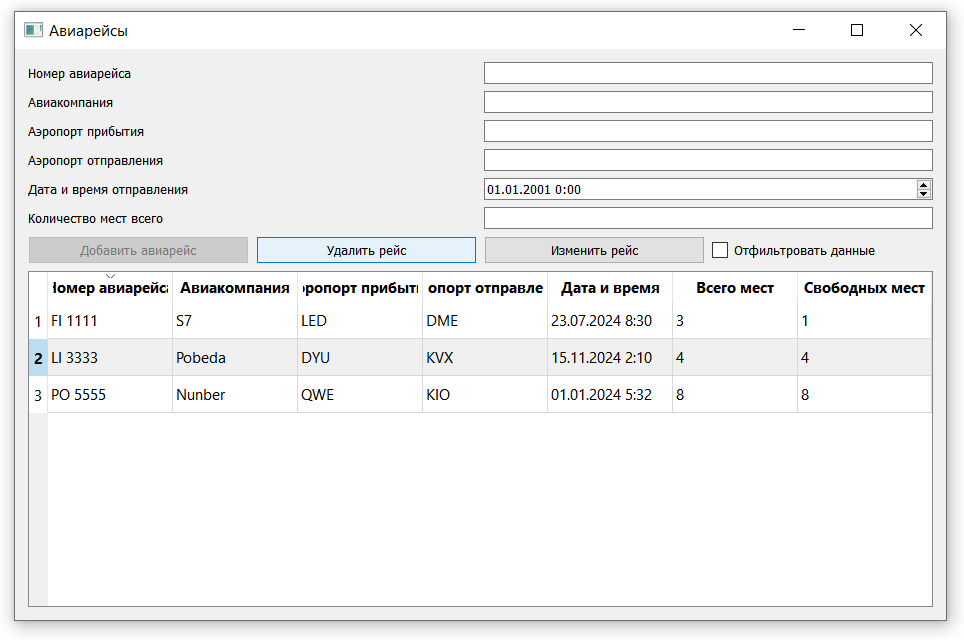
Рисунок 3.3.17 Удаление рейса

Рисунок 3.3.18 Рейс успешно удален

Для того чтобы изменить данные рейса, пользователю надо выбрать его в таблице, после чего изменить данные и нажать на кнопку "Изменить рейс" (рис. 3.3.21, 3.3.22). Данные номера авиарейса менять запрещено, если пользователь попытается, то будет вызвана соответствующая ошибка (рис. 3.3.19, 3.3.20).

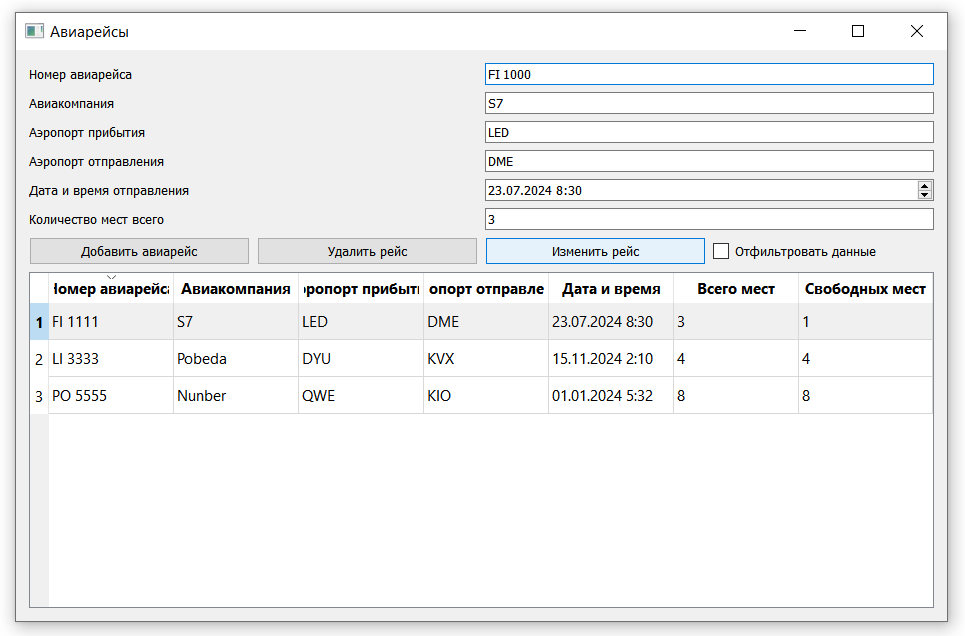


Рисунок 3.3.19 Попытка изменить номер рейса

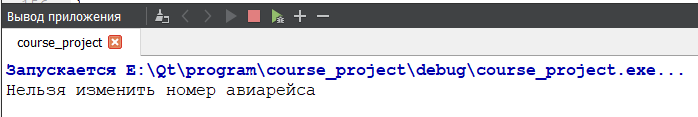


Рисунок 3.3.20 Ошибка, вызванная изменением номера рейса

Пример успешного изменения данных:

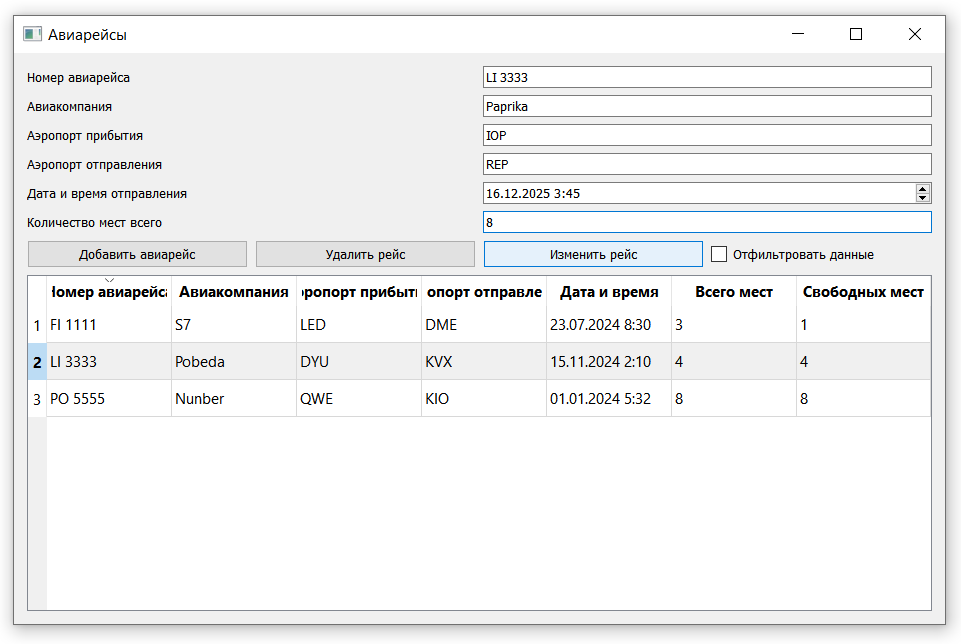


Рисунок 3.3.21 Ввод новых данных

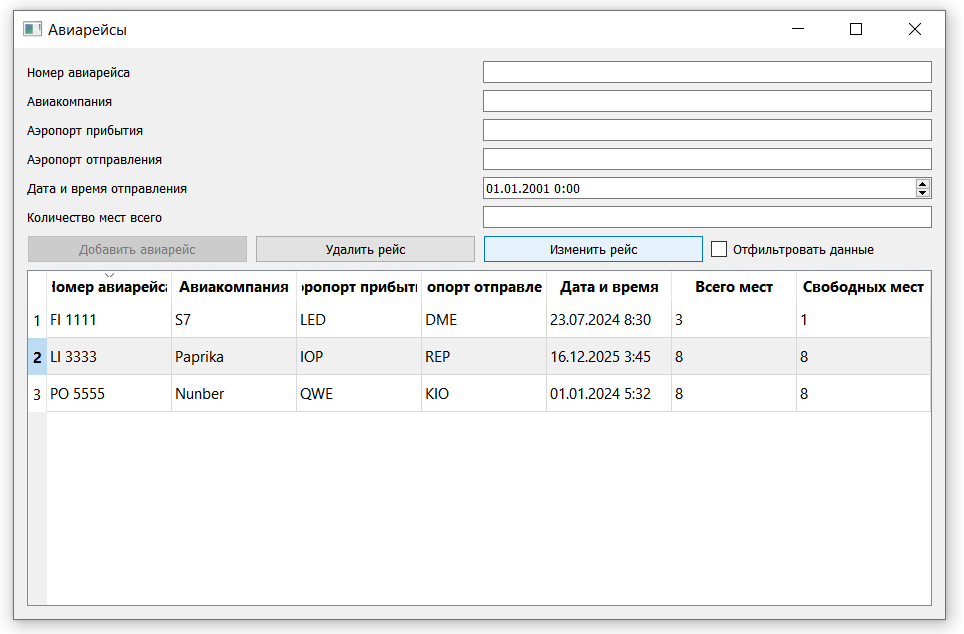


Рисунок 3.3.22 Успешное обновление данных

При добавлении пользователь не может вводить дату отправления в прошлом и нулевое количество мест, а также номер авиарейса должен быть уникальным, а аэропорты отправления и прибытия – разные.

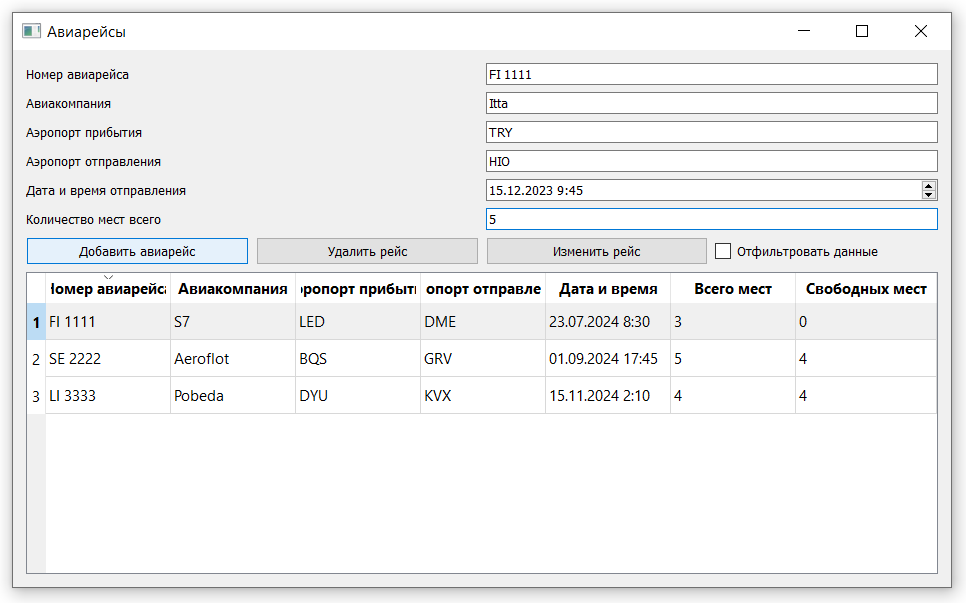


Рисунок 3.3.23 Добавление существующего рейса

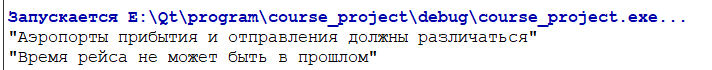


Рисунок 3.3.24 Ошибки, вызванная одинаковыми аэропортами и датой отправления в прошлом

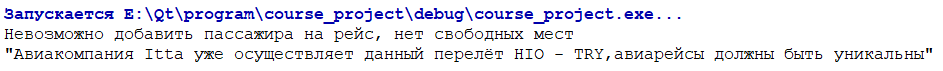


Рисунок 3.3.25 Ошибка, вызванная неуникальностью рейса

Если пользователь выберет для изменения рейс, на который были проданы билеты и укажет количество мест меньше количества проданных на него билетов, то программа выдаст ошибку.

