СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных условиях развития процессов глобализации экономики, увеличения товарных потоков, расширения международных связей, их достойное обеспечение возможно лишь при наличии отлаженной работы авиационной отрасли. Грамотное обеспечение авиаперевозок, поддержание высокого уровня развития аэропортового и аэродромного хозяйства является залогом комфорта и выгоды пассажиров и грузоотправителей, ключевым моментом, определяющим динамику экономического развития страны.

Набирающие ход процессы модернизации аэропортов затрагивают как внедрение современных моделей управления, так и обновление парка наземного оборудования и IT-инфраструктуры. На очереди – совершенствование технологий наземного обслуживания.

Традиционно наземное обслуживание как бизнес – это постоянные колебания, взлеты и падения, что в свою очередь ставит задачи по управлению персоналом и эксплуатации оборудования. Сегодня эти задачи можно решать с использованием интегрированного программного обеспечения планирования, которое поможет оптимизировать работу персонала наземного обслуживания и увеличить его коэффициент полезного действия. Исходя из вышесказанного, встает необходимость немедленно приступить к окончательной доработке разрабатываемой системы, в плане автоматизации бизнес-процессов аэропорта.

Таким образом, курсовой проект ставит перед собой следующие цели:

– уменьшение вероятности ввода ошибочной информации в базу данных;

– минимизация участия человека в процессе учетной деятельности;

– автоматизация создания результирующих документов.

Выполнение этих целей позволит ускорить работу касс, увеличить электронный документооборот и повысить качество предоставляемых услуг.

Поставленная цель потребовала решения следующих задач:

− проанализировать процесс деятельности аэропорта;

− ознакомиться с особенностями выполнения авиа услуг;

− изучить основные проблемы и способы их решения;

− определить методы автоматизации;

− разработать алгоритмы работы программы и их схемы;

− спроектировать и реализовать базу данных;

− выполнить программную реализацию.

Объектом исследования является аэропорт.

Предметом исследования являются методы ресурсно-временной оптимизации процесса ОУ аэропортом.

1 ОПИСАНИЕ ПРОЦЕССА РАБОТЫ АЭРОПОРТА

В наше время воздушный транспорт (в частности самолеты) является наиболее быстрым и особенно ценится при перемещении на далекие расстояния. В мире существует множество аэропортов и соответственно еще больше маршрутов полетов.

Современные люди проводят в путешествиях немало времени, а для многих аэропорт – как второй дом.

Крупные аэропорты имеют несколько вестибюлей и пассажирских терминалов. Путеводители по аэропортам даже размещают информацию о каждом терминале на отдельной странице, поскольку в них расположено значительное количество важных для пассажиров объектов. Перемещение из одного в другой конец терминала может занимать около часа.

Под аэропортом располагается целая сеть конвейерных лент для багажа. Багаж поступает сюда по специальным «рукавам». Процедура проверки багажа тщательно продумана. Сначала на стойке регистрации на каждый чемодан наклеивается бирка. Затем сканеры считывают информацию, и багаж перемещается по ленте. В некоторых случаях требуется дополнительная ручная проверка.

После этого пассажир проходит через контроль. Зеленым коридором стоит пользоваться, если количество наличной валюты, алкоголя, сигарет и прочих лимитируемых веществ не превышает норму. Если пассажир не уверен в этом, лучше воспользоваться красным коридором. В крайнем случае, нужно будет заплатить пошлину, а в случае превышения нормы в зеленом коридоре могут взимать крупный штраф и завести дело об административном правонарушении.

Пассажирам стоит помнить и о нормах ввоза в страну прибытия – нередко за покупки в магазинах беспошлинной торговли приходится платить значительную сумму штрафа.

После регистрации пассажиры проходят на посадку. Высадка и посадка в самолеты осуществляется либо через «рукав», либо на автобусе. Поскольку в первом случае с авиакомпаний берут дополнительную плату, многие из них экономят и пользуются автобусом.

Стоянка самолета имеет номерную маркировку, как и обычная автомобильная парковка. Перон летного поля используется для стоянки воздушных судов во время посадки – высадки пассажиров и размещения багажа.

Затем самолет следует на взлетно-посадочную полосу.

По прибытию в пункт назначения, к самолету сразу же подъезжает машина, которая забирает багаж. Грузчики выгружают его на ленту, откуда он направляется прямо на выдачу.

Далее производится уборка самолета. Если уборщики обнаруживают забытые в салоне вещи, то их передают авиакомпании. Параллельно с уборкой самолет заправляют топливом.

Таким образом, можно сделать вывод о том, что аэропорт – довольно сложная система, основной целью которой является качественное обслуживание клиентов, которое заключается в транспортировке пассажиров и грузов, предоставление услуг, направленных на улучшение удобства перелёта. Вместе с этим необходимо обеспечить выполнение ряда других задач, таких как найм персонала, поддержание оборудования в хорошем состоянии, ремонт, обеспечение авиационной безопасности, составление и изменение расписания полётов, продажа и бронирование билетов и др.

На рисунке 1.1 представлена схема аэровокзального комплекса аэропорта.

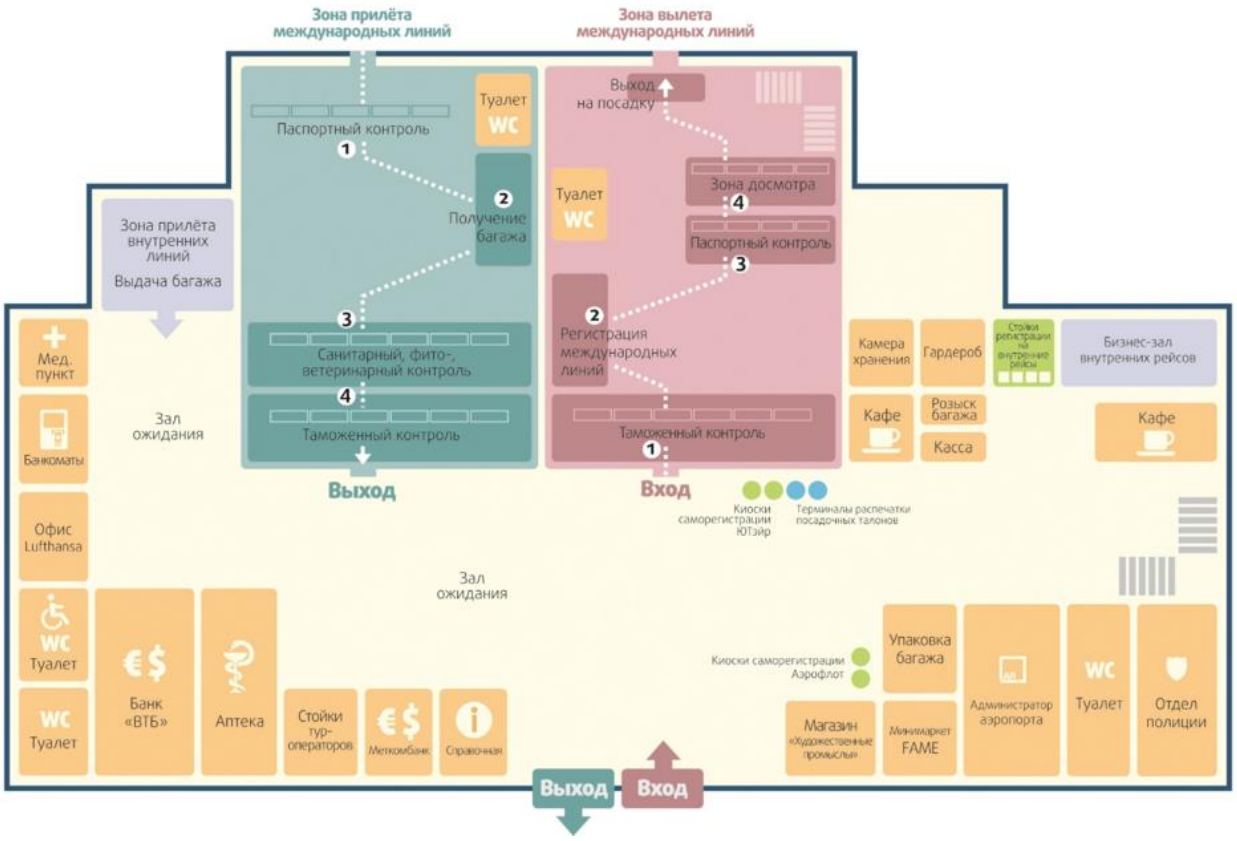

Рисунок 1.1 – Схема аэровокзального комплекса аэропорта

Организационная структура аэропорта представлена на рисунке 1.2.

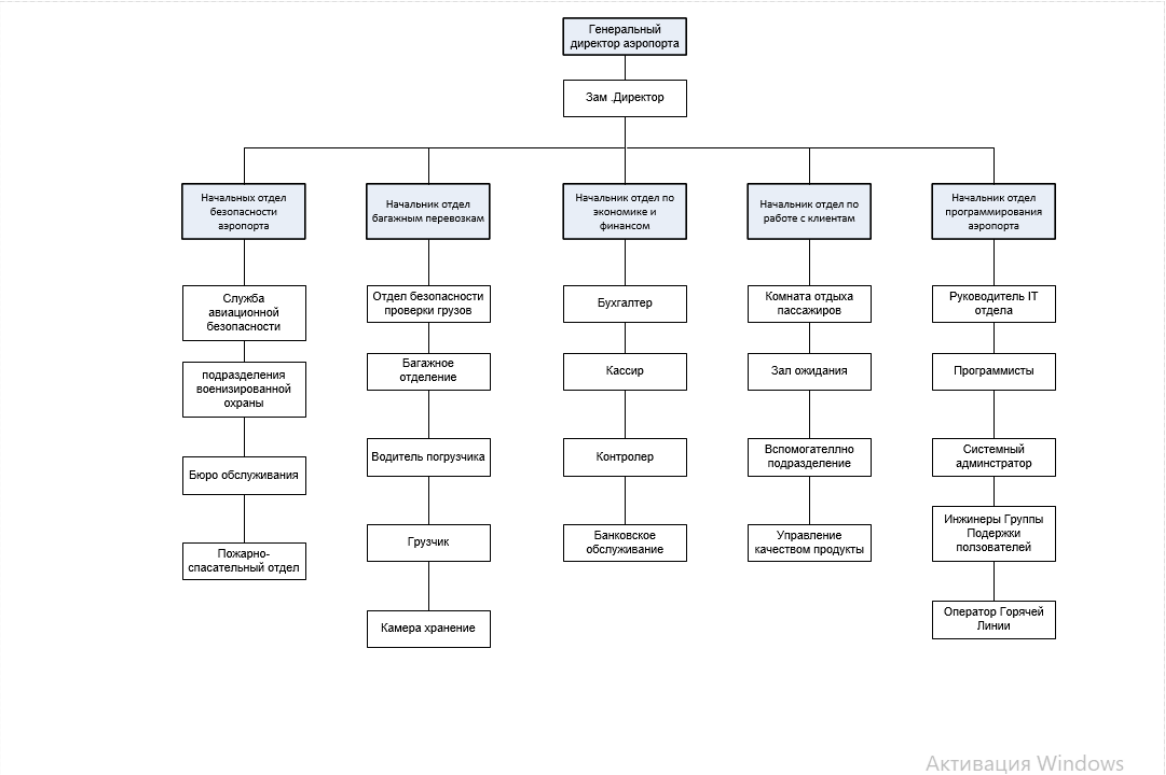

Рисунок 1.1 – Организационная структура аэропорта

Потоки информации, циркулирующие в мире, который нас окружает, огромны. Во времени они имеют тенденцию к увеличению. Поэтому в любой организации, как большой, так и маленькой, возникает проблема такой организации управления данными, которая обеспечила бы наиболее эффективную работу.

Существует много веских причин перевода существующей информации на компьютерную основу. Использование клиент/серверных технологий позволяют сберечь значительные средства, а главное и время для получения необходимой информации, а также упрощают доступ и ведение, поскольку они основываются на комплексной обработке данных и централизации их хранения.

Использование новых, современных технологий при работе с пассажирами является одним из основных направлений деятельности аэропорта. Предоставление информации и обслуживание клиентов становится одним из ключевых факторов эффективной работы аэропорта.

Используя данную систему, пассажир в любое время может получить подробную информацию об актуальных рейсах, а также заказать билет для себя и своих близких или друзей, избавляясь от ненужного и утомительного ожидания в аэропорту, тем самым уменьшая нагрузку обслуживания клиентов аэропорта.

2 ПОСТАНОВКА ЗАДАЧИ НА РАЗРАБОТКУ ПРОГРАММНОГО СРЕДСТВА И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

Как уже говорилось выше необходимо разработать комплекс программ для информационной системы "Аэропорт", предназначенной для автоматизации работы сотрудников аэропорта и предоставления им доступа к необходимой информации, а также для выдачи запрашиваемой информации пассажирам. Комплекс программ должен состоять из следующих частей:

– приложение для сотрудников аэропорта;

– приложение для туристов, которые хотят забронировать место на рейс.

Информационная система для сотрудников должна обеспечивать следующий функционал:

– добавление, удаление и редактирование личных данных туристов;

–добавление, удаление и редактирование зарегистрированных пользователей;

– предоставление информации о сотрудниках;

– мониторинг рейсов, включая их редактирование и добавление в любое время;

– сортировка, фильтрация, формирование отчетов о всех полетах;

– регистрация на рейсы для выбранных туристов.

Информационная система для пассажиров должна обеспечивать следующий функционал:

– выбор рейса;

– мониторинг возможных рейсов;

– возможность управления своим аккаунтом.

Значительным преимуществом автоматизации данных операций является скорость их выполнения, возможность в любой момент осуществить запрос к данным, находящимся в базе и получить по этому запросу исчерпывающую информацию. Программа позволяет отказаться от необходимости ведения бумажных архивов. В ходе её использования снижается вероятность введения ошибочной информации.

Пользователь программы (кассир аэропорта) должен получить от клиента ряд сведений, необходимых для оформления заказа. А именно, дату вылета самолёта, номер рейса (хранящий в себе пункт назначения) и класс желаемого места. Для решения такой задачи будет очень полезным использование современных информационных технологий, а именно разработка автоматизированного системного приложения с клиент-серверной архитектурой.

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

Для полного представления функций аэровокзального комплекса аэропорта была разработана функциональная модель приложения. Для данной цели был выбран стандарт IDEF0. IDEF0 – методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов.

Ключевым процессом в данной теме является авиаперелет пассажиров.

На контекстной диаграмме верхнего уровня (рисунок 3.1) представлена функциональная модель «Авиаперелет», а также определены потоки входных и выходных данных, механизмы ограничения и управления данными.