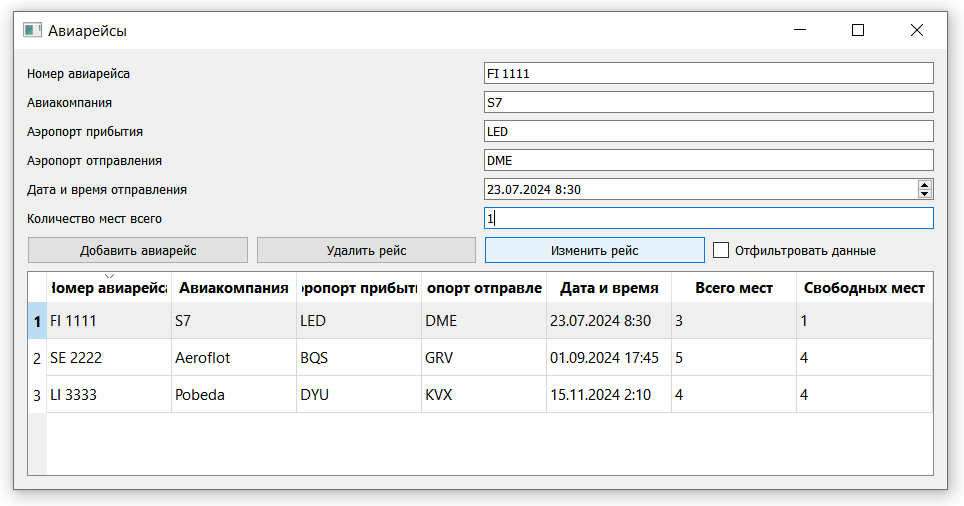


Рисунок 3.3.26 Изменение с меньшим количеством мест



Рисунок 3.3.27 Ошибка, вызванная

недостатком мест при изменении

А также при изменении нельзя назначать рейс на дату в прошлом. Будет вызвана ошибка представленная ниже.

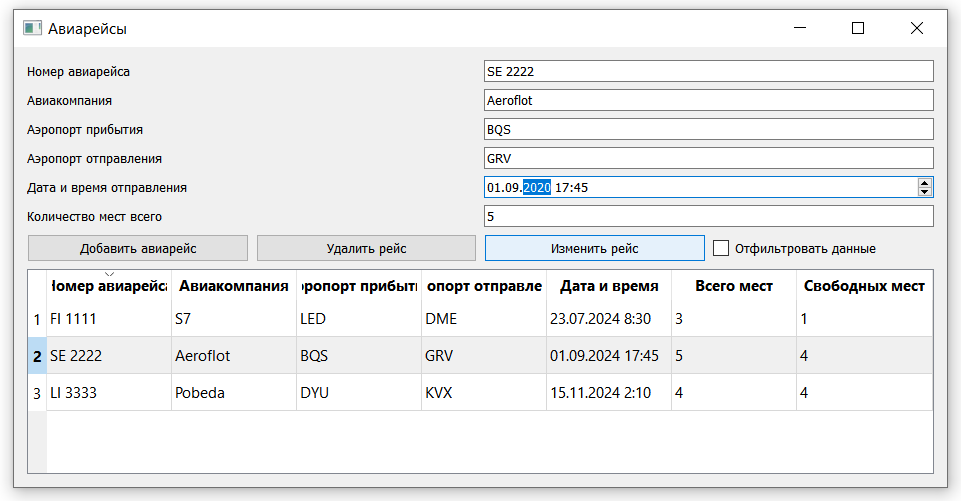


Рисунок 3.3.28 Изменение на рейс в прошлом

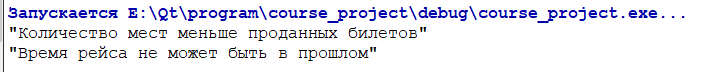


Рисунок 3.3.29 Ошибки, вызванная недостатком мест и

датой отправления в прошлом

Фильтрация данных происходит после проставления галочки в checkbox-е и заполнения интересующих для поиска полей (фильтрация происходит по номеру рейса, авиакомпании, аэропортам прибытия и отправления) (рис.3.3.30).

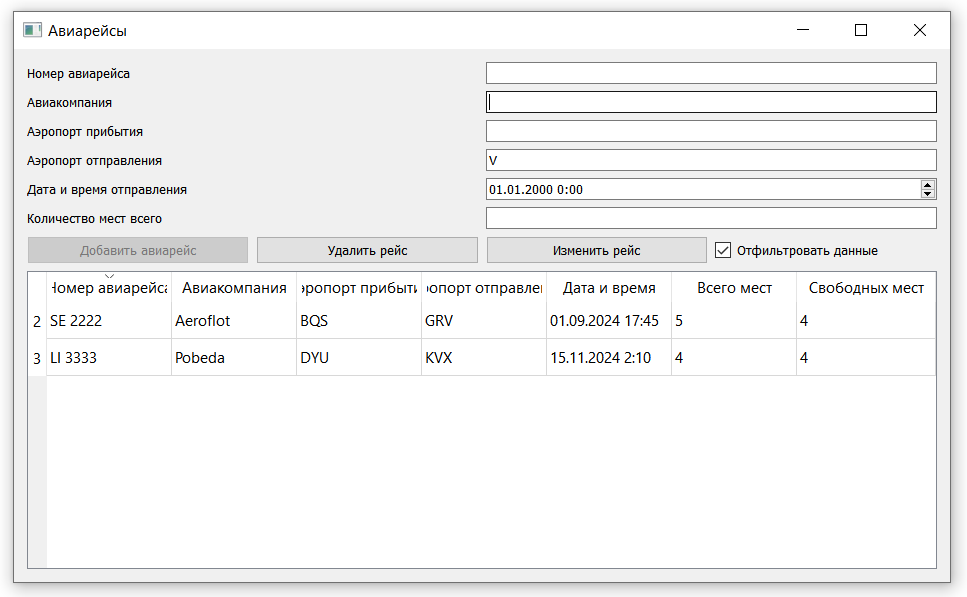


Рисунок 3.3.30 Фильтрация данных

При нажатии на "Продажа/возврат билетов" в главном окне происходит переход в оконную форму с данными о билетах (рис. 3.3.31).

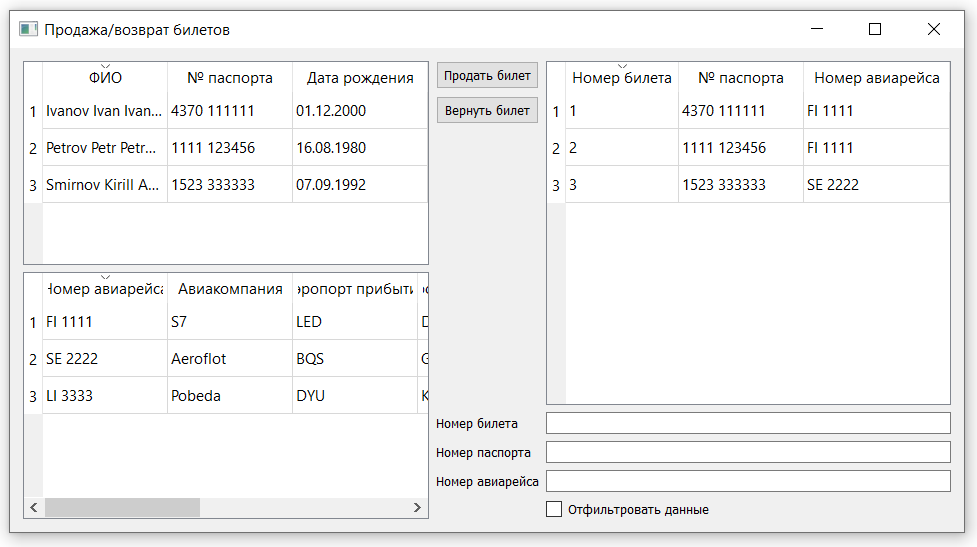


Рисунок 3.3.31 Оконная форма с билетами

Чтобы купить билет пользователь выбирает из таблиц по одному пассажиру и рейс, после чего нажимает на кнопку "Продать билет" (рис. 3.3.32-33).

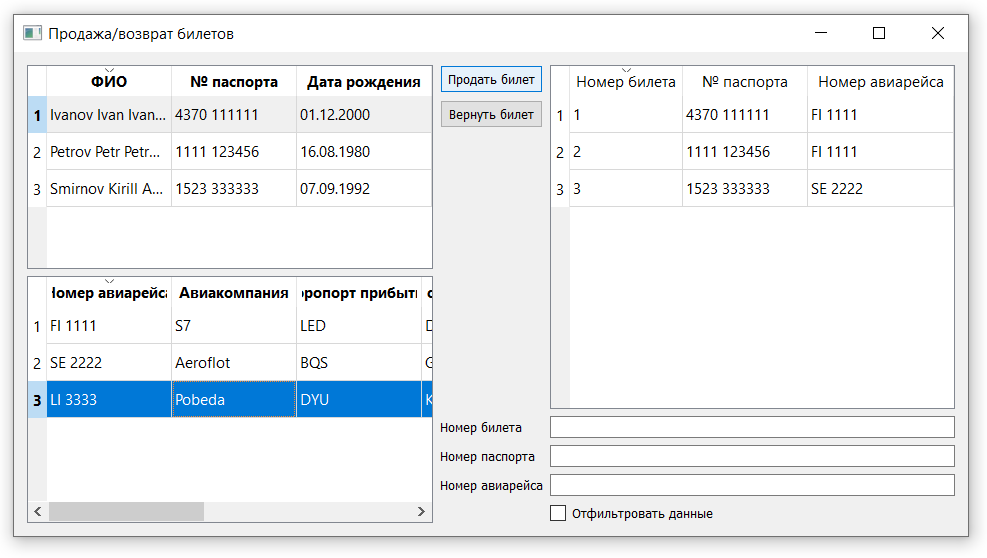


Рисунок 3.3.32 Выбор строк из таблиц

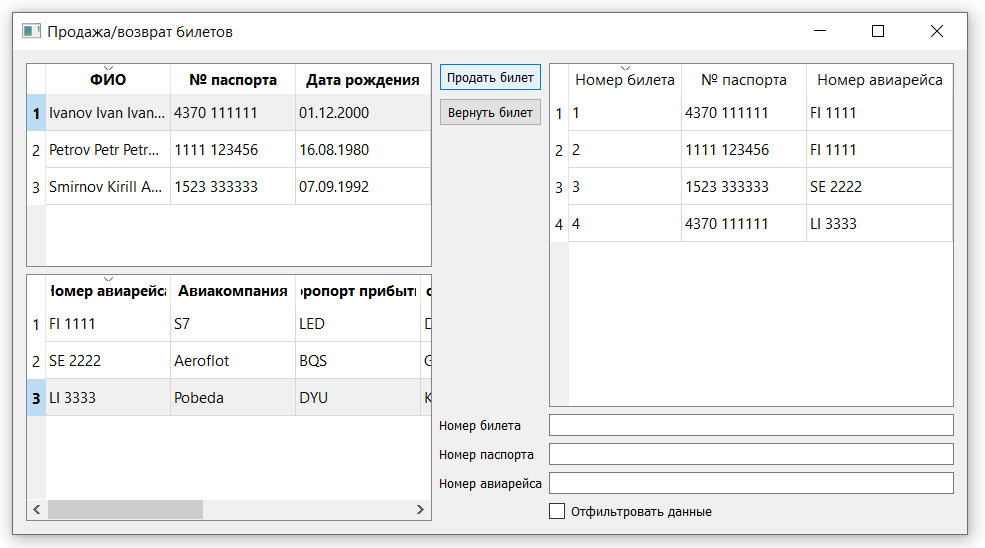


Рисунок 3.3.33 Успешная продажа

Если на рейс нет свободных мест, то будет вызвана ошибка, билеты не будут проданы.



Рисунок 3.3.34 Ошибка, вызванная недостатком мест

Для возврата билетов пользователь выбирает из таблицы одну строку, после чего нажимает – "Вернуть билет". В соответствующих таблицах также будут изменены данные: в таблице с рейсом появится одно свободное место, а в таблице с пассажиром появится возможность его удалить, если на руках больше нет билетов.

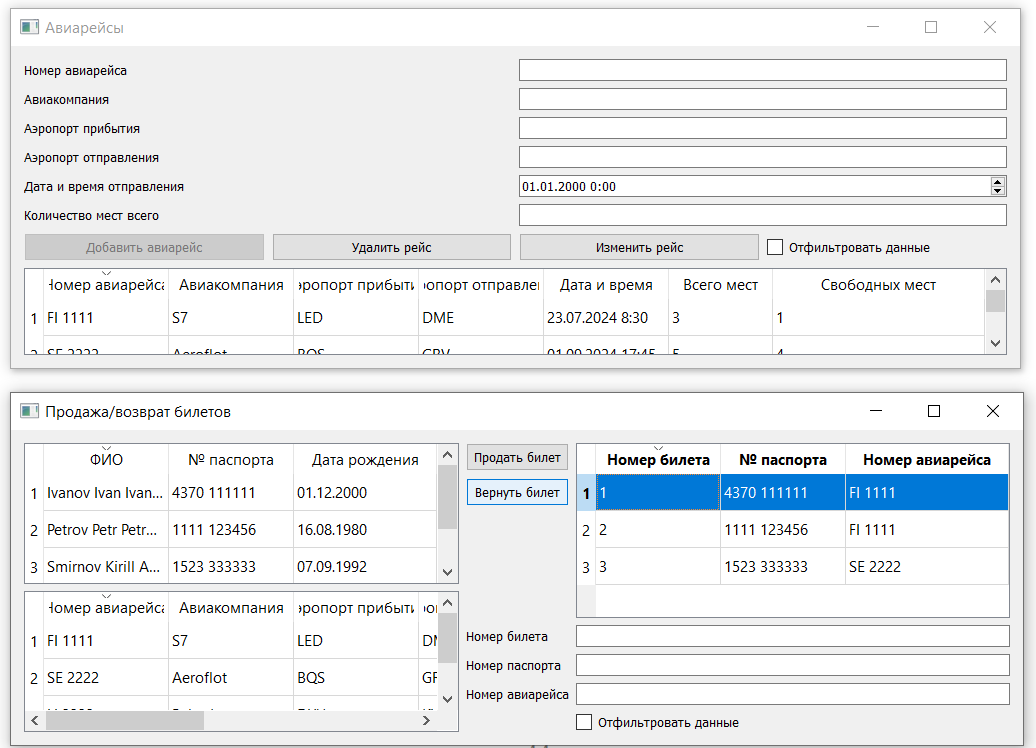


Рисунок 3.3.35 До возврата билета

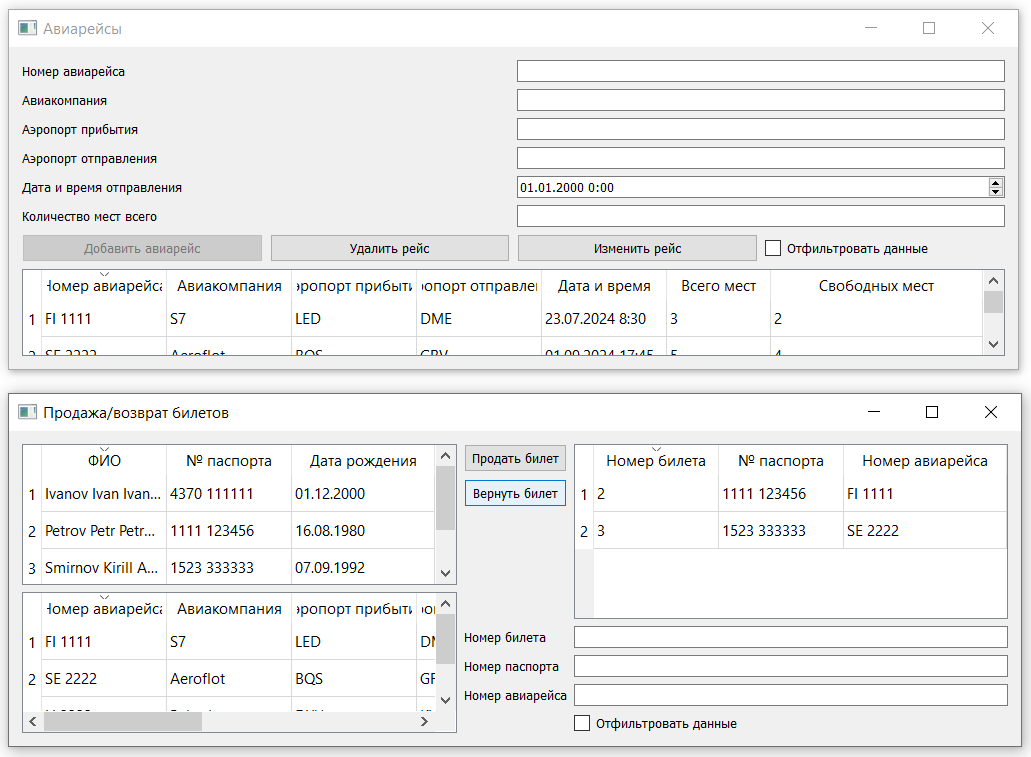


Рисунок 3.3.36 После возврата билета

# Заключение

После выполнения данной работы можно сделать выводы, что были выполнены:

- закрепление знаний, полученных в ходе изучения дисциплины «Объектно-ориентированное программирование»;

- приобретение навыков практического программирования с использованием объектно-ориентированной парадигмы;

- подготовка к выполнению выпускной квалификационной работы. Также следует отметить, что задача курсового проекта - разработка элементов программного обеспечения, включающих в себя иерархии классов в заданной предметной области – выполнена.

Спроектированное приложение работает без сбоев, хорошо выполняя заявленные функции. Преимуществом подобного приложения является оконный интерфейс, который прост в понимании и использовании. Рекомендовалось бы сделать более приятный интерфейс, а также устранить мелкие неполадки.

# Список использованных источников

1. Шумова Е. О. Объектно-ориентированное программирование. Методические указания к выполнению курсового проекта / ФГАОУ ВО СПбГУАП., 2020.
2. Буч Г., Рамбо Д., Якобсон И. Введение в UML от создателей языка – ДМК Пресс, 2015. – 496 с.
3. Ларман К. Применение UML 2.0 и шаблонов проектирования. - 3-е изд. – Вильямс, 2020. – 736 с.

# Приложение 1. Исходный код проекта

Код файла ticketscontroller.h

#pragma once

#include <QObject>

#include <QWeakPointer>

#include "structs.h"

class ModelFlights;

class ModelPassengers;

class ModelTickets;

class TicketsController : public QObject

{

Q\_OBJECT

public:

explicit TicketsController(QWeakPointer<ModelPassengers> passengerModel,

QWeakPointer<ModelTickets> ticketModel,

QWeakPointer<ModelFlights> flightModel,

QObject \*parent = nullptr);

public slots:

void handleShowTickets();

void handleFlightParamsChanged(const QModelIndex &indexFlight,

const Flight &newFlightParams);

void handleRemovePassenger(const QModelIndex &index);

private slots:

void handleAddTicket(const QModelIndex &flightIndex,

const QModelIndex &passengerIndex);

void handleRemoveFlight(const QString &flight);

void handleRemoveTicket(const Ticket &ticket);

private:

QWeakPointer<ModelPassengers> \_passengerModel;

QWeakPointer<ModelTickets> \_ticketModel;

QWeakPointer<ModelFlights> \_flightModel;

};

Код файла flightsortfilterproxymodel.h

#ifndef FLIGHTSORTFILTERPROXYMODEL\_H

#define FLIGHTSORTFILTERPROXYMODEL\_H

#include <QSortFilterProxyModel>

class FlightSortFilterProxyModel : public QSortFilterProxyModel

{

Q\_OBJECT

public:

struct FilterParams

{

QString number;

QString airline;

QString arrival;

QString departure;

QString total\_places;

bool isFilterEnabled;

};

FlightSortFilterProxyModel(QObject \*parent = nullptr): QSortFilterProxyModel(parent){}

public slots:

void handleUpdateFilterParams(const FilterParams &params);

protected:

bool *filterAcceptsRow*(int sourceRow, const QModelIndex &sourceParent) const override;

private:

FilterParams \_currentFilterParams;

};

#endif // FLIGHTSORTFILTERPROXYMODEL\_H

Код файла modelflights.h

#ifndef MODELFLIGHTS\_H

#define MODELFLIGHTS\_H

#include <QAbstractTableModel>

#include <QMap>

#include "structs.h"

class ModelFlights : public QAbstractTableModel

{

Q\_OBJECT

public:

enum class ColumnIndex : int

{

Unknown = -1,

Number,

Airline,

Arrival,

Departure,

Date,

Total\_places,

Free\_places,

count ///< must be last item

};

explicit ModelFlights(QObject \*parent = nullptr);

virtual int *rowCount*(const QModelIndex &parent = QModelIndex()) const override;

virtual int *columnCount*(const QModelIndex &parent = QModelIndex()) const override;

virtual QVariant *data*(const QModelIndex &index, int role = Qt::DisplayRole) const override;

virtual QVariant *headerData*(int section, Qt::Orientation orientation,

int role = Qt::DisplayRole) const override;