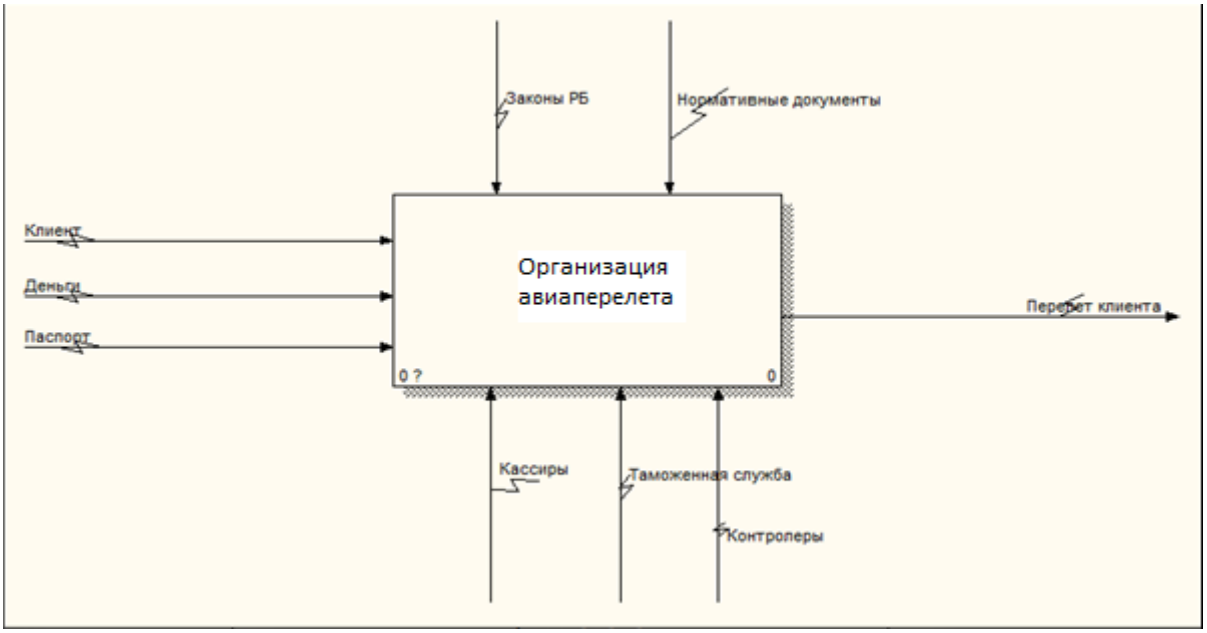

Рисунок 3.1 – Контекстная диаграмма авиаперелета пассажиров

Входными данными являются клиент, деньги и паспорт; управляющими механизмами являются законы РБ и нормативные документы; механизмами являются кассиры, контроллеры и таможенная служба. Целью данной бизнес-модели является перелет клиента, который является выходным параметром данной диаграммы.

Декомпозиция данной диаграммы подробно представлена на рисунке 3.2. Она состоит из следующих блоков: «Обратиться в кассу», «Зарегистрироваться на рейс», «Пройти багажно-таможенный контроль», «Осуществить перелет».

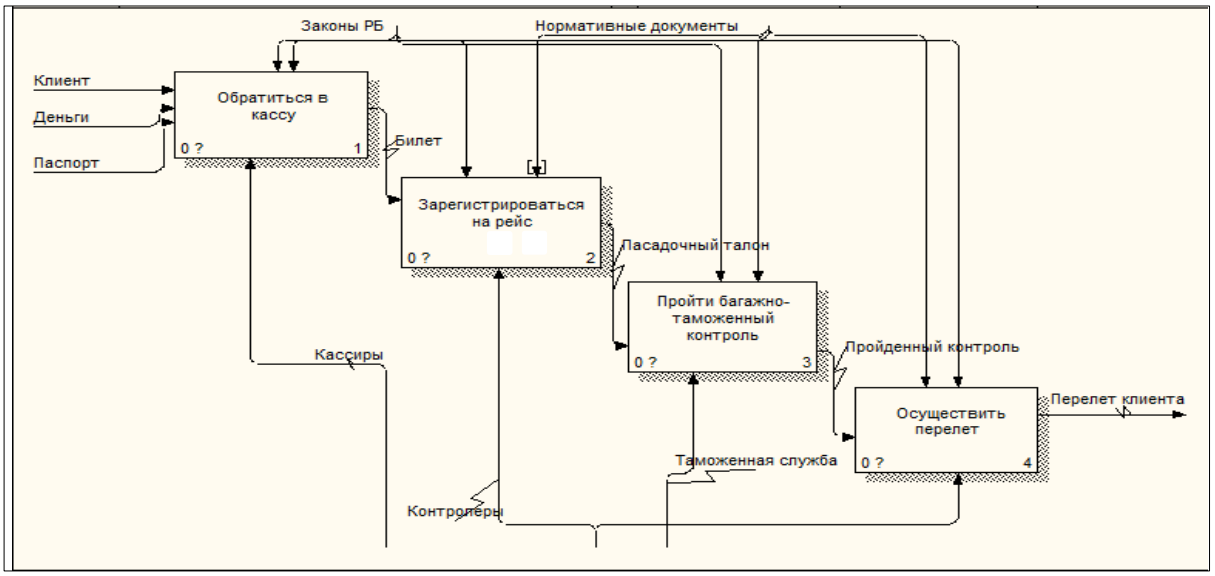

Рисунок 3.2 – Декомпозиция главного блока

Первый блок данной декомпозиции «Обратиться в кассу» подразумевает получение информации о рейсе, выбор подходящего рейса и покупка билета.

Декомпозиция данного блока представлена на рисунке 3.3.

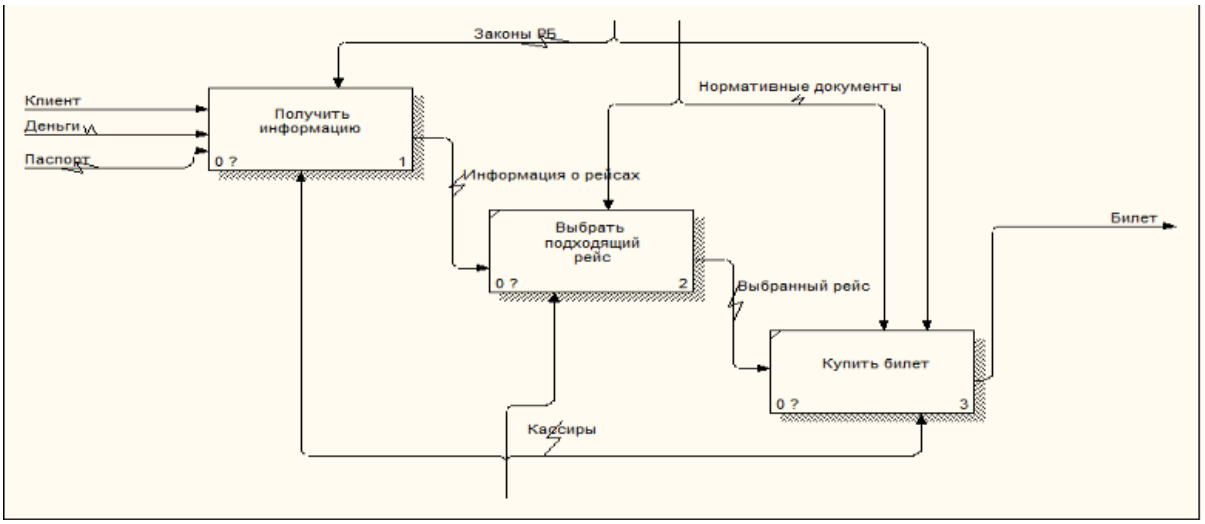

Рисунок 3.3 – Декомпозиция блока «Обратиться в кассу»

Данная декомпозиция состоит из следующих этапов: «Получить информацию», «Выбрать подходящий рейс», «Купить билет».

Этап «Получить информацию» подразумевает получении информации о маршруте, дате вылета и количестве свободных мест. Декомпозиция данного этапа представлена на рисунке 3.4.

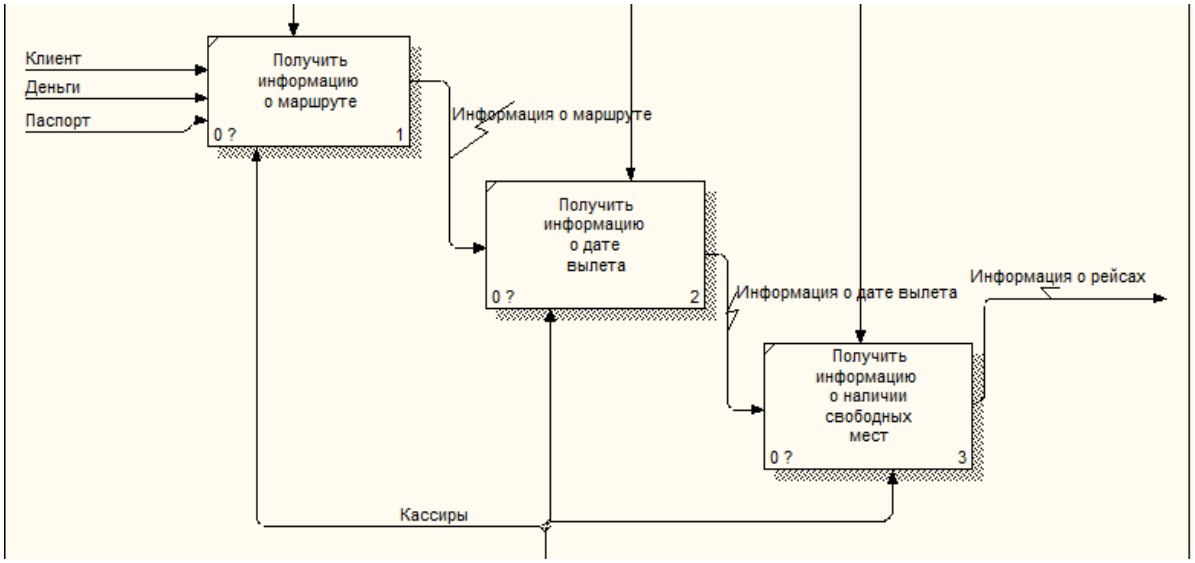

Рисунок 3.4 – Декомпозиция блока «Получить информацию»

Вторым блоком декомпозиции главного процесса является «Зарегистрироваться на рейс», который предполагает предоставление пассажиром паспорта и билета, для последующей регистрации.

На рисунке 3.5 представлена декомпозиция данного блока.

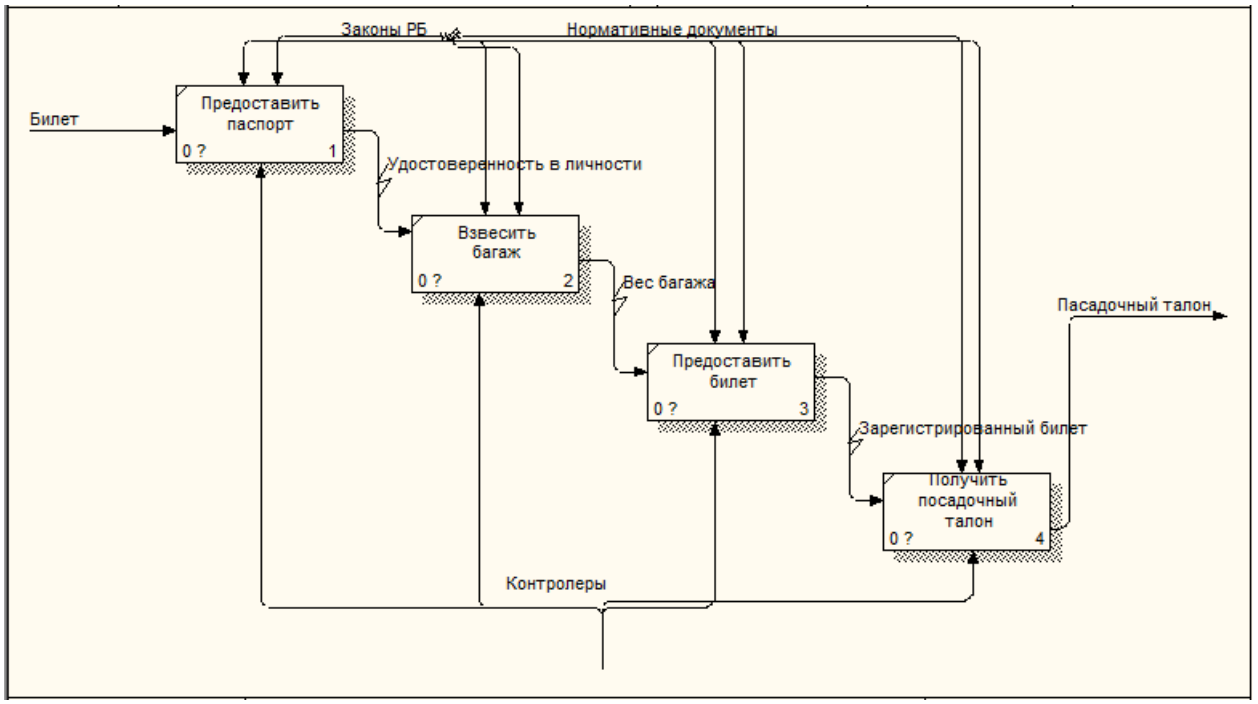

Рисунок 3.5 – Декомпозиция блока «Зарегистрироваться на рейс»

Данная декомпозиция состоит из следующих блоков: «Предоставить паспорт», «Взвесить багаж», «Предоставить билет», «Получить посадочный талон».

Третьим блоком главного процесса является «Пройти багажно-таможенный контроль», который подразумевает прохождение контроля, удостоверяющего личность, и предполетного осмотра.

На рисунке 3.6 представлена декомпозиция данного блока.

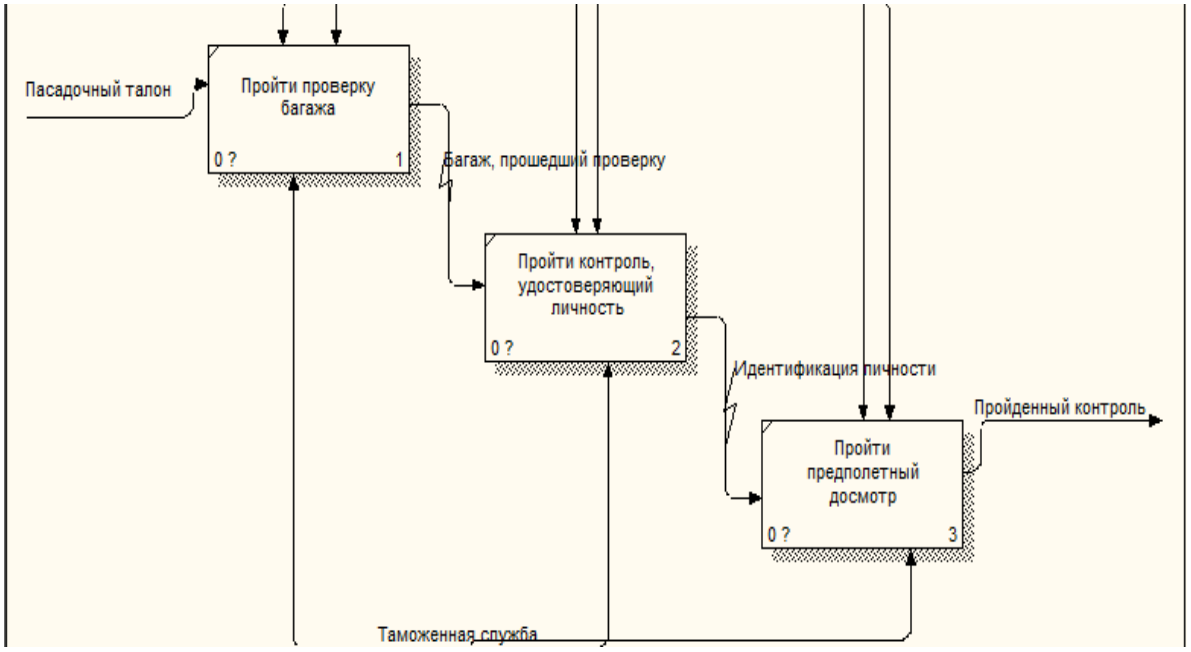

Рисунок 3.6 – Декомпозиция блока «Пройти багажно-таможенный контроль»

Данная декомпозиция состоит из следующих блоков: «Пройти проверку багажа», «Пройти контроль, удостоверяющий личность», «Пройти предполетный досмотр».