Qt::ItemFlags *flags*(const QModelIndex &index) const override;

void addFlight(const Flight &flight);

void removeFlight(const QModelIndex &index);

Flight getFlight(const QModelIndex &index);

void addPassenger(const QString &flightNumber);

void removePassengerFromFlight(const QStringList &flightList);

void changeFlight(const QModelIndex &index, Flight flight);

private:

bool isFlightParamsValid(const Flight &flight);

signals:

void sigRemoveFlight(const QString &number);

private:

QVector<QString /\*number\*/> \_flightIndexes;

QMap<QString /\*number\*/, Flight> \_flightMap;

};

#endif // MODELFLIGHTS\_H

Код файла modelpassengers.h

#ifndef MODELPASSENGERS\_H

#define MODELPASSENGERS\_H

#include <QAbstractTableModel>

#include <QMap>

#include "structs.h"

class ModelPassengers : public QAbstractTableModel

{

Q\_OBJECT

public:

enum class ColumnIndex : int

{

Unknown = -1,

Name,

Passport,

BirthDay,

count ///< must be last item

};

explicit ModelPassengers(QObject \*parent = nullptr);

virtual int *rowCount*(const QModelIndex &parent = QModelIndex()) const override;

virtual int *columnCount*(const QModelIndex &parent = QModelIndex()) const override;

virtual QVariant *data*(const QModelIndex &index, int role = Qt::DisplayRole) const override;

virtual QVariant *headerData*(int section, Qt::Orientation orientation,

int role = Qt::DisplayRole) const override;

Qt::ItemFlags *flags*(const QModelIndex &index) const override;

void addPassenger(const Passenger &passenger);

void removePassenger(const QModelIndex &index);

Passenger getPassenger(const QModelIndex &index);

void changePassenger(const QModelIndex &index, const Passenger &passenger);

signals:

void sigRemovePassenger(const QString &passport);

private:

QVector<QString /\*passport\*/> \_passengerIndexes;

QMap<QString, Passenger> \_passengerMap;

};

#endif // MODELPASSENGERS\_H

Код файла modeltickets.h

#pragma once

#include <QAbstractTableModel>

#include <QMap>

#include "structs.h"

class ModelTickets : public QAbstractTableModel

{

Q\_OBJECT

public:

enum class ColumnIndex : int

{

Unknown = -1,

TicketNumber,

Passport,

Flight,

count ///< must be last item

};

explicit ModelTickets(QObject \*parent = nullptr);

virtual int *rowCount*(const QModelIndex &parent = QModelIndex()) const override;

virtual int *columnCount*(const QModelIndex &parent = QModelIndex()) const override;

virtual QVariant *data*(const QModelIndex &index, int role = Qt::DisplayRole) const override;

virtual QVariant *headerData*(int section, Qt::Orientation orientation,

int role = Qt::DisplayRole) const override;

Qt::ItemFlags *flags*(const QModelIndex &index) const override;

QStringList getFlightsByPassenger(const QString &passenger) const;

bool addTicket(const Ticket &ticket);

void removeTicket(const QModelIndex &index);

void removePassenger(const QString &passenger);

void removeFlight(const QString &flight);

bool isChangedFlightParamsValid(const Flight &newParams,

const Flight &oldFlight,

QString \*error = nullptr);

bool doesPassengerHasTicket(const QString &passengerPassport);

signals:

void sigRemoveTicket(const Ticket &ticket);

private:

QVector<QString /\*ticket\*/> \_ticketIndexes;

QMap<QString /\*ticket\*/, Ticket> \_ticketMap;

QMultiMap<QString /\*flight\*/, QString /\*passenger\*/> \_flightPassengerMap;

int number = 1;

};

Код файла passengersortfilterproxymodel.h

#ifndef PASSENGERSORTFILTERPROXYMODEL\_H

#define PASSENGERSORTFILTERPROXYMODEL\_H

#include <QSortFilterProxyModel>

class PassengerSortFilterProxyModel : public QSortFilterProxyModel

{

Q\_OBJECT

public:

struct FilterParams

{

QString name;

QString passport;

bool isFilterEnabled;

};

explicit PassengerSortFilterProxyModel(QObject \*parent = nullptr);

public slots:

void handleUpdateFilterParams(const FilterParams &params);

protected:

bool *filterAcceptsRow*(int sourceRow, const QModelIndex &sourceParent) const override;

private:

FilterParams \_currentFilterParams;

};

#endif // PASSENGERSORTFILTERPROXYMODEL\_H

Код файла ticketsortfilterproxymodel.h

#ifndef TICKETSORTFILTERPROXYMODEL\_H

#define TICKETSORTFILTERPROXYMODEL\_H

#include <QObject>

#include <QSortFilterProxyModel>

class TicketSortFilterProxyModel: public QSortFilterProxyModel

{

Q\_OBJECT

public:

struct FilterParams

{

QString ticket;

QString passport;

QString flight;

bool isFilterEnabled;

};

explicit TicketSortFilterProxyModel(QObject \*parent = nullptr) : QSortFilterProxyModel(parent) {}

virtual ~*TicketSortFilterProxyModel*(){}

public slots:

void handleUpdateFilterParams(const FilterParams &params);

protected:

bool *filterAcceptsRow*(int sourceRow, const QModelIndex &sourceParent) const override;

private:

FilterParams \_currentFilterParams;

};

#endif // TICKETSORTFILTERPROXYMODEL\_H

Код файла flights.h

#ifndef FLIGHTS\_H

#define FLIGHTS\_H

#include <QWidget>

#include <QAbstractTableModel>

#include "Models/flightsortfilterproxymodel.h"

#include "structs.h"

namespace Ui {

class Flights;

}

class Flights : public QWidget

{

Q\_OBJECT

public:

explicit Flights(QWidget \*parent = 0);

void setFlightsModel(QAbstractTableModel \*model);

~*Flights*();

signals:

void sigUpdateFilterParams(const FlightSortFilterProxyModel::FilterParams &params);

void sigChangeFlight(const QModelIndex &flightIndex, const Flight &newFlightParams);

private:

void initLineEdit();

Flight getFlightParams() const;

void setFlightParams(const Flight &flight);

private slots:

void handleClearFilterParams();

void handleDeleteFlightButtonClicked();

void handleAddFlightButtonClicked();

bool handleAddFlightButtonActivated();

void handleUpdateFilterParams();

void handleChangeFlightButtonClicked();

void handleCurrentSelectedIndexChanged();

private:

Ui::Flights \*ui;

};

#endif // FLIGHTS\_H

Код файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QSharedPointer>

class ModelPassengers;

class ModelFlights;

class ModelTickets;

class TicketsController;

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void handlePassengerButtonClicked();

void handleFlightButtonClicked();

private:

Ui::MainWindow \*ui;

QSharedPointer<ModelPassengers> \_modelPassenger;

QSharedPointer<ModelFlights> \_modelFlight;

QSharedPointer<ModelTickets> \_modelTicket;

QSharedPointer<TicketsController> \_ticController;

};

#endif // MAINWINDOW\_H

Код файла passengers.h

#ifndef PASSENGERS\_H

#define PASSENGERS\_H

#include <QWidget>

#include <QAbstractTableModel>

#include "Models/passengersortfilterproxymodel.h"

#include "structs.h"

namespace Ui {

class Passengers;

}

class Passengers : public QWidget

{

Q\_OBJECT

public:

explicit Passengers(QWidget \*parent = 0);

void setPassengersModel(QAbstractTableModel \*model);

~*Passengers*();

signals:

void sigUpdateFilterParams(const PassengerSortFilterProxyModel::FilterParams &params);

void sigRemovePassenger(const QModelIndex &passangerIndex);

private:

void initLineEdit();

void setPassengerParams(Passenger passenger);

Passenger getPassengerParams() const;

private slots:

void handleDeletePassengerButtonClicked();

void handleAddPassengerButtonClicked();

bool handleAddPassengerButtonActivated();

void handleUpdateFilterParams();

void handleClearFilterParams();

void handleChangePassengerButtonClicked();

void handleCurrentSelectedIndexChanged();

private:

Ui::Passengers \*ui;

};

#endif // PASSENGERS\_H

Код файла structs.h

#ifndef STRUCTS\_H

#define STRUCTS\_H

#include <QString>

#include <QDate>

struct Passenger

{

Passenger() = default;

QString name;

QString passport;

QDate birthday;

};

struct Flight

{

Flight() = default;

QString number;

QString airline;

QString arrival;

QString departure;

QDateTime date;

int total\_places;

int free\_places;

};

struct Ticket

{

Ticket() = default;

QString passport;

QString flight;

QString number;

};

#endif // STRUCTS\_H

Код файла ticketswidget.h

#pragma once

#include <QWidget>

#include <QAbstractTableModel>

#include <QTableView>

#include "Models/ticketsortfilterproxymodel.h"

namespace Ui {

class TicketWidget;

}

class TicketWidget : public QWidget

{

Q\_OBJECT

public:

explicit TicketWidget(QWidget \*parent = 0);

~*TicketWidget*();

void setModels(QAbstractTableModel \*ticketModel,

QAbstractTableModel \*passengerModel,

QAbstractTableModel \*flightModel);

signals:

void sigAddTicket(const QModelIndex& flightIndex,

const QModelIndex& passengerIndex);

void sigUpdateFilterParams(const TicketSortFilterProxyModel::FilterParams &params);

private slots:

void handleAddTicketButtonClicked();

void handleDeleteTicketButtonClicked();

void handleUpdateFilterParams();

private:

void initLineEdit();

QAbstractItemModel \*createProxyModels(QAbstractItemModel\* sourceModel);

void initTableView(QTableView\* tableView);

private:

Ui::TicketWidget \*ui;

};

Код файла ticketscontroller.cpp

#include "ticketscontroller.h"

#include "Models/modelflights.h"

#include "Models/modelpassengers.h"

#include "Models/modeltickets.h"

#include "ticketswidget.h"

#include "structs.h"

#include <QDebug>

TicketsController::TicketsController(QWeakPointer<ModelPassengers> passengerModel,

QWeakPointer<ModelTickets> ticketModel,

QWeakPointer<ModelFlights> flightModel,

QObject \*parent)

: QObject(parent),

\_passengerModel(passengerModel),

\_ticketModel(ticketModel),

\_flightModel(flightModel)

{

if (auto fliModelStrongRef = \_flightModel.toStrongRef())

connect(fliModelStrongRef.data(), &ModelFlights::sigRemoveFlight,

this, &TicketsController::handleRemoveFlight);

if (auto ticModelStrongRef = \_ticketModel.toStrongRef())

connect(ticModelStrongRef.data(), &ModelTickets::sigRemoveTicket,

this, &TicketsController::handleRemoveTicket);

auto flightModRef = \_flightModel.toStrongRef();

auto ticModRef = \_ticketModel.toStrongRef();

{Ticket ticket;

ticket.flight = "FI 1111";

ticket.passport = "4370 111111";

if (ticModRef->addTicket(ticket))

flightModRef->addPassenger(ticket.flight);

}

{Ticket ticket;

ticket.flight = "FI 1111";

ticket.passport = "1111 123456";

if (ticModRef->addTicket(ticket))

flightModRef->addPassenger(ticket.flight);

}

{Ticket ticket;

ticket.flight = "SE 2222";

ticket.passport = "1523 333333";

if (ticModRef->addTicket(ticket))

flightModRef->addPassenger(ticket.flight);

}

}

void TicketsController::handleShowTickets()

{

auto pasModRef = \_passengerModel.toStrongRef();

auto ticModRef = \_ticketModel.toStrongRef();

auto flightModRef = \_flightModel.toStrongRef();

if (!pasModRef || !ticModRef || !flightModRef)

return;

auto ticketWdg = new TicketWidget();

ticketWdg->setModels(ticModRef.data(),

pasModRef.data(),

flightModRef.data());

connect(ticketWdg, &TicketWidget::sigAddTicket,

this, &TicketsController::handleAddTicket);

ticketWdg->setAttribute(Qt::WA\_DeleteOnClose);

ticketWdg->show();

ticketWdg->raise();

ticketWdg->activateWindow();

}

void TicketsController::handleFlightParamsChanged(const QModelIndex &indexFlight,

const Flight &newFlightParams)

{

auto flModel = \_flightModel.toStrongRef();

auto ticModel = \_ticketModel.toStrongRef();

if (!flModel || !ticModel)

return;

auto oldFlightParams = flModel->getFlight(indexFlight);

if (oldFlightParams.number != newFlightParams.number)

{

qDebug() << "Нельзя изменить номер авиарейса";

return;

}

QString errorString;

if (!ticModel->isChangedFlightParamsValid(newFlightParams, oldFlightParams, &errorString))

{

qDebug() << errorString;

return;

}

flModel->changeFlight(indexFlight, newFlightParams);

}

void TicketsController::handleRemovePassenger(const QModelIndex &index)

{

auto ticModRef = \_ticketModel.toStrongRef();

auto flightModRef = \_flightModel.toStrongRef();

auto pasModRef = \_passengerModel.toStrongRef();

if (!ticModRef || !flightModRef)

return;

auto passenger = pasModRef->getPassenger(index);

if (ticModRef->doesPassengerHasTicket(passenger.passport))

{

qDebug() << "Нельзя удалить пассажира, "

"Он зарегистрирован на рейс";

return;

}

pasModRef->removePassenger(index);

}

void TicketsController::handleAddTicket(const QModelIndex& flightIndex,

const QModelIndex& passengerIndex)

{

auto pasModRef = \_passengerModel.toStrongRef();

auto ticModRef = \_ticketModel.toStrongRef();

auto flightModRef = \_flightModel.toStrongRef();

if (!pasModRef || !ticModRef || !flightModRef)

return;

auto passenger = pasModRef->getPassenger(passengerIndex);

auto flight = flightModRef->getFlight(flightIndex);

if (flight.free\_places == 0)

{

qDebug() << "Невозможно добавить пассажира на рейс, "

"нет свободных мест";

return;

}

Ticket ticket;

ticket.flight = flight.number;

ticket.passport = passenger.passport;

if (ticModRef->addTicket(ticket))

flightModRef->addPassenger(flight.number);

}

void TicketsController::handleRemoveFlight(const QString &flight)

{

auto ticModRef = \_ticketModel.toStrongRef();

if (!ticModRef)

return;

ticModRef->removeFlight(flight);

}

void TicketsController::handleRemoveTicket(const Ticket &ticket)

{

auto flightModRef = \_flightModel.toStrongRef();

if (!flightModRef)

return;

QStringList flightList;

flightList << ticket.flight;

flightModRef->removePassengerFromFlight(flightList);

}

Код файла flightsortfilterproxymodel.cpp

#include "flightsortfilterproxymodel.h"

#include "modelflights.h"

void FlightSortFilterProxyModel::handleUpdateFilterParams(const FlightSortFilterProxyModel::FilterParams &params)

{

\_currentFilterParams = params;

invalidateFilter();

}

bool FlightSortFilterProxyModel::*filterAcceptsRow*(int sourceRow, const QModelIndex &sourceParent) const

{

if (!sourceModel())

return false;

if (!\_currentFilterParams.isFilterEnabled)

return true;

QModelIndex numberIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelFlights::ColumnIndex::Number),

sourceParent);

QModelIndex airlineIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelFlights::ColumnIndex::Airline),

sourceParent);

QModelIndex arrivalIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelFlights::ColumnIndex::Arrival),

sourceParent);

QModelIndex departureIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelFlights::ColumnIndex::Departure),

sourceParent);

bool numberChecked = sourceModel()->*data*(numberIndex).toString().contains(\_currentFilterParams.number,

Qt::CaseInsensitive);

bool airlineChecked = sourceModel()->*data*(airlineIndex).toString().contains(\_currentFilterParams.airline,

Qt::CaseInsensitive);

bool arrivalChecked = sourceModel()->*data*(arrivalIndex).toString().contains(\_currentFilterParams.arrival,

Qt::CaseInsensitive);

bool departureChecked = sourceModel()->*data*(departureIndex).toString().contains(\_currentFilterParams.departure,

Qt::CaseInsensitive);

return numberChecked && airlineChecked && arrivalChecked && departureChecked;

}

Код файла modelflights.cpp

#include "modelflights.h"

#include <QDebug>

ModelFlights::ModelFlights(QObject \*parent) : QAbstractTableModel(parent)

{

\_flightMap.insert("FI 1111",{

"FI 1111",

"S7",

"LED",

"DME",

QDateTime(QDate(2024, 7, 23), QTime(8, 30, 0)),

3,

3

});

\_flightIndexes.append("FI 1111");

\_flightMap.insert("SE 2222",{

"SE 2222",

"Aeroflot",

"BQS",

"GRV",

QDateTime(QDate(2024, 9, 1), QTime(17, 45, 0)),

5,

5

});

\_flightIndexes.append("SE 2222");

\_flightMap.insert("LI 3333",{

"LI 3333",

"Pobeda",

"DYU",

"KVX",

QDateTime(QDate(2024, 11, 15), QTime(2, 10, 0)),

4,

4

});

\_flightIndexes.append("LI 3333");

}

int ModelFlights::*rowCount*(const QModelIndex &parent) const

{

Q\_UNUSED(parent);

return \_flightIndexes.size();

}

int ModelFlights::*columnCount*(const QModelIndex &parent) const

{

Q\_UNUSED(parent);

return static\_cast<int>(ColumnIndex::count);

}

QVariant ModelFlights::*data*(const QModelIndex &index, int role) const

{

if (!index.isValid())

return {};

if (role != Qt::DisplayRole)

return {};

auto flight = \_flightMap.value(\_flightIndexes.at(index.row()));

switch (static\_cast<ColumnIndex>(index.column()))

{

case ColumnIndex::Number: return flight.number;

case ColumnIndex::Airline: return flight.airline;

case ColumnIndex::Arrival: return flight.arrival;

case ColumnIndex::Departure: return flight.departure;

case ColumnIndex::Date: return flight.date;

case ColumnIndex::Total\_places: return flight.total\_places;

case ColumnIndex::Free\_places: return flight.free\_places;

default:

break;

}

return {};

}

QVariant ModelFlights::*headerData*(int section, Qt::Orientation orientation, int role) const

{

if (role != Qt::DisplayRole)

return {};

if (orientation == Qt::Orientation::Vertical)

return section + 1;

switch (static\_cast<ColumnIndex>(section))

{

case ColumnIndex::Number: return QString("Номер авиарейса");

case ColumnIndex::Airline: return QString("Авиакомпания");

case ColumnIndex::Arrival: return QString("Аэропорт прибытия");

case ColumnIndex::Departure: return QString("Аэропорт отправления");

case ColumnIndex::Date: return QString("Дата и время");

case ColumnIndex::Total\_places: return QString("Всего мест");

case ColumnIndex::Free\_places: return QString("Свободных мест");

default:

return {};

}

}

Qt::ItemFlags ModelFlights::*flags*(const QModelIndex &index) const

{

Q\_UNUSED(index);

return Qt::ItemFlag::ItemIsEnabled | Qt::ItemFlag::ItemIsSelectable;

}

void ModelFlights::addFlight(const Flight &flight)

{

if (!isFlightParamsValid(flight))

return;

beginInsertRows({}, \_flightIndexes.size(), \_flightIndexes.size());

\_flightMap.insert(flight.number, flight);

\_flightIndexes.append(flight.number);

endInsertRows();

}

void ModelFlights::removeFlight(const QModelIndex &index)

{

if (!index.isValid())

return;

beginRemoveRows({}, index.row(), index.row());

\_flightMap.remove(\_flightIndexes.at(index.row()));

auto number = \_flightIndexes.takeAt(index.row());

endRemoveRows();

emit sigRemoveFlight(number);

}

Flight ModelFlights::getFlight(const QModelIndex &index)

{

return \_flightMap.value(\_flightIndexes.at(index.row()));

}

void ModelFlights::addPassenger(const QString &flightNumber)

{

\_flightMap[flightNumber].free\_places--;

emit dataChanged(createIndex(\_flightIndexes.indexOf(flightNumber),

static\_cast<int>(ColumnIndex::Free\_places)),

createIndex(\_flightIndexes.indexOf(flightNumber),

static\_cast<int>(ColumnIndex::Free\_places)));

}

void ModelFlights::removePassengerFromFlight(const QStringList &flightList)

{

for (const auto &flight : flightList)

{

\_flightMap[flight].free\_places++;

emit dataChanged(createIndex(\_flightIndexes.indexOf(flight),

static\_cast<int>(ColumnIndex::Free\_places)),

createIndex(\_flightIndexes.indexOf(flight),

static\_cast<int>(ColumnIndex::Free\_places)));

}

}

void ModelFlights::changeFlight(const QModelIndex &index, Flight flight)

{

if (\_flightMap.contains(flight.number) &&

\_flightIndexes.at(index.row()) != flight.number)

{

qDebug() << QString("Авиакомпания %1 уже осуществляет данный перелёт %2 - %3 ,"

"авиарейсы должны быть уникальны").

arg(flight.airline, flight.departure,flight.arrival);

return;

}

else if (flight.arrival == flight.departure)

{

qDebug() << QString("Аэропорты прибытия и отправления должны различаться");

return;

}

else if (flight.date.secsTo(QDateTime::currentDateTime()) > 0)

{

qDebug() << QString("Время рейса не может быть в прошлом");

return;