Последним этапом главного процесса является «Осуществить перелет», который подразумевает предоставление посадочного талона и прохождение на свое посадочное место.

Декомпозиция данного процесса представлена на рисунке 3.7.

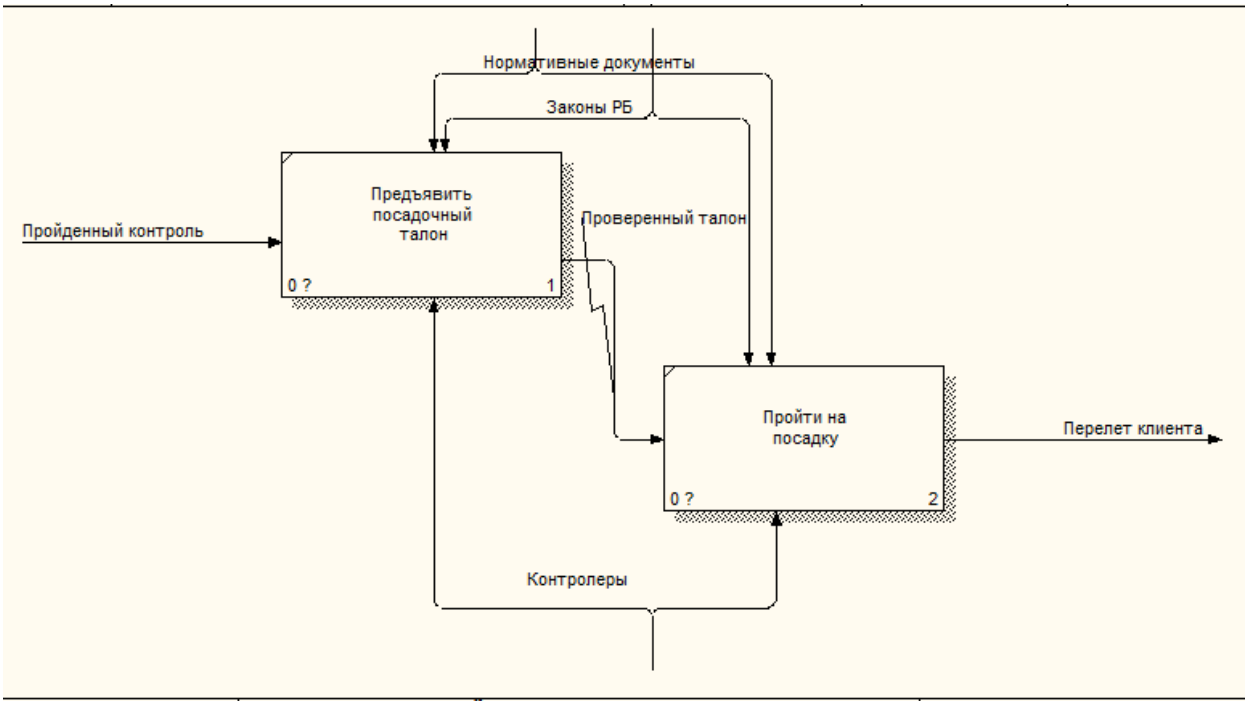

Рисунок 3.7 – Декомпозиция блока «Осуществить перелет»

Данная декомпозиция состоит из следующих блоков: «Предъявить посадочный талон», «Пройти на посадку».

Таким образом, из представленных декомпозиций можно сделать вывод о том, какой сложно является работа аэровокзального комплекса. Не возникает сомнений, что такая работа нуждается в автоматизации некоторых процессов для повышения качества обслуживания пассажиров.

4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ

Информационная модель в данном курсовом проекте была построена с помощью приложения DBeaver. DBeaver – это клиент SQL и средство администрирования базы данных. Для реляционных баз данных он использует JDBC API для взаимодействия с базами данных с помощью драйвера JDBC. Он предоставляет редактор, поддерживающий компиляцию кода и подсветку синтаксиса. Он предоставляет архитектуру плагинов, которая позволяет модифицировать поведение большой части приложения для обеспечения специальных функций базы данных, которые не зависят от самой базы данных.

Сущностями данной модели являются appointment, client, employee, employeeservice, service и user.

Сущность «appointment» содержит информацию о дате вылета, дате прилета обратно, id туриста и id закрепленного за ним сотрудника.

Сущность «client» содержит информацию о туристе: идентификационный номер, имя, фамилия, номер его рейса, email.

Сущность «employee» содержит информацию о сотрудниках: имя и фамилия, а также id.

Сущность «employeeservice» содержит информацию о рейсах, за который отвечает тот или иной сотрудник.

Сущность «service» содержит информацию о самом рейсе: его название, цена и страна, в которую осуществляется полет.

Сущность «user» содержит информацию о зарегистрированных сотрудниках: логин, пароль, роль и id.

Общая схема связанных сущностей базы данных представлена на рисунке 4.1.

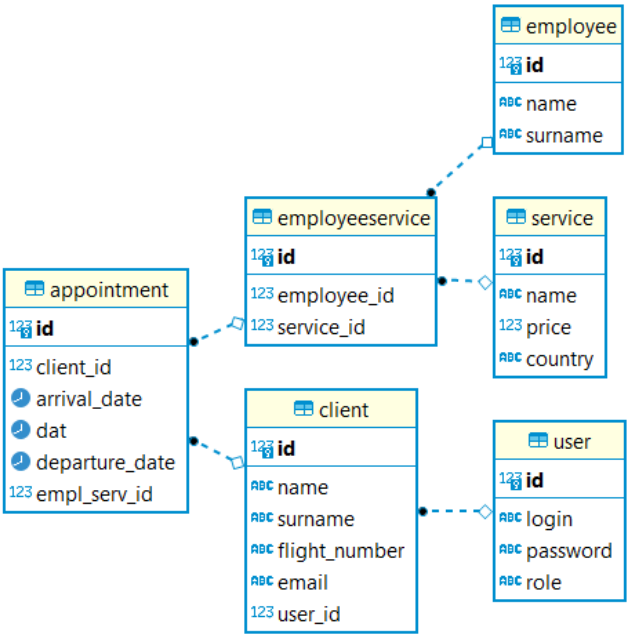

Рисунок 4.1 – Информационная модель базы данных

Для доказательства нахождение таблиц в третьей нормальной форме необходимо воспользоваться следующими понятиями. Таблица находится в первой нормальной форме, если все ее поля имеют простые значения. Таблица находится во второй нормальной форме, если она находится в первой нормальной форме, а каждое не ключевое поле функционально полно зависит от составного ключа. Таблица находится в третьей нормальной форме, если она находится во второй нормальной форме, и каждое не ключевое поле не транзитивно зависит от первичного ключа.

Таким образом, база данных приведена к третей нормальной форме, поскольку у каждой таблицы имеется всего один первичный ключ, а каждое не ключевое поле не транзитивно зависит от первичного ключа, т.е. изменив значение в одном столбце не потребуется изменение в другом столбце.

5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ НА ОСНОВЕ UML

5.1 Спецификация вариантов использования системы

Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать. Диаграмма вариантов использования представлена на рисунке 4.1. Существует два актера: администратор и пользователь. Они имеют четыре общих вариантов использования: вывод отчета, ранжирование видов методом последовательных сравнений, просмотр данных табличной формы и прочитать все данные из файла. Все остальные варианты использования различаются.

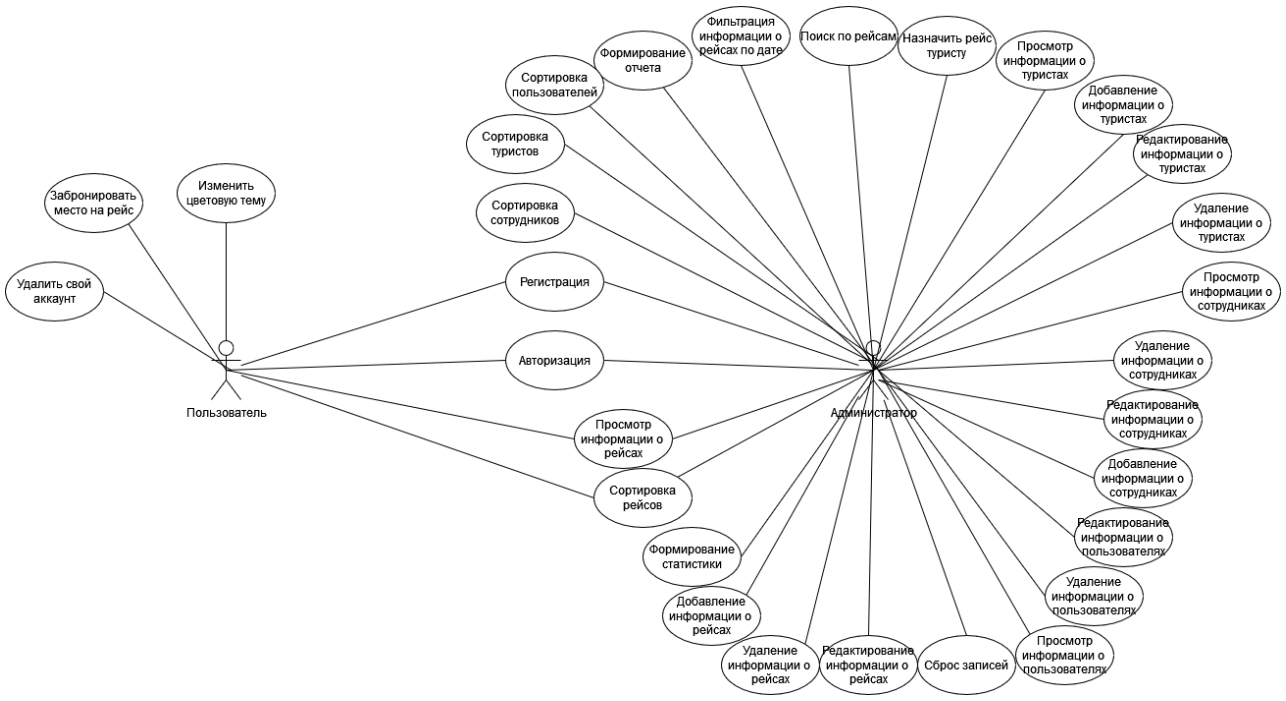

Рисунок 5.1 – Диаграмма вариантов использования

5.2 Описание диаграммы последовательности

Диаграмма последовательности описывает поведение только одного варианта использования. На такой диаграмме отображаются только экземпляры объектов и сообщения, которыми они обмениваются между собой.

На рисунке 5.2 представлена диаграмма деятельности авторизации пользователя.

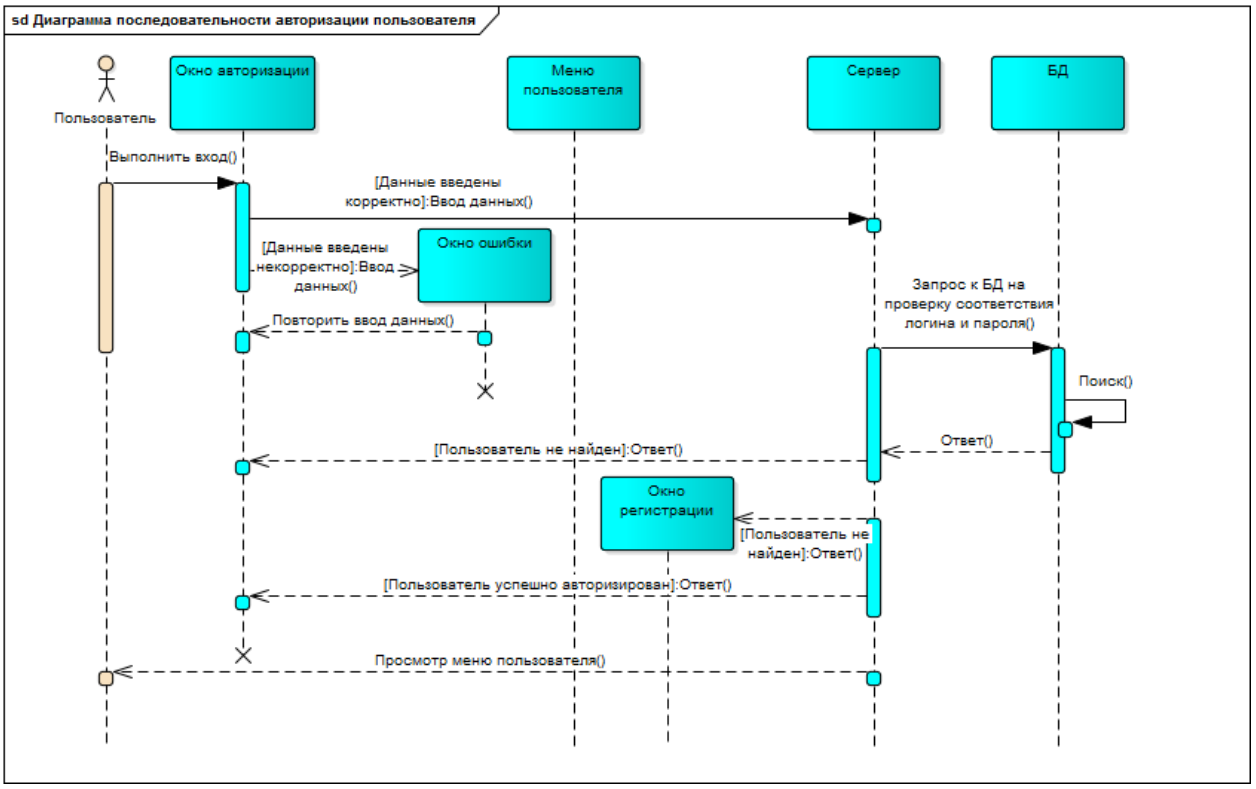

Рисунок 5.2 – Диаграмма последовательности авторизации пользователя

На данной диаграмме пользователь выполняет вход в программу, вводит данные (логин и пароль) в окне авторизации.

В случае корректно введенных данных сервер посылает заброс к БД на проверку соответствия логина и пароля, т. е. введенные логин и пароль пользователя сравнивается со всеми имеющимися в БД логинами и паролями пользователей. Если соответствие найдено, сервер информирует пользователя об успешной авторизации. При этом жизненный цикл окна авторизации прекращается, а пользователю предоставляется меню для последующей работы с системой.

Если данные были введены некорректно, появляется окно ошибки, информирующее о неправильно введенных данных.

Если в БД не было найдено соответствие логину и паролю, которые ввел пользователь, сервер информирует пользователя об отсутствии такого пользователя.

5.3 Диаграмма состояний

Диаграмма состояний описывает все возможные состояния одного экземпляра определенного класса и возможные последовательности его переходов из одного состояния в другое, то есть моделирует все изменения

состояний объекта как его реакцию на внешние воздействия [11].

На рисунке 5.3 представлена диаграмма состояний билета.

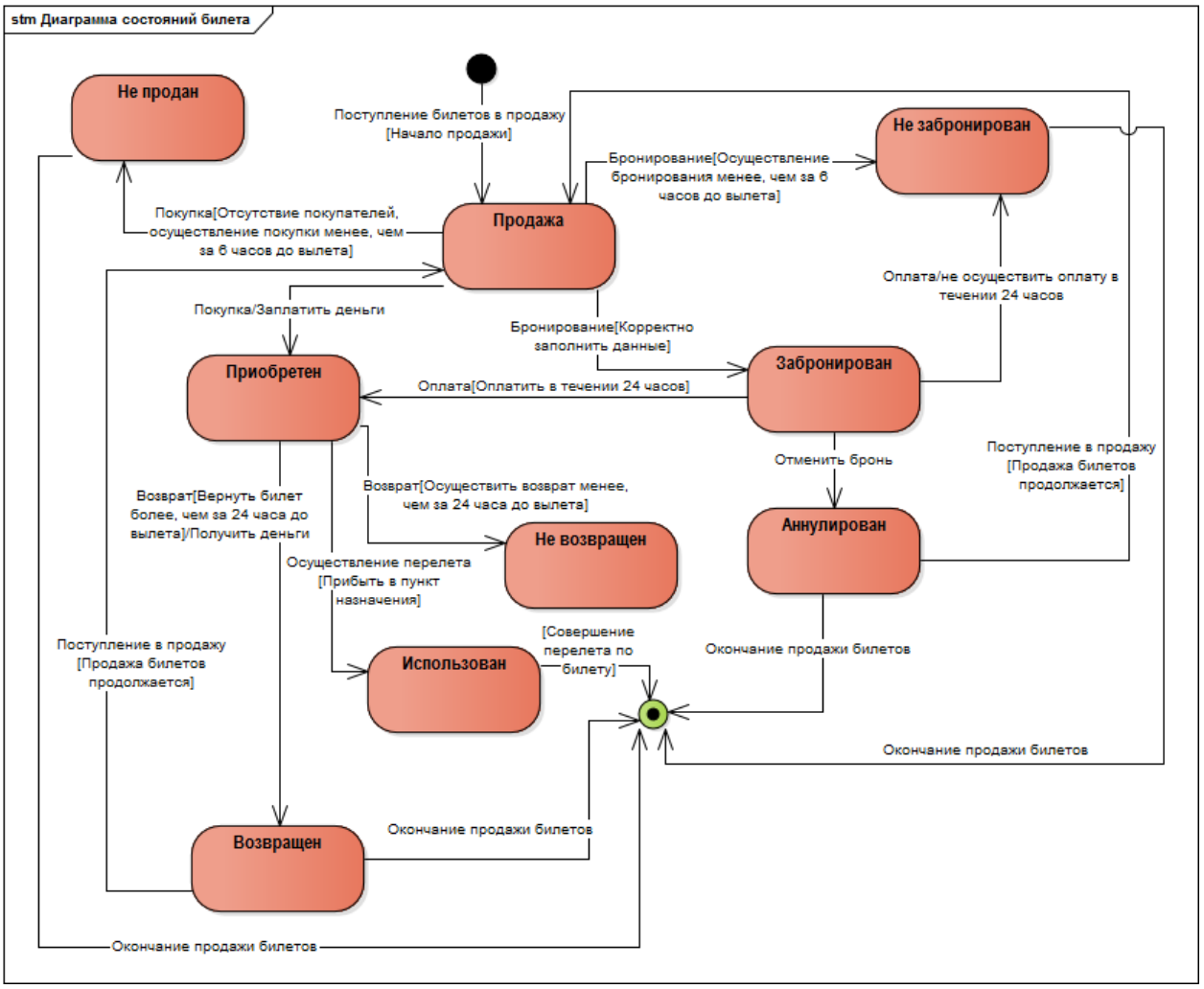

Рисунок 5.3 – Диаграмма состояний билета

Начальным состоянием данной диаграммы является поступление билетов в продажу, затем билет переходит в состояние «продажи». Из состояния «продажи» билет попадает в состояние «не продан» при условии отсутствия покупателей или осуществления покупки менее, чем за 6 часов до вылета. Билет попадает в состояние «не забронирован» при условии осуществления бронирования менее, чем за 6 часов до вылета. Также из состояния продажи билет попадает в состояние «приобретен», если произошла покупка и пассажир заплатил деньги. Из состояния «приобретен» билет попадает в состояние «возвращен», если пассажир вернул билет и забрал деньги. Также из состояний «приобретен» билет попадает в состояние использован, если пассажир осуществил перелет по билету и прибыл в пункт назначения. Из состояния «использован» билет попадает в конечное состояние.

В состояние «забронирован» билет попадает при условии корректно введенных данных. При этом из состояния «забронирован» при неоплате билета в течении 24 часов, он попадает в состояние «не забронирован». Из состояния «забронирован» билет попадает в «аннулирован», если пассажир отменил бронь. Из состояния «аннулирован» и «возвращен» билет может снова попасть в состояние «продажи», если продажа билетов продолжается, или же в конечное состояние, если продажа билетов завершилась. Также из состояния «приобретен» билет попадает в состояние «не возвращен», если возврат осуществляется менее, чем за 24 часа до вылета.

Конечным состоянием данной диаграммы является «окончание продажи билетов».

5.4 Диаграмма развертывания и компонентов

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения. При этом представляются только компоненты-экземпляры программы, являющиеся исполняемыми файлами или динамическими библиотеками.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними.

Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходныйкомпонентов системы.

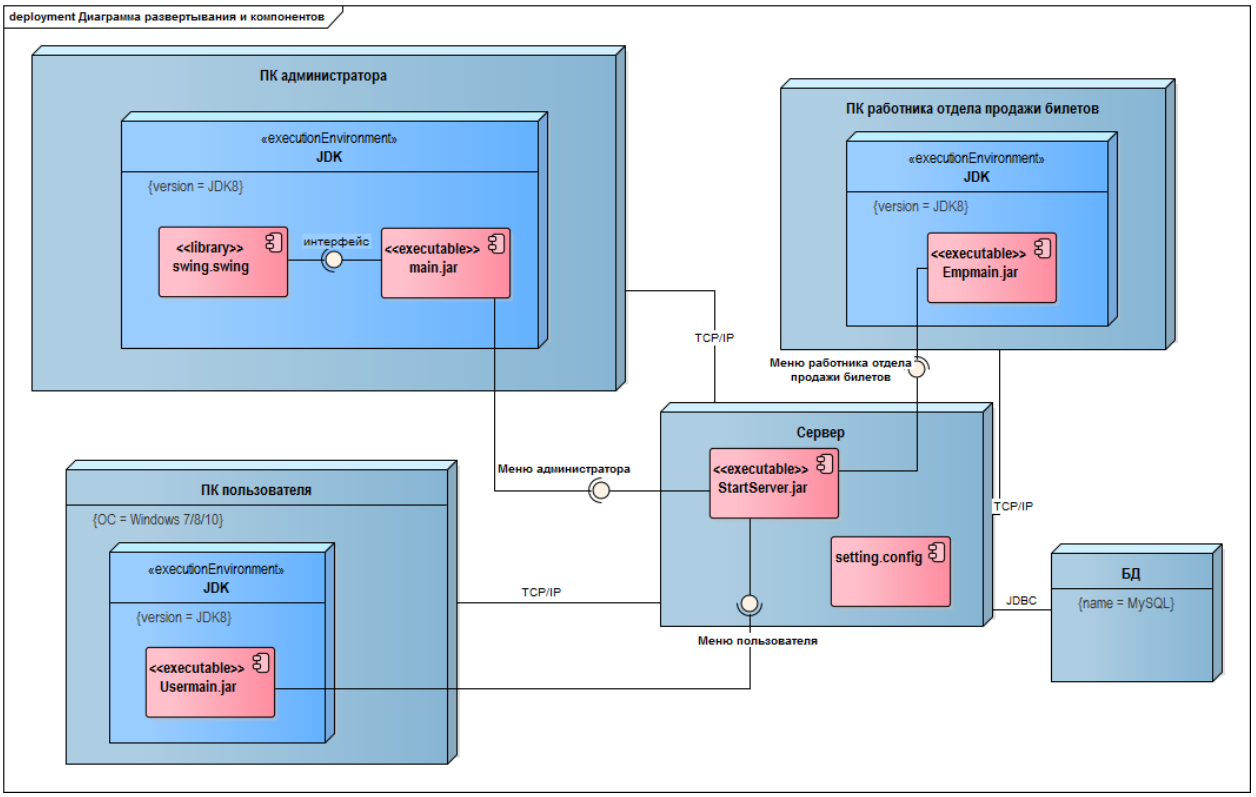

Рисунок 5.4 – Диаграмма развертывания и компонентов системы

На данной диаграмме в качестве узлов выступают ПК пользователя, ПК администратора, ПК работника отдела продажи билетов, сервер и БД.

Настройки сервера содержатся в компоненте setting.config, они необходимы для запуска сервера.

Для запуска разрабатываемого приложения необходимо наличие исполняемой среды JDK версии JDK8 на компьютерах пользователя, администратора и работника отдела продаж билетов.

Связь узлов ПК пользователя, ПК администратора и ПК работника отдела продаж билетов с сервером осуществляется по протоколу TCP/IP, а связь сервера с БД осуществляется по протоколу JDBC.

5.5 Диаграмма классов

Диаграмма классов описывает типы объектов системы и различные отношения между ними.

На рисунке 5.5 представлена диаграмма классов пакета AdminFXML.

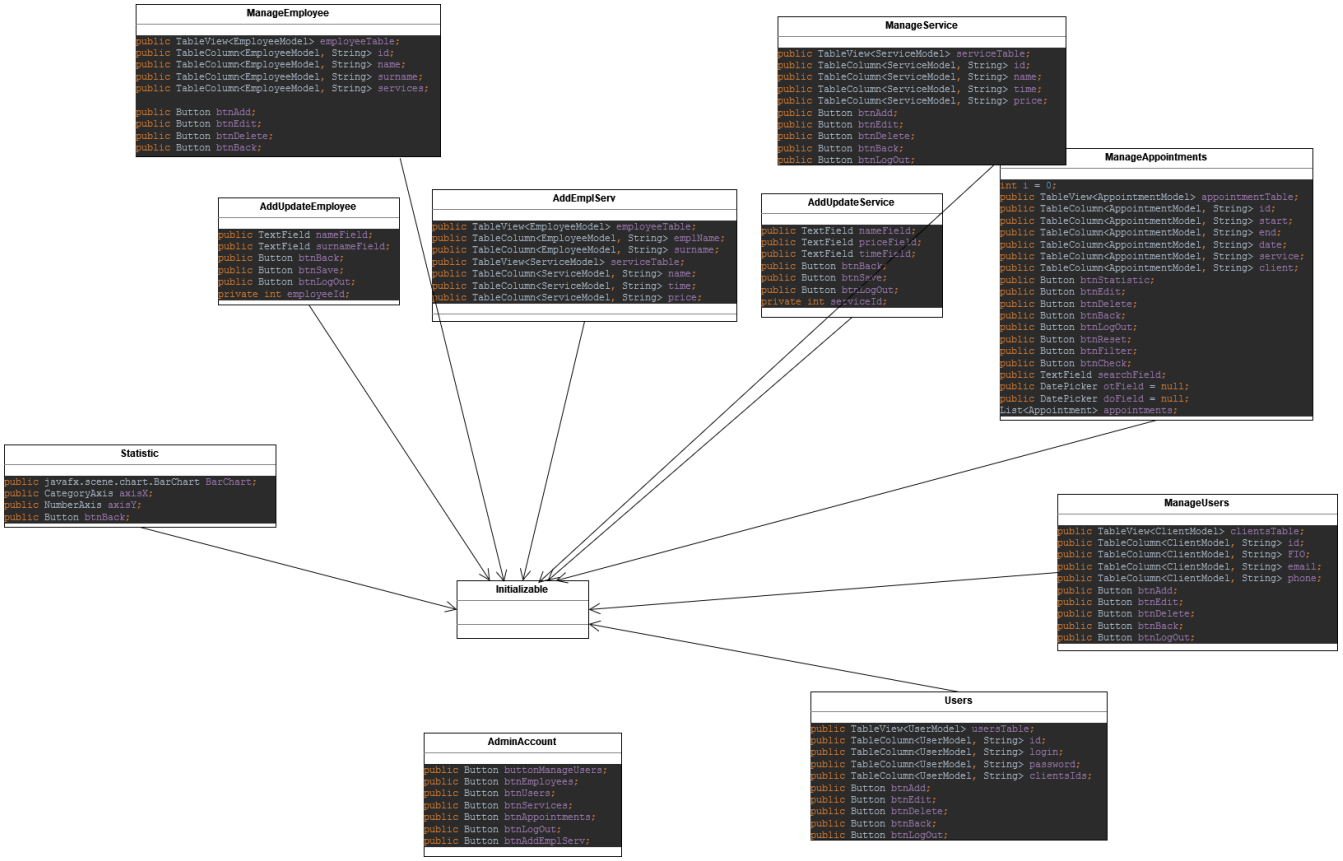

Рисунок 5.5 – Диаграмма классов пакета AdminFXML

6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ

6.1 Схема алгоритма клиент-серверного взаимодействия

На рисунке 6.1 представлена схема алгоритма авторизации пользователя.

Рисунок 6.1 – Клиент-серверное взаимодействие

6.2 Алгоритм работы администратора

Данный алгоритм позволяет выполнять администратору все необходимые ему функции: просматривать текущее расписание рейсов, удалять, корректировать и добавлять расписание, просматривать списки пассажиров, списки самолетов, а также добавлять информацию о самолетах, просматривать статистику планируемых рейсов.

Блок-схема данного алгоритма представлена на рисунке 6.2.