}

auto oldFlight = \_flightMap.value(\_flightIndexes.at(index.row()));

flight.free\_places = flight.total\_places - (oldFlight.total\_places

- oldFlight.free\_places);

\_flightMap.insert(\_flightIndexes.at(index.row()), flight);

emit dataChanged(createIndex(index.row(),

static\_cast<int>(ColumnIndex::Number)),

createIndex(index.row(),

static\_cast<int>(ColumnIndex::Free\_places)));

}

bool ModelFlights::isFlightParamsValid(const Flight &flight)

{

if (\_flightMap.contains(flight.number))

{

qDebug() << QString("Авиакомпания %1 уже осуществляет данный перелёт %2 - %3,"

"авиарейсы должны быть уникальны").

arg(flight.airline, flight.departure,flight.arrival);

return false;

}

else if (flight.arrival == flight.departure)

{

qDebug() << QString("Аэропорты прибытия и отправления долsжны различаться");

return false;

}

else if (flight.date.secsTo(QDateTime::currentDateTime()) > 0)

{

qDebug() << QString("Время рейса не может быть в прошлом");

return false;

}

return true;

}

Код файла modelpassengers.cpp

#include "modelpassengers.h"

#include <QDebug>

ModelPassengers::ModelPassengers(QObject \*parent) : QAbstractTableModel(parent)

{

\_passengerMap.insert("4370 111111",{

"Ivanov Ivan Ivanovich",

"4370 111111",

QDate (2000,12,01)

});

\_passengerIndexes.append("4370 111111");

\_passengerMap.insert("1111 123456",{

"Petrov Petr Petrovich",

"1111 123456",

QDate (1980,8,16)

});

\_passengerIndexes.append("1111 123456");

\_passengerMap.insert("1523 333333",{

"Smirnov Kirill Antonovich",

"1523 333333",

QDate (1992,9,07)

});

\_passengerIndexes.append("1523 333333");

}

int ModelPassengers::*rowCount*(const QModelIndex &parent) const

{

Q\_UNUSED(parent);

return \_passengerIndexes.size();

}

int ModelPassengers::*columnCount*(const QModelIndex &parent) const

{

Q\_UNUSED(parent);

return static\_cast<int>(ColumnIndex::count);

}

QVariant ModelPassengers::*data*(const QModelIndex &index, int role) const

{

if (!index.isValid())

return {};

if (role != Qt::DisplayRole)

return {};

auto passenger = \_passengerMap.value(\_passengerIndexes.at(index.row()));

switch (static\_cast<ColumnIndex>(index.column()))

{

case ColumnIndex::Name: return passenger.name;

case ColumnIndex::Passport: return passenger.passport;

case ColumnIndex::BirthDay: return passenger.birthday;

default:

break;

}

return {};

}

QVariant ModelPassengers::*headerData*(int section, Qt::Orientation orientation, int role) const

{

if (role != Qt::DisplayRole)

return {};

if (orientation == Qt::Orientation::Vertical)

return section + 1;

switch (static\_cast<ColumnIndex>(section))

{

case ColumnIndex::Name: return QString("ФИО");

case ColumnIndex::Passport: return QString("№ паспорта");

case ColumnIndex::BirthDay: return QString("Дата рождения");

default:

return {};

}

}

Qt::ItemFlags ModelPassengers::*flags*(const QModelIndex &index) const

{

Q\_UNUSED(index);

return Qt::ItemFlag::ItemIsEnabled | Qt::ItemFlag::ItemIsSelectable;

}

void ModelPassengers::addPassenger(const Passenger &passenger)

{

if (\_passengerMap.contains(passenger.passport))

{

qDebug() << QString("Пассажир %1 с паспортом %2 присутствует, "

"номер паспорта должен быть уникален").

arg(passenger.name, passenger.passport);

return;

}

beginInsertRows({}, \_passengerIndexes.size(), \_passengerIndexes.size());

\_passengerMap.insert(passenger.passport, passenger);

\_passengerIndexes.append(passenger.passport);

endInsertRows();

}

void ModelPassengers::removePassenger(const QModelIndex &index)

{

if (!index.isValid())

return;

beginRemoveRows({}, index.row(), index.row());

\_passengerMap.remove(\_passengerIndexes.at(index.row()));

\_passengerIndexes.remove(index.row());

endRemoveRows();

}

Passenger ModelPassengers::getPassenger(const QModelIndex &index)

{

return \_passengerMap.value(\_passengerIndexes.at(index.row()));

}

void ModelPassengers::changePassenger(const QModelIndex &index, const Passenger &passenger)

{

if (\_passengerIndexes.at(index.row()) != passenger.passport)

{

qDebug() << QString("Паспорт пассажира изменить нельзя");

return;

}

\_passengerMap.insert(passenger.passport, passenger);

emit dataChanged(createIndex(\_passengerIndexes.indexOf(passenger.passport),

static\_cast<int>(ColumnIndex::Unknown) + 1),

createIndex(\_passengerIndexes.indexOf(passenger.passport),

static\_cast<int>(ColumnIndex::count) - 1));

}

Код файла modeltickets.cpp

#include "modeltickets.h"

#include <QDebug>

ModelTickets::ModelTickets(QObject \*parent) : QAbstractTableModel(parent)

{

}

int ModelTickets::*rowCount*(const QModelIndex &parent) const

{

Q\_UNUSED(parent);

return \_ticketIndexes.size();

}

int ModelTickets::*columnCount*(const QModelIndex &parent) const

{

Q\_UNUSED(parent);

return static\_cast<int>(ColumnIndex::count);

}

QVariant ModelTickets::*data*(const QModelIndex &index, int role) const

{

if (!index.isValid())

return {};

if (role != Qt::DisplayRole)

return {};

auto ticket = \_ticketMap.value(\_ticketIndexes.at(index.row()));

switch (static\_cast<ColumnIndex>(index.column()))

{

case ColumnIndex::Flight: return ticket.flight;

case ColumnIndex::Passport: return ticket.passport;

case ColumnIndex::TicketNumber: return ticket.number;

default:

break;

}

return {};

}

QVariant ModelTickets::*headerData*(int section, Qt::Orientation orientation, int role) const

{

if (role != Qt::DisplayRole)

return {};

if (orientation == Qt::Orientation::Vertical)

return section + 1;

switch (static\_cast<ColumnIndex>(section))

{

case ColumnIndex::Flight: return QString("Номер авиарейса");

case ColumnIndex::Passport: return QString("№ паспорта");

case ColumnIndex::TicketNumber: return QString("Номер билета");

default:

return {};

}

}

Qt::ItemFlags ModelTickets::*flags*(const QModelIndex &index) const

{

Q\_UNUSED(index);

return Qt::ItemFlag::ItemIsEnabled | Qt::ItemFlag::ItemIsSelectable;

}

QStringList ModelTickets::getFlightsByPassenger(const QString &passenger) const

{

QStringList flightList;

for (auto it = \_flightPassengerMap.cbegin();

it != \_flightPassengerMap.cend();

++it)

if (it.value() == passenger)

flightList.append(it.key());

return flightList;

}

bool ModelTickets::addTicket(const Ticket &ticket)

{

if (\_flightPassengerMap.contains(ticket.flight))

{

for(const auto &passport : \_flightPassengerMap.values(ticket.flight))

{

if (ticket.passport == passport)

{

qDebug() << QString("Пассажир с паспортом %1, "

"зарегистрирован на рейс %2").

arg(passport, ticket.flight);

return false;

}

}

}

Ticket ticketModel = ticket;

ticketModel.number = QString::number(number++);

beginInsertRows({}, \_ticketIndexes.size(), \_ticketIndexes.size());

\_ticketMap.insert(ticketModel.number, ticketModel);

\_ticketIndexes.append(ticketModel.number);

\_flightPassengerMap.insert(ticket.flight, ticket.passport);

endInsertRows();

return true;

}

void ModelTickets::removeTicket(const QModelIndex &index)

{

if (!index.isValid())

return;

beginRemoveRows({}, index.row(), index.row());

auto ticket = \_ticketMap.take(\_ticketIndexes.at(index.row()));

\_ticketIndexes.remove(index.row());

\_flightPassengerMap.remove(ticket.flight, ticket.passport);

endRemoveRows();

emit sigRemoveTicket(ticket);

}

void ModelTickets::removePassenger(const QString &passenger)

{

for (auto it = \_ticketMap.begin();

it != \_ticketMap.end();

++it)

{

if (it.value().passport == passenger)

{

beginRemoveRows({},\_ticketIndexes.indexOf(it.value().number),

\_ticketIndexes.indexOf(it.value().number));

\_ticketIndexes.removeAt(\_ticketIndexes.indexOf(it.value().number));

\_ticketMap.erase(it);

auto iter =\_flightPassengerMap.find(it.value().flight, it.value().passport);

if (iter != \_flightPassengerMap.end())

\_flightPassengerMap.erase(iter);

endRemoveRows();

}

}

}

void ModelTickets::removeFlight(const QString &flight)

{

for (auto it = \_ticketMap.begin();

it != \_ticketMap.end();

++it)

{

if (it.value().flight == flight)

{

beginRemoveRows({},\_ticketIndexes.indexOf(it.value().number),

\_ticketIndexes.indexOf(it.value().number));

\_ticketIndexes.removeAt(\_ticketIndexes.indexOf(it.value().number));

\_ticketMap.erase(it);

endRemoveRows();

}

}

\_flightPassengerMap.remove(flight);

}

bool ModelTickets::isChangedFlightParamsValid(const Flight &newParams, const Flight &oldFlight, QString \*error)

{

if(\_flightPassengerMap.values(oldFlight.number).size() >

newParams.total\_places)

{

if (error)

\*error = "Количество мест меньше проданных билетов";

return false;

}

return true;

}

bool ModelTickets::doesPassengerHasTicket(const QString &passengerPassport)

{

return \_flightPassengerMap.values().contains(passengerPassport);

}

Код файла passengersortfilterproxymodel.cpp

#include "passengersortfilterproxymodel.h"

#include "modelpassengers.h"

PassengerSortFilterProxyModel::PassengerSortFilterProxyModel(QObject \*parent)

: QSortFilterProxyModel(parent)

{

}

void PassengerSortFilterProxyModel::handleUpdateFilterParams(const PassengerSortFilterProxyModel::FilterParams &params)

{

\_currentFilterParams = params;

invalidateFilter();

}

bool PassengerSortFilterProxyModel::*filterAcceptsRow*(int sourceRow, const QModelIndex &sourceParent) const

{

if (!sourceModel())

return false;

if (!\_currentFilterParams.isFilterEnabled)

return true;

QModelIndex nameIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelPassengers::ColumnIndex::Name),

sourceParent);

QModelIndex passportIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelPassengers::ColumnIndex::Passport),

sourceParent);

bool nameChecked = sourceModel()->*data*(nameIndex).toString().contains(\_currentFilterParams.name,

Qt::CaseInsensitive);

bool passportChecked = sourceModel()->*data*(passportIndex).toString().contains(\_currentFilterParams.passport,