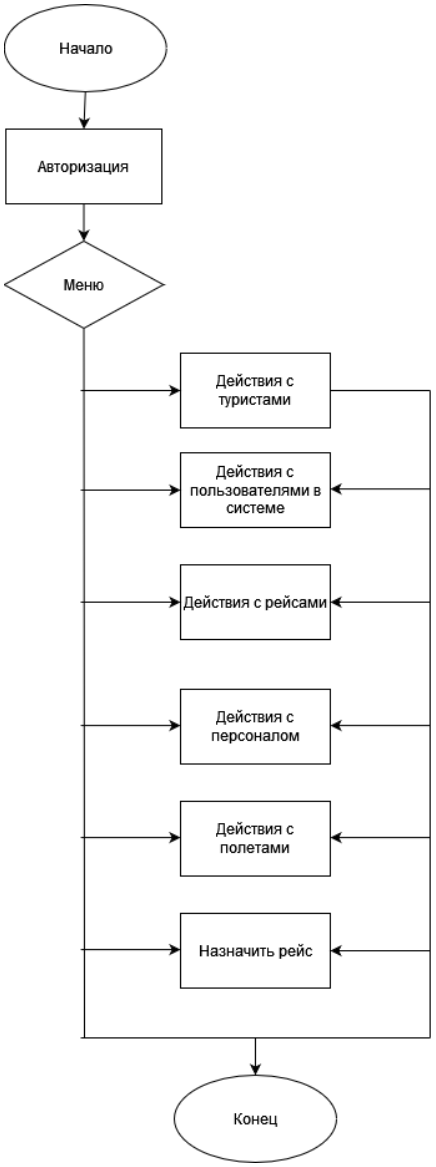

Рисунок 6.2 – Блок-схема алгоритма работы администратора

6.3 Алгоритм работы пользователя

Данный алгоритм позволяет выполнять пользователю все необходимые ему функции: регистрироваться, просматривать текущее расписание рейсов, заказывать билеты, просматривать информацию о заказанных билетах.

Блок-схема данного алгоритма представлена на рисунке 6.3.

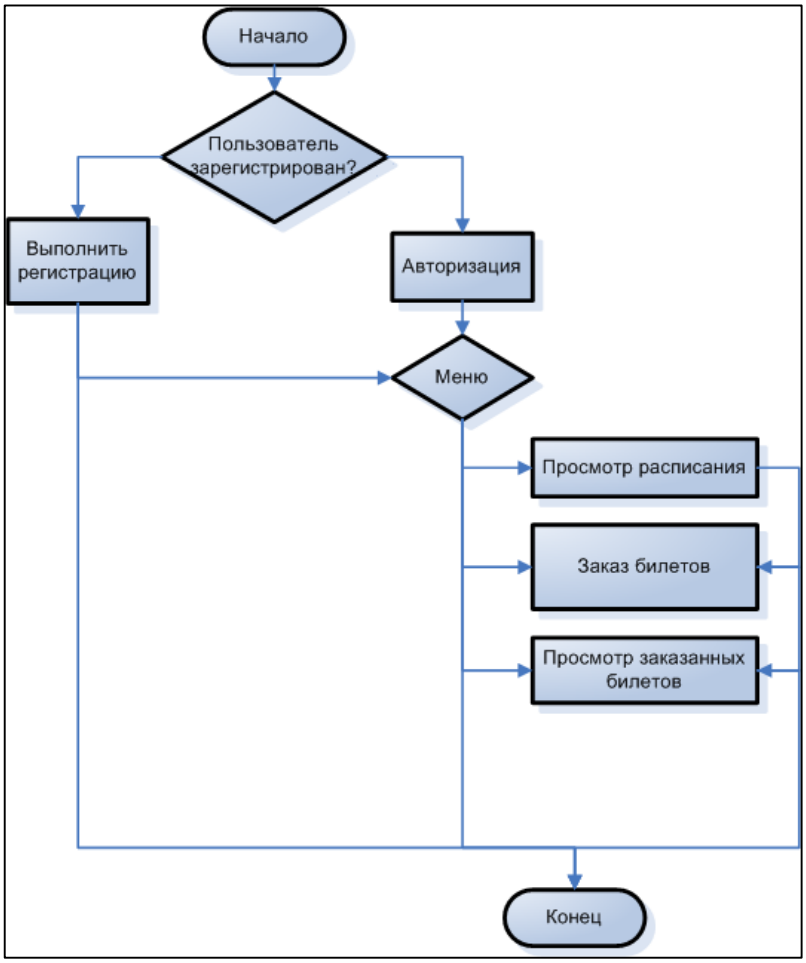

Рисунок 6.3 – Блок-схема работы пользователя

6.4 Алгоритм добавления информации о рейсе

Данный алгоритм позволяет добавлять информацию о рейсах в базу.

Для добавления информации необходимо получить данные на добавление от клиента, создать запрос на проверку существования добавляемого самолета с уже существующими. Если информацию уже существует в базе данных, посылается соответствующее сообщение. Если же информация не существует в базе, то она добавляется и сервер уведомляет клиента об успешном добавлении записи.

```
      timeChoice.setItems(tm);
      time.setCellValueFactory(new PropertyValueFactory<>("time"));
      name.setCellValueFactory(new PropertyValueFactory<>("name"));
      price.setCellValueFactory(new PropertyValueFactory<>("price"));
      GetService<Service> batchGetService = new GetService<>(Service.class);
      Type listType = new TypeToken<ArrayList<Service>>() {
      }.getType();
      List<Service> services = new
Gson().fromJson(batchGetService.GetEntities(Requests.GET_ALL_SERVICES),
listType);
      ObservableList<ServiceModel> list = FXCollections.observableArrayList();
      if (services.size() != 0)
      for (Service service : services) {
      ServiceModel tableService = new ServiceModel(service.getId(),
service.getName(),
      service.getTime(), service.getPrice());
                              list.add(tableService);
                          }

      serviceTable.setItems(list);
      FIO.setCellValueFactory(new PropertyValueFactory<>("FIO"));
                      id.setCellValueFactory(new PropertyValueFactory<>("id"));

                      clientsTable.setItems(list2);

                      emplName.setCellValueFactory(new
PropertyValueFactory<>("name"));
                      surname.setCellValueFactory(new
PropertyValueFactory<>("surname"));
                  }
```

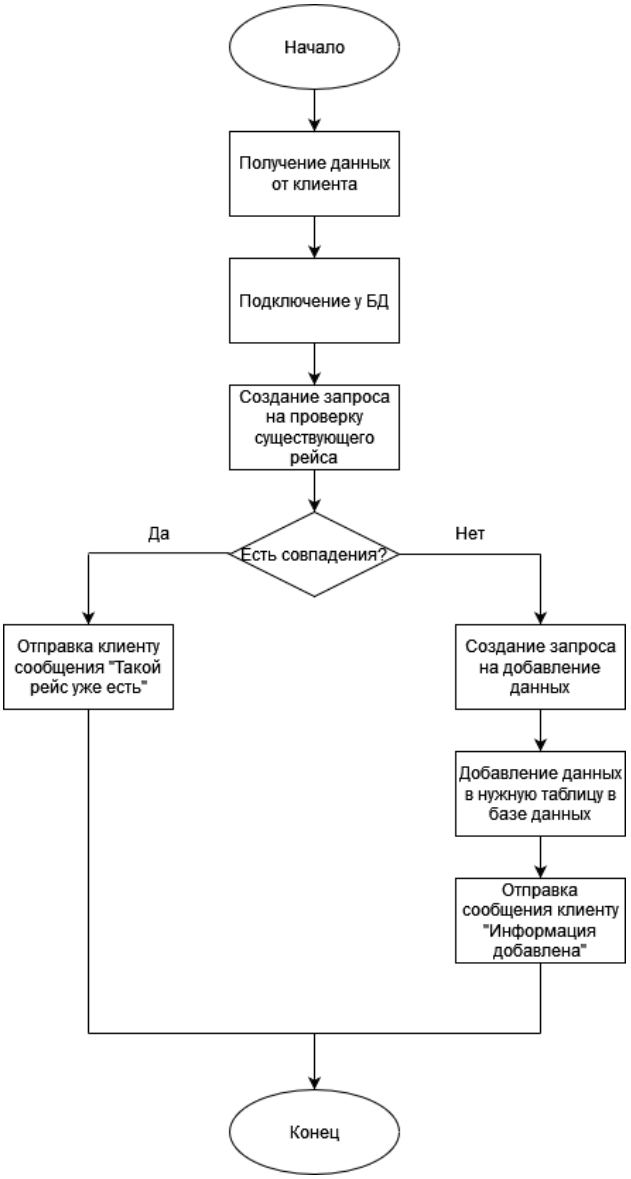

Рисунок 6.4 – Блок-схема добавления информации о рейсе

6.5 Алгоритм авторизации пользователя

Данный алгоритм позволяет проверить введенные пользователем данные для авторизации пользователя.

```
labelMessage.setVisible(false);
User user = new User();
user.setLogin(textfieldLogin.getText());
user.setPassword(passwordfieldPassword.getText());
user.setRole(Roles.USER);
if
(!passwordfieldPassword.getText().equals(passwordfieldConfirmPassword.get
Text())) {
    labelMessage.setText("ошибка");
    labelMessage.setVisible(true);
    return;
}
```

Блок-схема данного алгоритма представлена на рисунке 6.5.

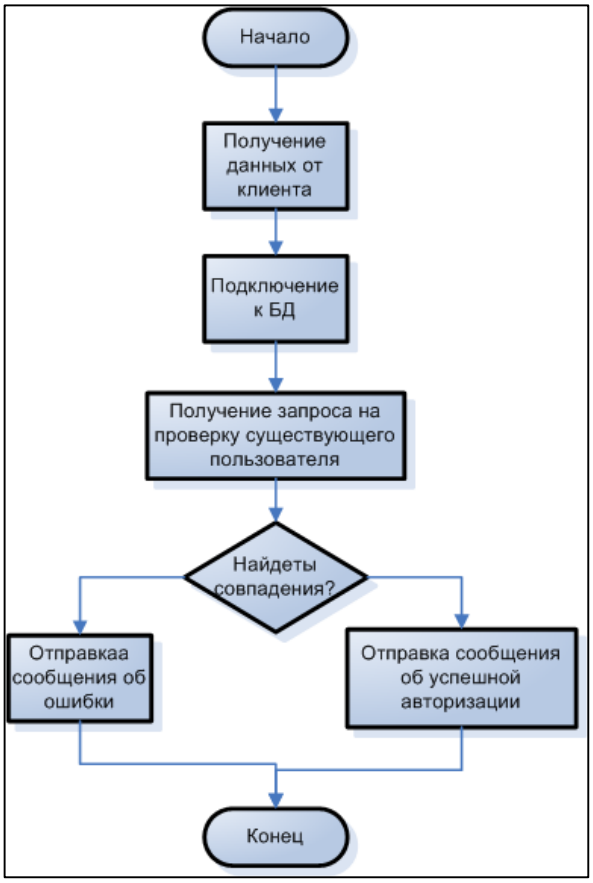

Рисунок 6.5 – Блок-схема авторизации пользователя

Для успешной авторизации пользователя, серверу необходимо получить данные от пользователя, выполнить запрос на поиск данных о пользователе, если результат поиска положительный, отправить клиенту сообщение об успешной авторизации. Если же пользователь не найден в базе, клиенту посылается сообщение об ошибке авторизации.

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данное приложение функционирует в режиме пользователя и администратора. Проект выполнен в архитектуре «клиент-сервер». Чтобы данный проект работал корректно, необходимо запустить два приложения: клиент и сервер.

7.1 Настройка серверной части

Для запуска серверной части необходимо установить jdk 8. А также обязательным является установка DBeaver 21.3.0. В этой базе данных следует выполнить скрипт MyDBScript.sql. После нужно запустить сервер. Приложение сервера будет выглядеть следующим образом (рисунок 7.1).

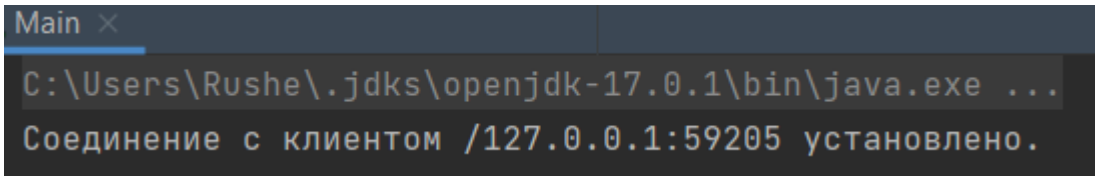

Рисунок 7.1 – Приложение сервера

Сервер работает на порту 5432. Для изменения настроек сервера необходимо внести изменения в файл hebirnate.cfg.xml.

7.2 Настройка клиентской части

Для корректного запуска приложения требуется открыть настройки конфигурации и указать там путь к javafx-sdk-17.0.1:

--module-path "ваш путь" --add-modules javafx.controls,javafx.fxml.

Далее необходимо запустить приложение клиента и мы увидим окно авторизации (Рисунок 7.2).

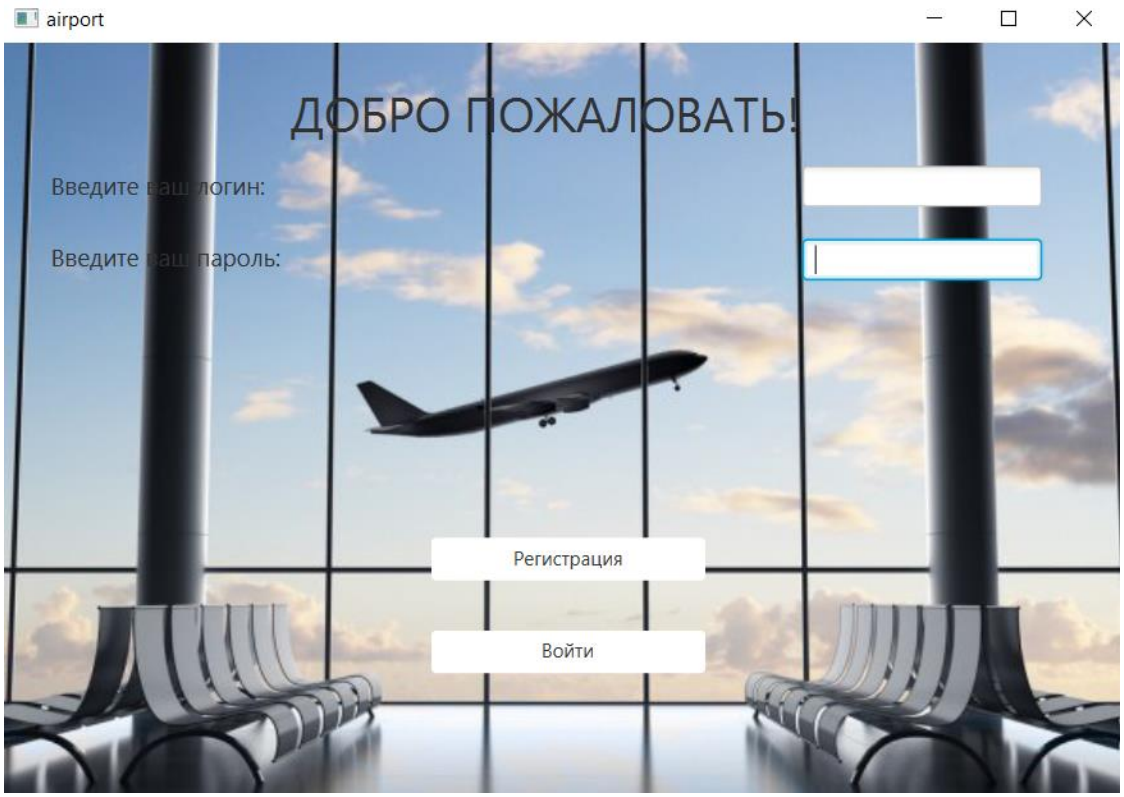

Рисунок 7.2 – Окно авторизации

7.3 Работа в режиме администратора

Чтобы войти под учетной записью администратора, необходимо ввести указанные в базе данных логин и пароль с ролью ADMIN.

После авторизации администратора, появляется меню администратора, с помощью которого он может управлять туристами, зарегистрированными пользователями, рейсами, персоналом и полетами.

На рисунке 7.3 представлено меню администратора.

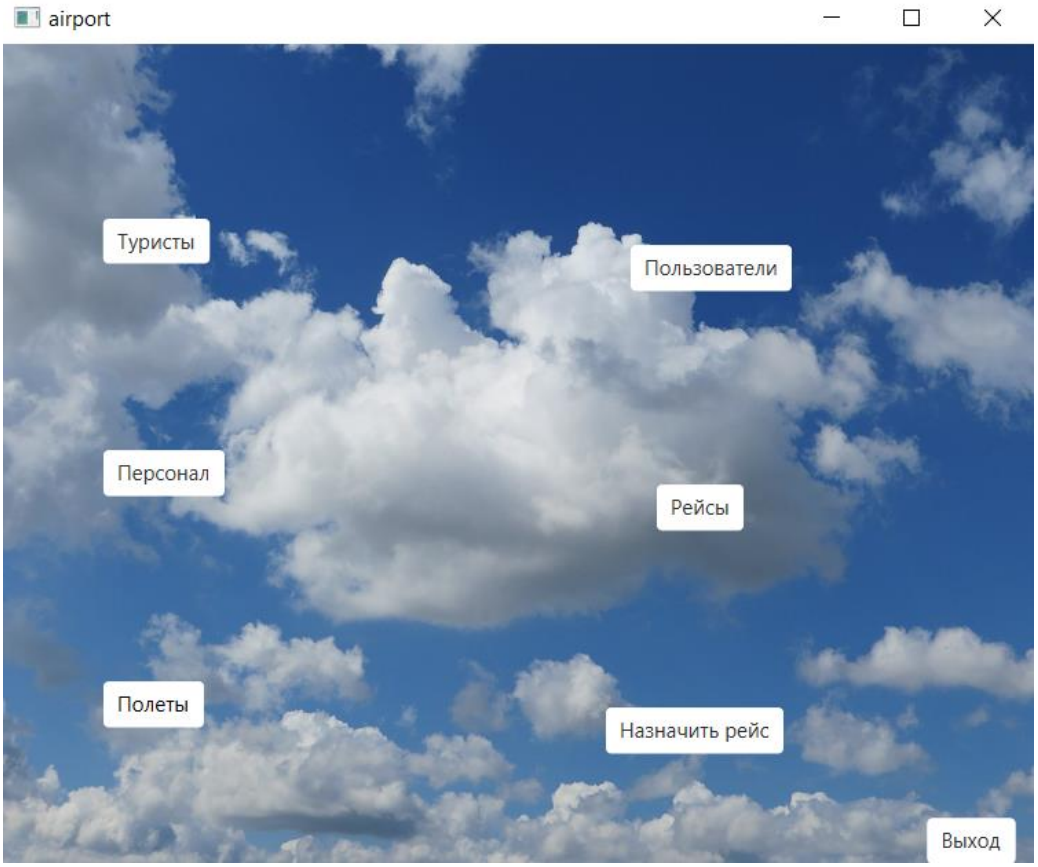

Рисунок 7.3 – Меню администратора

По нажатию на кнопку “Туристы”, появится информация о туристах в табличной форме: id, ФИО, эл.почта, номер телефона. Администратор может добавлять туристов, удалять, редактировать информацию о них (Рисунок 7.4).

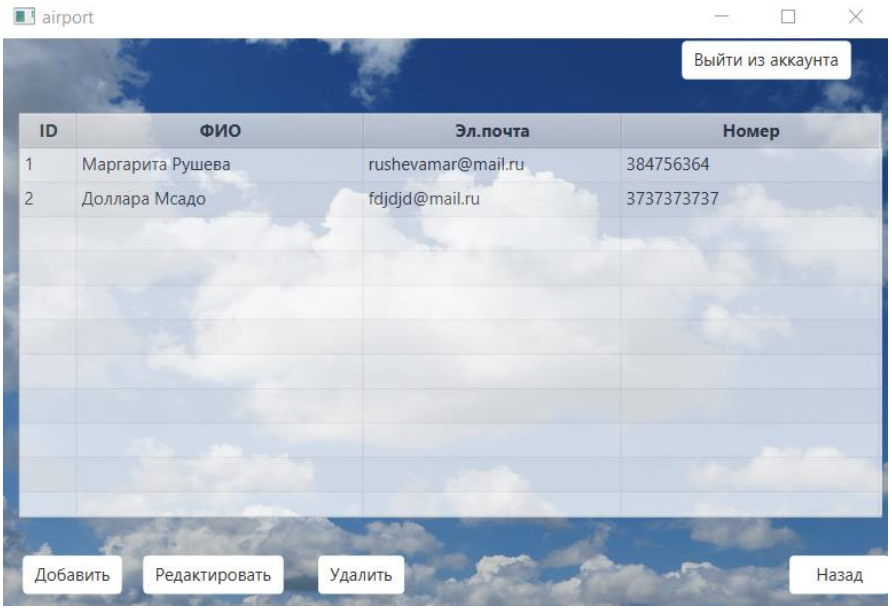

Рисунок 7.4 – Управление информацией о туристах

Чтобы смотреть и редактировать пользователей системы, необходимо нажать кнопку “Пользователи” в меню администратора. Перед нами появится таблица с логинами и паролями зарегистрированных пользователей (Рисунок 7.5).

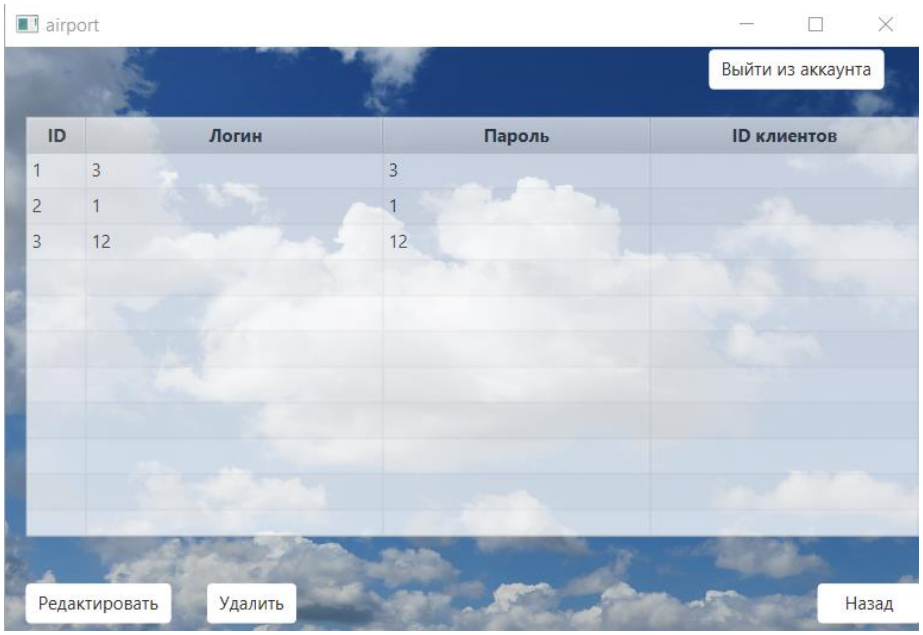

Рисунок 7.5 – Управление информацией о пользователях

Информацию о сотрудниках можно посмотреть, нажав на кнопку меню “Персонал”. Тут мы можем добавлять новых сотрудников, редактировать какую-либо информацию или удалять. В таблице есть поля id, имя, фамилия и рейс. Если какой-либо сотрудник отвечает за какой-либо полет, то в колонке “рейс” это отобразится (Рисунок 7.6).

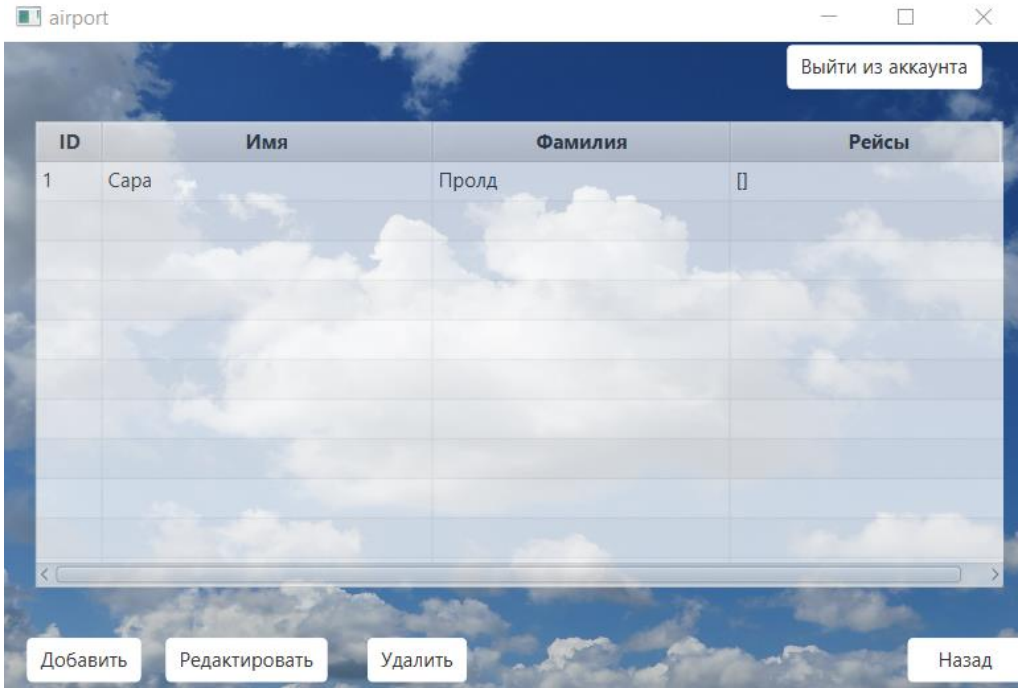

Рисунок 7.6 – Управление информацией о сотрудниках

По нажатию на пункт меню “Рейсы” появится информация о всех возможных рейсах. Мы также можем добавлять/редактировать или удалять (Рисунок 7.7).

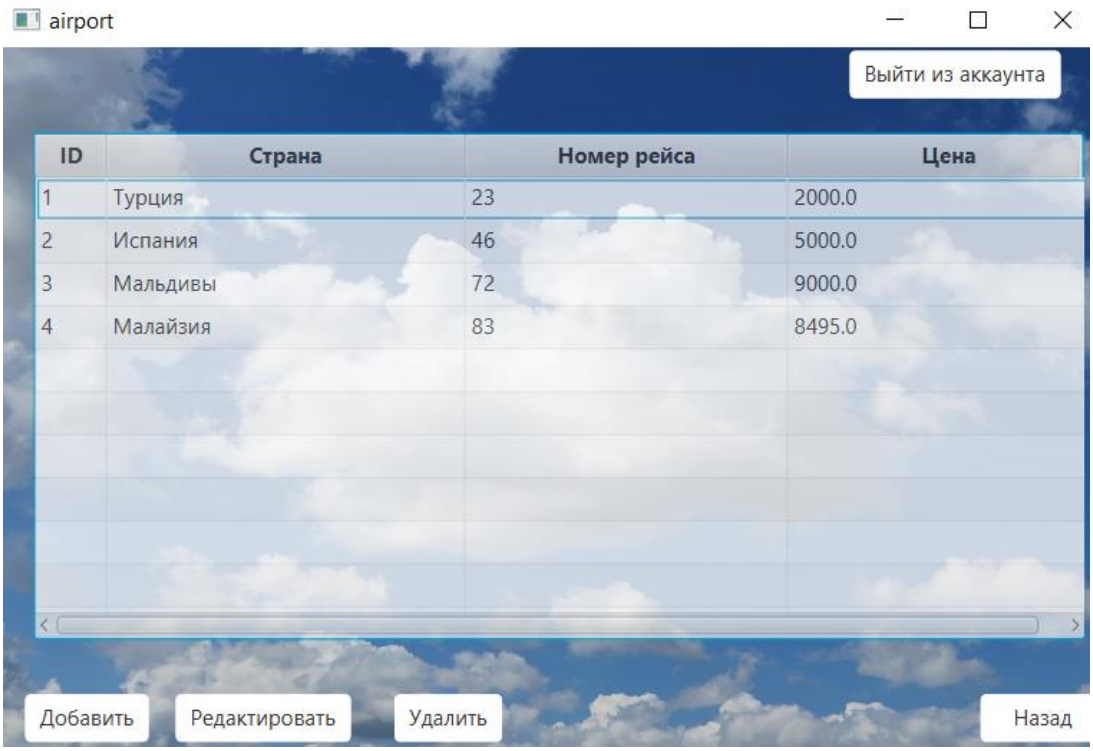

Рисунок 7.7 – Управление информацией о рейсах

Далее мы можем посмотреть информацию о полетах (Рисунок 7.8). Тут мы можем сортировать, просматривать статистику, формировать отчет, фильтровать, искать нужную нам информацию о полете или делать сброс записей. нажав на кнопку формирования отчета, мы получим текстовый файл (Рисунок 7.9).

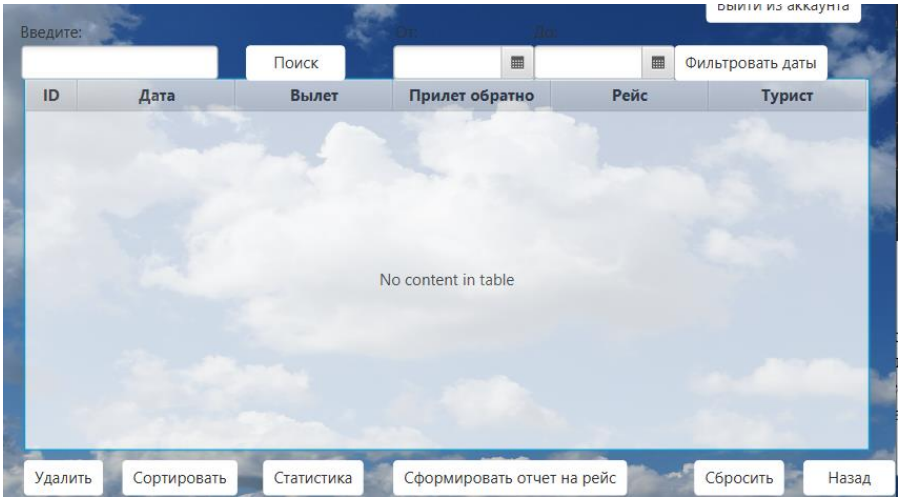

Рисунок 7.8 – Управление полетами (после сброса)

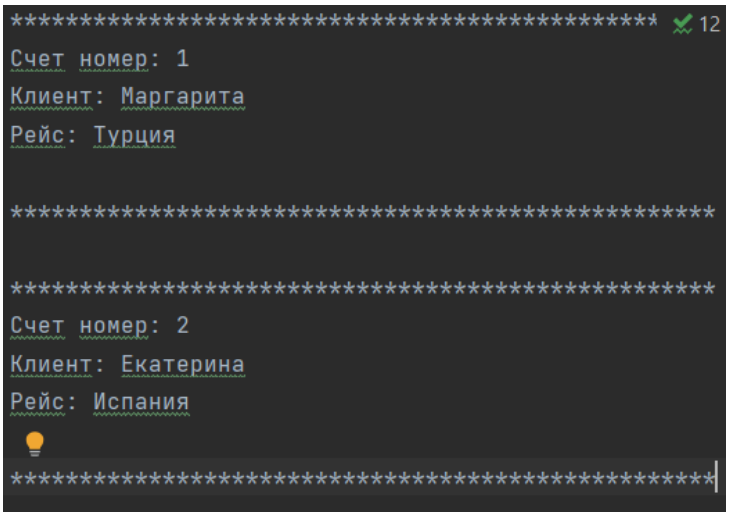

Рисунок 7.9 – Формирование отчета

Так же мы можем назначить сотрудника ответственным за какой-либо рейс. Для этого следует нажать кнопку “Назначить рейс”, выбрать нужного сотрудника и номер рейса, после этого нужно нажать на кнопку “Сохранить”, и данная запись отобразится в базе данных (Рисунок 7.10).