Qt::CaseInsensitive);

return nameChecked && passportChecked;

}

Код файла ticketsortfilterproxymodel.cpp

#include "ticketsortfilterproxymodel.h"

#include "modeltickets.h"

void TicketSortFilterProxyModel::handleUpdateFilterParams(const TicketSortFilterProxyModel::FilterParams &params)

{

\_currentFilterParams = params;

invalidateFilter();

}

bool TicketSortFilterProxyModel::*filterAcceptsRow*(int sourceRow, const QModelIndex &sourceParent) const

{

if (!sourceModel())

return false;

if (!\_currentFilterParams.isFilterEnabled)

return true;

QModelIndex ticketIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelTickets::ColumnIndex::TicketNumber),

sourceParent);

QModelIndex passportIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelTickets::ColumnIndex::Passport),

sourceParent);

QModelIndex flightIndex = sourceModel()->*index*(sourceRow,

static\_cast<int>(ModelTickets::ColumnIndex::Flight),

sourceParent);

bool ticketChecked = sourceModel()->*data*(ticketIndex).toString().contains(\_currentFilterParams.ticket,

Qt::CaseInsensitive);

bool passportChecked = sourceModel()->*data*(passportIndex).toString().contains(\_currentFilterParams.passport,

Qt::CaseInsensitive);

bool flightChecked = sourceModel()->*data*(flightIndex).toString().contains(\_currentFilterParams.flight,

Qt::CaseInsensitive);

return ticketChecked && passportChecked && flightChecked;

}

Код файла flights.cpp

#include <QSortFilterProxyModel>

#include "flights.h"

#include "ui\_flights.h"

#include <QHeaderView>

#include <Models/modelflights.h>

#include <QDebug>

Flights::Flights(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Flights)

{

ui->setupUi(this);

ui->pushButton\_add->setEnabled(false);

initLineEdit();

connect(ui->pushButton\_add, &QPushButton::clicked,

this, &Flights::handleAddFlightButtonClicked);

connect(ui->pushButton\_change, &QPushButton::clicked,

this, &Flights::handleChangeFlightButtonClicked);

connect(ui->lineEdit\_number, &QLineEdit::textChanged,

this, &Flights::handleAddFlightButtonActivated);

connect(ui->lineEdit\_airline, &QLineEdit::textChanged,

this, &Flights::handleAddFlightButtonActivated);

connect(ui->lineEdit\_arrival, &QLineEdit::textChanged,

this, &Flights::handleAddFlightButtonActivated);

connect(ui->lineEdit\_departure, &QLineEdit::textChanged,

this, &Flights::handleAddFlightButtonActivated);

connect(ui->lineEdit\_total\_places, &QLineEdit::textChanged,

this, &Flights::handleAddFlightButtonActivated);

connect(ui->lineEdit\_number, &QLineEdit::textChanged,

this, &Flights::handleUpdateFilterParams);

connect(ui->lineEdit\_airline, &QLineEdit::textChanged,

this, &Flights::handleUpdateFilterParams);

connect(ui->lineEdit\_arrival, &QLineEdit::textChanged,

this, &Flights::handleUpdateFilterParams);

connect(ui->lineEdit\_departure, &QLineEdit::textChanged,

this, &Flights::handleUpdateFilterParams);

connect(ui->lineEdit\_total\_places, &QLineEdit::textChanged,

this, &Flights::handleUpdateFilterParams);

connect(ui->tableView, &QTableView::clicked,

this, &Flights::handleCurrentSelectedIndexChanged);

connect (ui->checkBox\_search, &QCheckBox::clicked,

this, [this](bool isChecked){

ui->pushButton\_add->setEnabled(!isChecked && handleAddFlightButtonActivated());

});

connect (ui->checkBox\_search, &QCheckBox::clicked,

this, &Flights::handleUpdateFilterParams);

ui->tableView->horizontalHeader()->setStretchLastSection(true);

ui->tableView->setSortingEnabled(true);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

connect(ui->pushButton\_delete, &QPushButton::clicked,

this, &Flights::handleDeleteFlightButtonClicked);

}

void Flights::handleClearFilterParams()

{

ui->lineEdit\_number->clear();

ui->lineEdit\_airline->clear();

ui->lineEdit\_arrival->clear();

ui->lineEdit\_departure->clear();

ui->lineEdit\_total\_places->clear();

ui->dateTimeEdit->setDateTime(QDateTime(QDate(2001, 01, 01), QTime(00, 00, 00)));

}

void Flights::setFlightsModel(QAbstractTableModel \*model)

{

auto proxyModel = new FlightSortFilterProxyModel();

proxyModel->*setSourceModel*(model);

connect(this, &Flights::sigUpdateFilterParams,

proxyModel, &FlightSortFilterProxyModel::handleUpdateFilterParams);

ui->tableView->*setModel*(proxyModel);

}

void Flights::handleAddFlightButtonClicked()

{

auto proxyModel = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelFlights\*>(proxyModel->sourceModel());

if (!model)

return;

auto flight = getFlightParams();

model->addFlight(flight);

handleClearFilterParams();

}

bool Flights::handleAddFlightButtonActivated()

{

bool result = ui->lineEdit\_airline->text() != "" &&

ui->lineEdit\_arrival->text() != ""&&

ui->lineEdit\_departure->text()!=""&&

ui->lineEdit\_total\_places->text() != "";

if (result)

ui->pushButton\_add->setEnabled(true);

else

ui->pushButton\_add->setEnabled(false);

return result;

}

void Flights::handleUpdateFilterParams()

{

FlightSortFilterProxyModel::FilterParams params;

params.number = ui->lineEdit\_number->text();

params.airline = ui->lineEdit\_airline->text();

params.arrival = ui->lineEdit\_arrival->text();

params.departure = ui->lineEdit\_departure->text();

params.total\_places = ui->lineEdit\_total\_places->text();

params.isFilterEnabled = ui->checkBox\_search->isChecked();

emit sigUpdateFilterParams(params);

}

Flights::~*Flights*()

{

delete ui;

}

void Flights::initLineEdit()

{

{

QRegExp regExp("[а-яА-Яa-Az-Z"" ""-""]+");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_airline->setValidator(validator);

ui->lineEdit\_arrival->setValidator(validator);

ui->lineEdit\_departure->setValidator(validator);

}

{

QRegExp regExp("[A-Z][A-Z][ ]\\d{4}");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_number->setValidator(validator);

}

{

QRegExp regExp("(?!0\*$)\\d{2}");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_total\_places->setValidator(validator);

}

}

Flight Flights::getFlightParams() const

{

Flight flight;

flight.number = ui->lineEdit\_number->text();

flight.airline = ui->lineEdit\_airline->text();

flight.arrival = ui->lineEdit\_arrival->text();

flight.departure = ui->lineEdit\_departure->text();

flight.date = ui->dateTimeEdit->dateTime();

flight.total\_places = ui->lineEdit\_total\_places->text().toInt();

flight.free\_places = flight.total\_places;

return flight;

}

void Flights::setFlightParams(const Flight &flight)

{

ui->lineEdit\_number->setText(flight.number);

ui->lineEdit\_airline->setText(flight.airline);

ui->lineEdit\_arrival->setText(flight.arrival);

ui->lineEdit\_departure->setText(flight.departure);

ui->dateTimeEdit->setDateTime(flight.date);

ui->lineEdit\_total\_places->setText(QString::number(flight.total\_places));

}

void Flights::handleDeleteFlightButtonClicked()

{

QItemSelectionModel \*selectFlights = ui->tableView->selectionModel();

auto selectedRowsFlights = selectFlights->selectedRows();

if (selectedRowsFlights.size() != 1)

{

qDebug() << "Выберите 1 рейс для удаления";

return;

}

auto proxyModelFli = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelFlights\*>(proxyModelFli->sourceModel());

if (!model)

return;

model->removeFlight(proxyModelFli->*mapToSource*(selectedRowsFlights.first()));

handleClearFilterParams();

}

void Flights::handleChangeFlightButtonClicked()

{

QItemSelectionModel \*selectFlights = ui->tableView->selectionModel();

auto selectedRowsFlights = selectFlights->selectedRows();

if (selectedRowsFlights.size() != 1)

{

qDebug() << "Выберите 1 рейс для редактирования";

return;

}

auto proxyModelFli = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

emit sigChangeFlight(proxyModelFli->*mapToSource*(selectedRowsFlights.first()),

getFlightParams());

handleClearFilterParams();

}

void Flights::handleCurrentSelectedIndexChanged()

{

QItemSelectionModel \*selectFlights = ui->tableView->selectionModel();

auto selectedRowsFlights = selectFlights->selectedRows();

if (selectedRowsFlights.size() != 1)

return;

auto proxyModelFlight = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelFlights\*>(proxyModelFlight->sourceModel());

if (!model)

return;

setFlightParams(model->getFlight(proxyModelFlight->*mapToSource*(selectedRowsFlights.first())));

}

Код файла main.cpp

#include "mainwindow.h"

#include <QApplication>

#include <QTextCodec>

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

{

QTextCodec::setCodecForLocale(QTextCodec::codecForName("Windows-1251"));

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Код файла mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "passengers.h"

#include "flights.h"

#include "Models/modelpassengers.h"

#include "Models/modelflights.h"

#include "Models/modeltickets.h"

#include "Controllers/ticketscontroller.h"

#include <QPushButton>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

\_modelPassenger = QSharedPointer<ModelPassengers>::create();

\_modelFlight = QSharedPointer<ModelFlights>::create();

\_modelTicket = QSharedPointer<ModelTickets>::create();

\_ticController = QSharedPointer<TicketsController>::create(\_modelPassenger.toWeakRef(),

\_modelTicket.toWeakRef(),

\_modelFlight.toWeakRef());

connect(ui->pushButton\_passenger, &QPushButton::clicked,

this, &MainWindow::handlePassengerButtonClicked);

connect(ui->pushButton\_flight, &QPushButton::clicked,

this, &MainWindow::handleFlightButtonClicked);

connect(ui->pushButton\_ticket, &QPushButton::clicked,

\_ticController.data(), &TicketsController::handleShowTickets);

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::handlePassengerButtonClicked()

{

auto passengerWdg = new Passengers();

connect(passengerWdg, &Passengers::sigRemovePassenger,

\_ticController.data(), &TicketsController::handleRemovePassenger);

passengerWdg->setPassengersModel(\_modelPassenger.data());

passengerWdg->setAttribute(Qt::WA\_DeleteOnClose);

passengerWdg->show();

passengerWdg->raise();

passengerWdg->activateWindow();

}

void MainWindow::handleFlightButtonClicked()

{

auto flightWdg = new Flights();

connect(flightWdg, &Flights::sigChangeFlight,

\_ticController.data(), &TicketsController::handleFlightParamsChanged);

flightWdg->setFlightsModel(\_modelFlight.data());

flightWdg->setAttribute(Qt::WA\_DeleteOnClose);

flightWdg->show();

flightWdg->raise();

flightWdg->activateWindow();

}

Код файла passengers.cpp

#include "passengers.h"

#include "ui\_passengers.h"

#include <QHeaderView>

#include <Models/modelpassengers.h>

#include <QRegExpValidator>

#include <QLineEdit>

#include <QValidator>

#include <QSortFilterProxyModel>

#include <QDebug>

Passengers::Passengers(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Passengers)

{

ui->setupUi(this);

ui->pushButton\_add->setEnabled(false);

initLineEdit();

connect(ui->lineEdit\_name, &QLineEdit::textChanged,

this, &Passengers::handleAddPassengerButtonActivated);

connect(ui->lineEdit\_number, &QLineEdit::textChanged,

this, &Passengers::handleAddPassengerButtonActivated);

connect(ui->lineEdit\_name, &QLineEdit::textChanged,

this, &Passengers::handleUpdateFilterParams);

connect(ui->lineEdit\_number, &QLineEdit::textChanged,

this, &Passengers::handleUpdateFilterParams);

connect(ui->pushButton\_add, &QPushButton::clicked,

this, &Passengers::handleAddPassengerButtonClicked);

connect(ui->pushButton\_change, &QPushButton::clicked,

this, &Passengers::handleChangePassengerButtonClicked);

connect(ui->tableView, &QTableView::clicked,

this, &Passengers::handleCurrentSelectedIndexChanged);

connect (ui->checkBox\_search, &QCheckBox::clicked,

this, [this](bool isChecked){

ui->pushButton\_add->setEnabled(!isChecked && handleAddPassengerButtonActivated());

});

connect (ui->checkBox\_search, &QCheckBox::clicked,

this, &Passengers::handleUpdateFilterParams);

ui->tableView->horizontalHeader()->setStretchLastSection(true);

ui->tableView->setSortingEnabled(true);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

connect(ui->pushButton\_delete, &QPushButton::clicked,

this, &Passengers::handleDeletePassengerButtonClicked);

}

void Passengers::handleClearFilterParams()

{

ui->lineEdit\_name->clear();

ui->lineEdit\_number->clear();

ui->dateEdit\_date->setDate(QDate(2001, 01, 01));

}

void Passengers::setPassengersModel(QAbstractTableModel \*model)

{

auto proxyModel = new PassengerSortFilterProxyModel();

proxyModel->*setSourceModel*(model);

connect(this, &Passengers::sigUpdateFilterParams,

proxyModel, &PassengerSortFilterProxyModel::handleUpdateFilterParams);

ui->tableView->*setModel*(proxyModel);

}

void Passengers::handleAddPassengerButtonClicked()

{

auto proxyModel = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelPassengers\*>(proxyModel->sourceModel());

if (!model)

return;

auto passenger = getPassengerParams();

if (passenger.birthday.daysTo(QDate(1890,01,01)) > 0 ||

passenger.birthday.daysTo(QDate(2023,11,22)) < 0)

{

qDebug() << "Error: Дата рождения не принадлежит отрезку "

"от 01.01.1890 до 20.11.2023";

return;

}

model->addPassenger(passenger);

handleClearFilterParams();

}

bool Passengers::handleAddPassengerButtonActivated()

{

bool result = ui->lineEdit\_name->text() != "" &&

ui->lineEdit\_number->text().length() == 11;

if (result)

ui->pushButton\_add->setEnabled(true);

else

ui->pushButton\_add->setEnabled(false);

return result;

}

void Passengers::handleUpdateFilterParams()

{

PassengerSortFilterProxyModel::FilterParams params;

params.name = ui->lineEdit\_name->text();

params.passport = ui->lineEdit\_number->text();

params.isFilterEnabled = ui->checkBox\_search->isChecked();

emit sigUpdateFilterParams(params);

}

Passengers::~*Passengers*()

{

delete ui;

}

void Passengers::initLineEdit()

{

{

QRegExp regExp("[а-яА-Яa-Az-Z"" ""-""]+");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_name->setValidator(validator);

}

{

QRegExp regExp("\\d{4}[ ]\\d{6}");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_number->setValidator(validator);

}

}

void Passengers::setPassengerParams(Passenger passenger)

{

ui->lineEdit\_name->setText(passenger.name);

ui->lineEdit\_number->setText(passenger.passport);

ui->dateEdit\_date->setDate(passenger.birthday);

}

Passenger Passengers::getPassengerParams() const

{

Passenger passenger;

passenger.birthday = ui->dateEdit\_date->date();

passenger.name = ui->lineEdit\_name->text();

passenger.passport = ui->lineEdit\_number->text();

return passenger;

}

void Passengers::handleDeletePassengerButtonClicked()

{

QItemSelectionModel \*selectPassengers = ui->tableView->selectionModel();

auto selectedRowsPassengers = selectPassengers->selectedRows();

if (selectedRowsPassengers.size() != 1)

{

qDebug() << "Выберите 1 пассажира для удаления";

return;

}

auto proxyModelPas = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelPassengers\*>(proxyModelPas->sourceModel());

if (!model)

return;

emit sigRemovePassenger(proxyModelPas->*mapToSource*(selectedRowsPassengers.first()));

handleClearFilterParams();

}

void Passengers::handleChangePassengerButtonClicked()

{

QItemSelectionModel \*selectPassengers = ui->tableView->selectionModel();

auto selectedRowsPassengers = selectPassengers->selectedRows();

if (selectedRowsPassengers.size() != 1)

{

qDebug() << "Выберите 1 пассажира для редактирования";

return;

}

auto proxyModel = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelPassengers\*>(proxyModel->sourceModel());

if (!model)

return;

auto passenger = getPassengerParams();

model->changePassenger(selectedRowsPassengers.first(), passenger);

handleClearFilterParams();

}

void Passengers::handleCurrentSelectedIndexChanged()

{

QItemSelectionModel \*selectFlights = ui->tableView->selectionModel();

auto selectedRows = selectFlights->selectedRows();

if (selectedRows.size() != 1)

return;

auto proxyModelPassenger = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView->model());

auto model = dynamic\_cast<ModelPassengers\*>(proxyModelPassenger->sourceModel());

if (!model)

return;

setPassengerParams(model->getPassenger(proxyModelPassenger->*mapToSource*(selectedRows.first())));

}

Код файла structs.cpp

#include "structs.h"

Код файла ticketswidget.cpp

#include "ticketswidget.h"

#include "ui\_tickets.h"

#include <QHeaderView>

#include <Models/modeltickets.h>

#include <QSortFilterProxyModel>

#include "Models/ticketsortfilterproxymodel.h"

#include <QTableView>

#include <QDebug>

TicketWidget::TicketWidget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::TicketWidget)

{

ui->setupUi(this);

initLineEdit();

connect(ui->pushButton\_sell, &QPushButton::clicked,

this, &TicketWidget::handleAddTicketButtonClicked);

connect(ui->pushButton\_return, &QPushButton::clicked,

this, &TicketWidget::handleDeleteTicketButtonClicked);

connect(ui->lineEdit\_ticket, &QLineEdit::textChanged,

this, &TicketWidget::handleUpdateFilterParams);

connect(ui->lineEdit\_passport, &QLineEdit::textChanged,

this, &TicketWidget::handleUpdateFilterParams);

connect(ui->lineEdit\_flight, &QLineEdit::textChanged,

this, &TicketWidget::handleUpdateFilterParams);

connect (ui->checkBox\_search, &QCheckBox::clicked,

this, &TicketWidget::handleUpdateFilterParams);

initTableView(ui->tableView\_flights);

initTableView(ui->tableView\_passengers);

initTableView(ui->tableView\_tickets);

}

void TicketWidget::handleAddTicketButtonClicked()

{

QItemSelectionModel \*selectFlight = ui->tableView\_flights->selectionModel();

auto selectedRowsFlight = selectFlight->selectedRows();

QItemSelectionModel \*selectPassengers = ui->tableView\_passengers->selectionModel();

auto selectedRowsPassengers = selectPassengers->selectedRows();

if (selectedRowsFlight.size() != 1 || selectedRowsPassengers.size() != 1)

{

qDebug() << "Выберите 1 пассажира и 1 рейс";

return;

}

auto proxyModelFlight = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView\_flights->model());

auto proxyModelPas = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView\_passengers->model());

emit sigAddTicket(proxyModelFlight->*mapToSource*(selectedRowsFlight.first()),

proxyModelPas->*mapToSource*(selectedRowsPassengers.first()));

}

void TicketWidget::handleDeleteTicketButtonClicked()

{

QItemSelectionModel \*selectTickets = ui->tableView\_tickets->selectionModel();

auto selectedRowsTickets = selectTickets->selectedRows();

if (selectedRowsTickets.size() != 1)

{

qDebug() << "Выберите 1 билет для удаления";

return;

}

auto proxyModelTic = dynamic\_cast<QSortFilterProxyModel\*>(ui->tableView\_tickets->model());

auto model = dynamic\_cast<ModelTickets\*>(proxyModelTic->sourceModel());

if (!model)

return;

model->removeTicket(proxyModelTic->*mapToSource*(selectedRowsTickets.first()));

}

QAbstractItemModel \*TicketWidget::createProxyModels(QAbstractItemModel \*sourceModel)

{

auto proxyModel = new QSortFilterProxyModel();

proxyModel->*setSourceModel*(sourceModel);

return proxyModel;

}

void TicketWidget::initTableView(QTableView \*tableView)

{

tableView->horizontalHeader()->setStretchLastSection(true);

tableView->setSortingEnabled(true);

tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

}

void TicketWidget::handleUpdateFilterParams()

{

TicketSortFilterProxyModel::FilterParams params;

params.ticket = ui->lineEdit\_ticket->text();

params.passport = ui->lineEdit\_passport->text();

params.flight = ui->lineEdit\_flight->text();

params.isFilterEnabled = ui->checkBox\_search->isChecked();

emit sigUpdateFilterParams(params);

}

TicketWidget::~*TicketWidget*()

{

delete ui;

}

void TicketWidget::initLineEdit()

{

{

QRegExp regExp("[0-9]+");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_ticket->setValidator(validator);

}

{

QRegExp regExp("\\d{4}[ ]\\d{6}");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_passport->setValidator(validator);

}

{

QRegExp regExp("[A-Z][A-Z][ ][0-9]+");

QRegExpValidator \*validator = new QRegExpValidator(regExp, this);

ui->lineEdit\_flight->setValidator(validator);

}

}

void TicketWidget::setModels(QAbstractTableModel \*ticketModel,

QAbstractTableModel \*passengerModel,

QAbstractTableModel \*flightModel)

{

ui->tableView\_passengers->*setModel*(createProxyModels(passengerModel));

ui->tableView\_flights->*setModel*(createProxyModels(flightModel));

auto ticketProxyModel = new TicketSortFilterProxyModel();

ticketProxyModel->*setSourceModel*(ticketModel);

ui->tableView\_tickets->*setModel*(ticketProxyModel);

connect(this, &TicketWidget::sigUpdateFilterParams,

ticketProxyModel, &TicketSortFilterProxyModel::handleUpdateFilterParams);

}