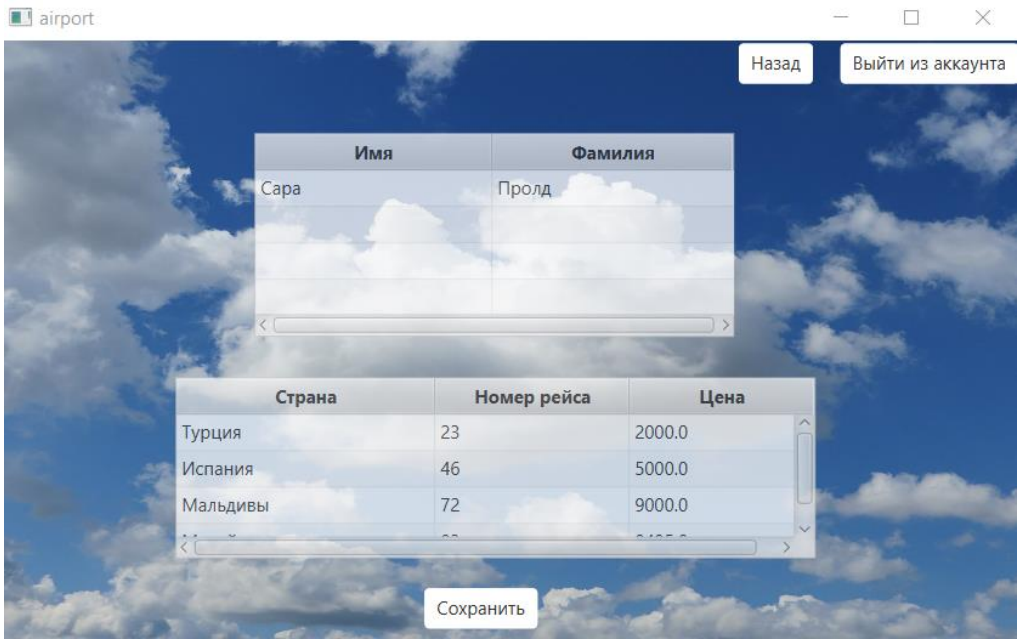

Рисунок 7.10 – Назначение сотрудника ответственным за рейс

7.4 Работа в режиме пользователя

Прежде чем пройти авторизацию, пользователю необходимо зарегистрироваться. Для этого в стартовом окне программы необходимо нажать кнопку «Регистрация», после чего появится соответствующее окно (Рисунок 7.11).

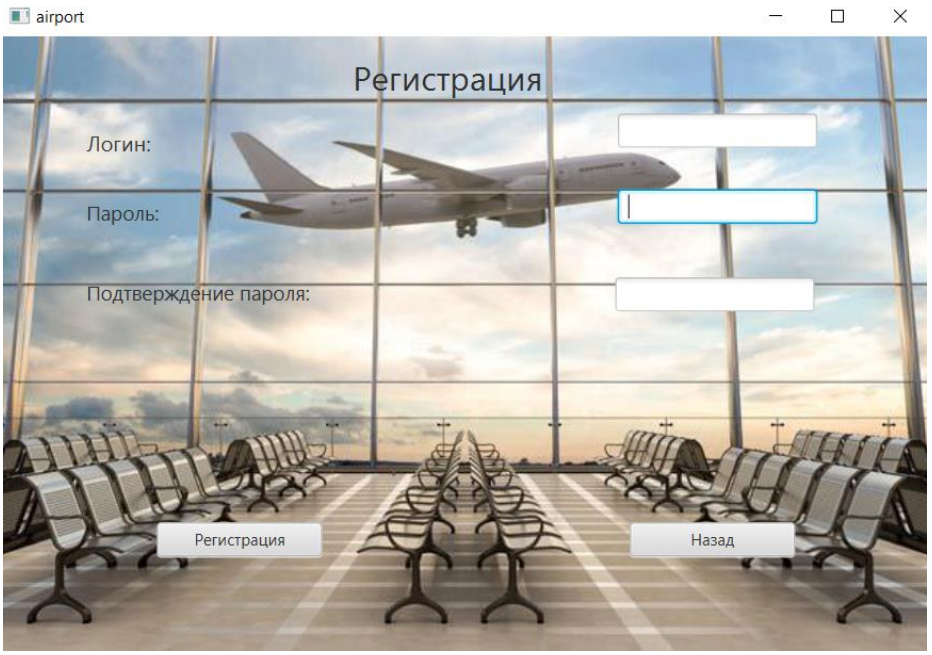

Рисунок 7.11 – Регистрация

После заполнения данными всех полей пользователь нажимает на кнопку «Зарегистрироваться», после чего появляется меню пользователя (Рисунок 7.12).

Рисунок 7.11 – Меню пользователя

У пользователя есть такие пункты меню как изменение темы, просмотр возможных рейсов, выбор рейса, просмотр полетов, управление аккаунтом. После нажатия кнопки “изменить тему”, у пользователя поменяется фон. После нажатия других кнопок в меню, пользователь будет иметь доступ к просмотру рейсов, полетов – это мы рассмотрели в разделе “Работа в режиме администратора”.

8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

После запуска программы появится главное окно программы, при вводе неправильных логина или пароля, снизу на странице появится сообщение ошибки (Рисунок 8.1).

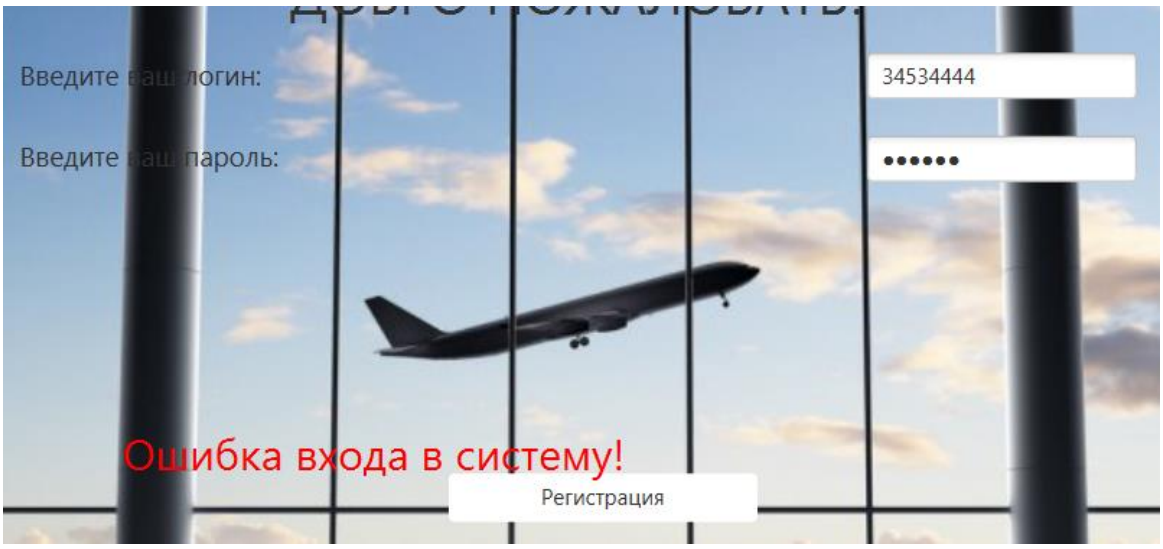

Рисунок 8.1 – Ошибка входа

При входе в систему под учетной записью администратора, мы можем добавлять какую-либо информацию о сотрудниках, туристах, рейсах или пользователях. Попробуем добавить информацию о туристе. Если администратор не заполнит все указанные поля ввода, то появится окошко с сообщением об ошибке (Рисунок 8.2).

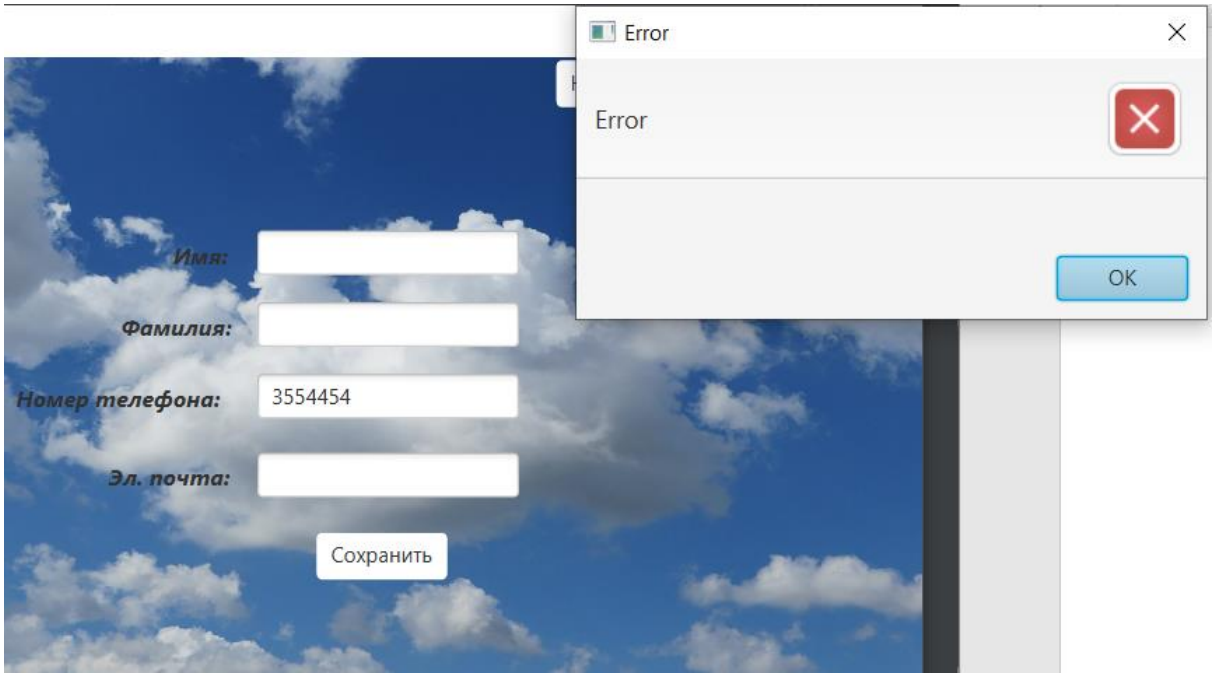

Рисунок 8.2 – Ошибка ввода данных

Также администратор имеет возможность искать записи полетов. Если строка поиска будет пуста, то он также получит сообщение о том, что записей либо нет, либо ничего не найдено (Рисунок 8.3). Такие же сообщения приходят, когда не заполнены все поля фильтрации или, когда недостаточно данных для формирования отчета.

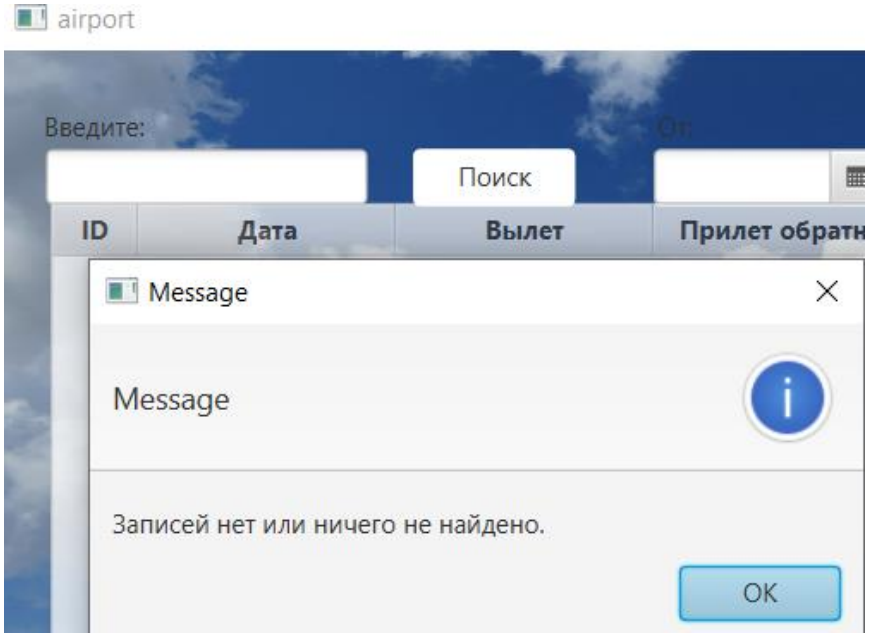

Рисунок 8.3 – Ошибка поиска

При вводе некорректных значений в полях, администратор получит ошибку о том, что поля были заполнены неправильно (Рисунок 8.4). Например, в этом случае, в поле “Цена” и “Рейс” должны быть введены цифры, а не буквы.

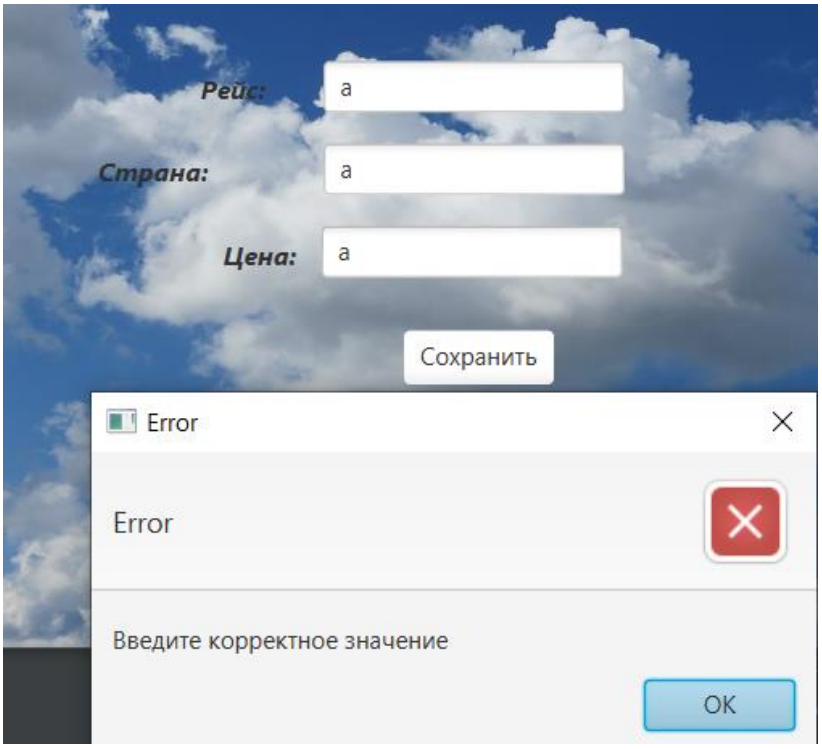

Рисунок 8.4 – Ошибка ввода

При регистрации пользователя действует проверка на ввод пароля. Значения в полях “Пароль” ”Подтверждение пароля” должны быть одинаковыми, иначе пользователь получит ошибку о том, что пароли не совпадают (Рисунок 8.5).

Рисунок 8.5 – Ошибка ввода паролей

ЗАКЛЮЧЕНИЕ

В процессе работы над курсовым проектом было разработано приложение, реализующее простой и удобный подход клиент-серверного взаимодействия. Данное приложение позволяет оптимизировать работу по поиску наиболее приоритетной цели, используя один из методов экспертных оценок, а именно – метод предпочтений.

Приложение представляет собой универсальный инструмент решения данной задачи. Гибкость его заключается в том, что существует возможность самостоятельного указания количества целей и экспертов, а также наименования цели. На клиентской части приложения происходит только ввод данных. Все вычисления происходят на серверной части.

Говоря о предметной области можно сделать вывод, что в рамках нашей страны данная тематика довольно распространена. Поэтому созданное приложение будет актуальным.

Подводя итог, можно сказать, что разработанное приложение может быть использовано в большинстве аэропортов. Поэтому считаю, что цели и задачи курсового проекта были достигнуты и выполнены в полном объеме.

Данное приложение упрощает работу и позволяет сотрудникам аэропорта быстро найти все недостатки своего аэропорта и убрать их. Это сделает работу аэропорта максимально эффективной. С помощью данной программы аэропорта сможет увеличить свою прибыль и количество клиентов. Это является главной целью для аэропорта.

Также были разработаны схемы алгоритмов, что позволяет понять принципы работы функций. Была использована методология функционального моделирования IDEF0, построена стратегическая карта и осуществлено описание организационной структуры аэропорта.

Цель курсового проекта была достигнута. Благодаря хорошей проектировке данной программы в дальнейшем она может легко может быть дополнена и усовершенствована.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Арнольд, К., Гослинг, Дж., Холмс, Д. Язык программирования Java. – 3-е изд. – М. : Вильямс, 2001. – 624 с.

[2] Эккель, Б. Философия Java. – 4-е изд. –СПб. : Питер, 2011. –640 с.

[3] Блох, Д. Java. Эффективное программирование. –М. : Лори, 2002. – 224 с.

[4] Макконнелл, С. Совершенный код. –СПб. : Питер, 2005. –896 с.

[5] Хорстманн, К. С., Корнелл, Г. Библиотека профессионала. Java 2 : Том 1.

Основы. –8-е изд. –М. : Вильямс, 2013. –816 с.

[6] Хорстманн, К. С., Корнелл, Г. Библиотека профессионала. Java 2. : Том 2.

Тонкости программирования. –8-е изд. –М. : Вильямс, 2012. –992 с.

[7] Блинов, И. Н., Романчик, В. С. Java 2. Практическое руководство. – Минск :УниверсалПресс, 2005. – 400 с.

[8] Блинов, И. Н., Романчик, В. С. Java. Промышленное программировние. – Минск : УниверсалПресс, 2007. –704 с.

[9] Тейт, Б. Горький вкус Java – СПб. : Питер, 2003. – 334 с.

[10] Буч, Г., Рамбо, Дж., Джекобсон, А. Язык UML. Руководство пользователя. – М. : ДМК, 2000. – 432 с.

[11] Фаулер, М. UML. Основы. – 3-е изд. – СПб. : Символ-плюс, 2006. – 192 с.

[12] Ларман, К. Применение UML 2.0 и шаблонов пректирования. Введение

в объектно-ориентированный анализ и проектирование. – 3-е изд. – СПб. : Вильямс, 2012. – 736 с.

[13] Стелтинг, С., Маасен, О. Java. Применение шаблонов Java. – М. : Вильямс, 2002. – 576 с.

[14] Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб. : Питер, 2007. – 366 с.

ПРИЛОЖЕНИЕ А
Листинг программного кода

Класс “Main”:

```
import utils.Server;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class Main {
    private static final int PORT_NUMBER = 5556;
    private static ServerSocket serverSocket;
    private static Server clientHandler;
    private static Thread thread;
    private static List<Socket> currentSockets = new ArrayList<>();

    public static void main(String[] args) throws IOException {
        serverSocket = new ServerSocket(PORT_NUMBER);
        while (true) {
            for (Socket socket :
                    currentSockets) {
                if(socket.isClosed()){
                    currentSockets.remove(socket);
                    continue;
                }
                String socketInfo = "Соединение с клиентом " +
socket.getInetAddress() + ":" + socket.getPort() + " установлено.";
                System.out.println(socketInfo);
            }
            Socket socket = serverSocket.accept();
            currentSockets.add(socket);
            clientHandler = new Server(socket);
            thread = new Thread(clientHandler);
            thread.start();
            System.out.flush();
        }
    }

    protected void finalize() throws IOException {
        serverSocket.close();
    }
```

Класс “Login”:

```
import com.google.gson.Gson;
import enums.Requests;
import enums.Responses;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
```

```
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import models.TCP.Request;
import models.TCP.Response;
import models.entities.User;
import utils.ClientSocket;

import java.io.IOException;

public class Login {
    public PasswordField passwordfieldPassword;
    public TextField textfieldLogin;
    public Button buttonRegister;
    public Button buttonLogin;
    public Label labelMessage;

    public void Login_Pressed() throws IOException {
        User user = new User();
        user.setLogin(textfieldLogin.getText());
        user.setPassword(passwordfieldPassword.getText());
        Request requestModel = new Request();
        requestModel.setRequestMessage(new Gson().toJson(user));
        requestModel.setRequestType(Requests.LOGIN);
        ClientSocket.getInstance().getOut().println(new
Gson().toJson(requestModel));
        ClientSocket.getInstance().getOut().flush();
        String answer = ClientSocket.getInstance().getInStream().readLine();
        Response responseModel = new Gson().fromJson(answer, Response.class);
        if (responseModel.getResponseStatus() == Responses.OK) {
            labelMessage.setVisible(false);
            ClientSocket.getInstance().setUser(new
Gson().fromJson(responseModel.getResponseData(), User.class));
            Stage stage = (Stage) buttonLogin.getScene().getWindow();
            Parent root;
 Root= FXMLLoader.load(getClass().getResource("userFXML/account.fxml"));
            Scene newScene = new Scene(root);
            stage.setScene(newScene);
   }
        if (responseModel.getResponseStatus() == Responses.ADMIN_OK) {
            labelMessage.setVisible(false);
            ClientSocket.getInstance().setUser(new
Gson().fromJson(responseModel.getResponseData(), User.class));
            Stage stage = (Stage) buttonLogin.getScene().getWindow();
            Parent root;
            root                                                              =
FXMLLoader.load(getClass().getResource("adminFXML/adminAccount.fxml"));
            Scene newScene = new Scene(root);
            stage.setScene(newScene);
```

```
        }
        else {
            labelMessage.setVisible(true);
        }
    }

    public void Register_Pressed() throws IOException {
        Stage stage = (Stage) buttonRegister.getScene().getWindow();
        Parent root = FXMLLoader.load(getClass().getResource("signUp.fxml"));
        Scene newScene = new Scene(root);
        stage.setScene(newScene);

    }
}
```

Класс “SignUp”:

```
import com.google.gson.Gson;
import enums.Requests;
import enums.Responses;
import enums.Roles;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;
import models.TCP.Request;
import models.TCP.Response;
import models.entities.User;
import utils.ClientSocket;

import java.io.IOException;
public class SignUp {
    public PasswordField passwordfieldPassword;
    public PasswordField passwordfieldConfirmPassword;
    public Button buttonSignUp;
    public Button buttonBack;
    public TextField textfieldLogin;
    public Label labelMessage;

    public void Signup_Pressed() throws IOException {
        labelMessage.setVisible(false);
        User user = new User();
        user.setLogin(textfieldLogin.getText());
        user.setPassword(passwordfieldPassword.getText());
        user.setRole(Roles.USER);
        if
(!passwordfieldPassword.getText().equals(passwordfieldConfirmPassword.getText()
)) {
            labelMessage.setText("Пароли не совпадают");
            labelMessage.setVisible(true);
            return;
```

```
        }
        if (textfieldLogin.equals("") || passwordfieldPassword.equals("") ||
passwordfieldConfirmPassword.equals("")) {
            labelMessage.setText("Не все поля заполнены.");
            labelMessage.setVisible(true);
            return;
        }
        Request request = new Request();
        request.setRequestMessage(new Gson().toJson(user));
        request.setRequestType(Requests.SIGNUP);
        ClientSocket.getInstance().getOut().println(new
Gson().toJson(request));
        ClientSocket.getInstance().getOut().flush();
        String answer = ClientSocket.getInstance().getInStream().readLine();
        Response response = new Gson().fromJson(answer, Response.class);
        if (response.getResponseStatus() == Responses.OK) {
            labelMessage.setVisible(false);
            ClientSocket.getInstance().setUser(new
Gson().fromJson(response.getResponseData(), User.class));
            Stage stage = (Stage) buttonBack.getScene().getWindow();
            Parent root;
            root                                                                =
FXMLLoader.load(getClass().getResource("userFXML/account.fxml"));
            Scene newScene = new Scene(root);
            stage.setScene(newScene);
        } else {
             labelMessage.setText("Пользователь с таким логином уже
существует.");
            labelMessage.setVisible(true);
        }
    }

    public void Back_Pressed() throws IOException {
        Stage stage = (Stage) buttonBack.getScene().getWindow();
        Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));
        Scene newScene = new Scene(root);
        stage.setScene(newScene);
    }
}
```

Класс “AddUpdateClient”:

```
import com.google.gson.Gson;
import enums.Requests;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
```

```
import models.TCP.Request;
import models.TCP.Response;
import models.entities.Client;
import models.entities.User;
import utils.ClientSocket;
import utils.GetService;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

public class AddUpdateClient implements Initializable {
    public TextField nameField;
    public TextField surnameField;
    public TextField phoneField;
    public TextField emailField;
    public Button btnBack;
    public Button btnSave;
    public Button btnLogOut;
    private int clientId;
    private int userId;
     Client client;

    public void onLogOut() throws IOException {
        Stage stage = (Stage) btnLogOut.getScene().getWindow();
        ClientSocket.getInstance().setUser(null);
        Parent                                root                                =
FXMLLoader.load(getClass().getClassLoader().getResource("login.fxml"));
        Scene newScene = new Scene(root);
        stage.setScene(newScene);
    }

    public void onBack() throws IOException {
   Stage stage = (Stage) btnBack.getScene().getWindow();
        Parent                                root                                =
FXMLLoader.load(getClass().getClassLoader().getResource("userFXML/manageAccount
.fxml"));
        Scene newScene = new Scene(root);
        stage.setScene(newScene);
    }

    public void onSave() {
        try {
            User user = new User();
            if (client == null) {
                client = new Client();
            }
            user.setId(ClientSocket.getInstance().getUser().getId());
            client.setName(nameField.getText());
            client.setSurname(surnameField.getText());
            client.setTelephone(phoneField.getText());
            client.setEmail(emailField.getText());
            client.setUser(user);
```

```
            Request request;
            if (ClientSocket.getInstance().getClientId() != -1) {
                client.setId(clientId);
                request     =     new     Request(Requests.UPDATE_CLIENT,     new
Gson().toJson(client));
            } else
                request     =     new     Request(Requests.ADD_CLIENT,     new
Gson().toJson(client));
            ClientSocket.getInstance().getOut().println(new
Gson().toJson(request));
            ClientSocket.getInstance().getOut().flush();
            Response                response                =                new
Gson().fromJson(ClientSocket.getInstance().getInStream().readLine(),
Response.class);
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setContentText(response.getResponseMessage());
            alert.showAndWait();
            ClientSocket.getInstance().setClientId(-1);
            Thread.sleep(1500);
            Stage stage = (Stage) btnSave.getScene().getWindow();
            Parent                              root                              =
FXMLLoader.load(getClass().getResource("userFXML/manageAccount.fxml"));
            Scene newScene = new Scene(root);
            stage.setScene(newScene);

        } catch (Exception e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.showAndWait();
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
    try {
            if (ClientSocket.getInstance().getClientId() != -1) {
                GetService<Client>         flightGetService         =         new
GetService<>(Client.class);
                client   =   flightGetService.GetEntity(Requests.GET_CLIENT,   new
Client(ClientSocket.getInstance().getClientId()));
                clientId = client.getId();
                userId = client.getUser().getId();
                nameField.setText(client.getName());
                surnameField.setText(client.getSurname());
                phoneField.setText(client.getTelephone());
                emailField.setText(client.getEmail());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ПРИЛОЖЕНИЕ Б
Скрипт для генерации базы данных

```
create table if not exists "user"
(
      id serial not null
            constraint users_pk
                  primary key,
      login varchar,
      password varchar,
      role varchar
);

create unique index if not exists users_login_uindex
      on "user" (login);

create table if not exists client
(
      id serial not null
            constraint clients_pk
                  primary key,
      name varchar not null,
      surname varchar,
      telephone varchar,
      email varchar,
      user_id integer
            constraint clients_users_id_fk
                  references "user"
);

create unique index if not exists clients_id_uindex
      on client (id);

create table if not exists employee
(
      id serial not null
            constraint employee_pk
                  primary key,
      name varchar,
      surname varchar
);

create unique index if not exists employee_id_uindex
      on employee (id);
create table if not exists service
(
      id serial not null
            constraint services_pk
                  primary key,
      name varchar,
      price double precision,
      time integer
);
```

```
create unique index if not exists services_id_uindex
      on service (id);

create table if not exists employeeservice
(
      employee_id integer not null
            constraint employeeservice_employee_id_fk
                  references employee
                        on update cascade on delete cascade,
      service_id integer not null
            constraint employeeservice_services_id_fk
                  references service
                        on update cascade on delete cascade,
      id serial not null
            constraint employeeservice_pk
                  primary key
);

create unique index if not exists employeeservice_id_uindex
      on employeeservice (id);

create table if not exists appointment
(
      id serial not null
            constraint appointments_pk
                  primary key,
      client_id integer
            constraint appointments_clients_id_fk
                  references client
                        on update cascade on delete cascade,
      endtime time,
      dat date,
      starttime time,
      empl_serv_id integer
            constraint appointments_employeeservice_id_fk
                  references employeeservice
                        on update cascade on delete cascade
);
create unique index if not exists appointments_id_uindex
      on appointment (id);
```