Министерство образования республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

**Институт информационных технологий**

Специальность: Информационные системы и технологии (в бизнес-менеджменте)

**Пояснительная записка**

**К КУРСОВОМУ ПРОЕКТУ**

по курсу «Программирование сетевых приложений»

Тема: «Система автоматизации работы аэропорта»

Студент-заочник 2 курса

группы № 884371

ФИО: Иванов А.А.

Адрес: г. Минск пр. Победителей 1-4

тел. +375 29 1000203

Минск, 2020

СОДЕРЖАНИЕ

[Введение 6](#_Toc437457256)

[1 Анализ работы аэровокзального комплекса 8](#_Toc437457257)

[1.1 Описание работы аэровокзального комплекса 8](#_Toc437457258)

[1.2Цельразработкиинформационнойсистемы 10](#_Toc437457259)

[2 Постановка задачи и обзор методов ее решения 11](#_Toc437457260)

[2.1 Постановка задачи 11](#_Toc437457261)

[2.2 Обоснование решений по использованию технических и программных средств реализации 12](#_Toc437457262)

[3 Функциональное моделирование на основе стандарта IDEF0 17](#_Toc437457263)

[4 Информационная модель системы и её описание 22](#_Toc437457264)

[5 Модели представления системы и их описание 25](#_Toc437457265)

[5.1 Диаграмма вариантов использования 25](#_Toc437457266)

[5.2 Диаграмма состояний 28](#_Toc437457267)

[5.3 Диаграмма последовательностей 29](#_Toc437457268)

[5.4 Диаграмма классов 30](#_Toc437457269)

[5.5 Диаграмма развертывания и компонентов 33](#_Toc437457270)

[5.6 Паттерны проектирования 34](#_Toc437457271)

[6 Обоснование оригинальных решений по использованию технических средств 38](#_Toc437457272)

[7 Описание алгоритмов реализующих бизнес-логику серверной части системы аэровокзального комплекса аэропорта 39](#_Toc437457273)

[7.1 Алгоритм работы администратора 39](#_Toc437457274)

[7.2 Алгоритм работы кассира 40](#_Toc437457275)

[7.3 Алгоритм работы пользователя 40](#_Toc437457276)

[7.4 Алгоритм добавления информации о самолете 41](#_Toc437457277)

[7.5 Алгоритм авторизации пользователя 43](#_Toc437457278)

[7.6 Алгоритм удаления информации о самолете 44](#_Toc437457279)

[8 Руководство пользователя 46](#_Toc437457280)

[8.1 Настройка серверной части 46](#_Toc437457281)

[8.2 Настройка клиентской части 46](#_Toc437457282)

[8.3 Описание формы авторизации 47](#_Toc437457283)

[8.4 Работа в режиме администратора 48](#_Toc437457284)

[8.5 Работа в режиме пользователя 57](#_Toc437457285)

[8.6 Работа в режиме кассира 61](#_Toc437457286)

[9 Результаты тестирования разработанной системы 65](#_Toc437457287)

[Заключение 67](#_Toc437457288)

[Список использованных источников 68](#_Toc437457289)

[ПРИЛОЖЕНИЕ А (Листинг программного кода) 69](#_Toc437457290)

[ПРИЛОЖЕНИЕ Б (SQL-скрипт для генерации базы данных) 78](#_Toc437457291)

# ВВЕДЕНИЕ

Аэропорты – важнейшие транспортные узлы, «форпосты» внутренних и внешних коммуникаций. Эта деятельность чрезвычайно сложна и многогранна. Ежедневно через аэропорты проходят массы пассажиров и тех, кто встречает или провожает их; дополнительные потоки формирует обслуживающий персонал, сотрудники авиакомпаний и экипажи воздушных судов, арендаторы, водители служебных машин.

Помимо выполнения транспортных функций, в аэропортах осуществляется также пограничный контроль и таможенный досмотр; работа этих транспортных узлов немыслима без обширных информационных систем; наконец, необходим надежный контроль доступа в служебные помещения и режимные зоны аэропортов. Активное внедрение новых автоматизированных технологий идет в аэропортах практически всех стран мира.

Аэровокзальный комплекс аэропорта представляет собой характерный пример системы массового обслуживания, основными элементами которой являются входящие потоки требований и обслуживающие их аппараты. Основными входящими потоками являются вылетающие и прилетевшие пассажиры и их багаж. Каждый из этих потоков проходит определенную последовательность технологических операций обслуживания.

В аэровокзале базируется большинство служб, обслуживающих пассажиров от момента входа на территорию аэропорта до вылета и от момента подачи [трапа](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D1%81%D0%BA%D0%BE%D0%BF%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D1%80%D0%B0%D0%BF) к [самолёту](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%BC%D0%BE%D0%BB%D1%91%D1%82) до покидания аэропорта:

– представительства авиакомпаний;

– служба организации пассажирских перевозок;

– службы безопасности;

– багажная служба;

– службы пограничного, иммиграционного и [таможенного](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BC%D0%BE%D0%B6%D0%BD%D1%8F) контроля;

– различные организации и предприятия, обеспечивающие [отдых](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%B4%D1%8B%D1%85), питание, досуг пассажиров.

Для наиболее эффективного и качественного обслуживания пассажиров необходима автоматизация некоторых и перечисленных служб.

Современные информационные технологии с их стремительно растущим потенциалом и быстро снижающимися издержками открывают большие возможности для новых форм организации труда и занятости в рамках как отдельных корпораций, так и общества в целом. Спектр таких возможностей значительно расширяется.

Увеличились информационные потоки и повысились требования к скорости обработки данных, и теперь уже большинство операций не может быть выполнено вручную, они требуют применения наиболее перспективных компьютерных технологий. И, конечно, обойтись без информационной модели производства в этом случае невозможно.

В данном курсовом проекте была автоматизирована деятельность администратора по составлению и корректированию расписания, введению и списанию воздушных судов, также автоматизирована деятельность кассира по оформлению и возврату билета, а также для пассажиров разработана система заказа билета.

Целью выполнения курсового проекта является повышение качества обслуживания пассажиров путем автоматизации деятельности кассира, администратора и пассажира.

Задачами данного курсового проекта являются:

– изучить предметную область;

– реализовать клиент-серверное приложение;

– связать сервер с базой данных;

– создать оконные формы для удобной работы пользователя;

– сделать приложение простым и понятным в использовании;

– предоставить пользователю аналитическую информацию в виде диаграмм;

– разработать базу данных;

– протестировать полученное приложение.

Преимуществами данной системы являются:

– быстрота обслуживания;

– экономия времени клиента;

– простота в использовании;

– быстрый доступ к постоянно меняющейся информации;

– доступ к информации в любое время суток

# АНАЛИЗ РАБОТЫ АЭРОВОКЗАЛЬНОГО КОМПЛЕКСА

## 1.1 Описание работы аэровокзального комплекса

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](http://formulapoleta.ru/aeroports/interesnuefaktu/153-kakproveryayutbagaj). 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 1.1.1 представлена схема аэровокзального комплекса аэропорта.

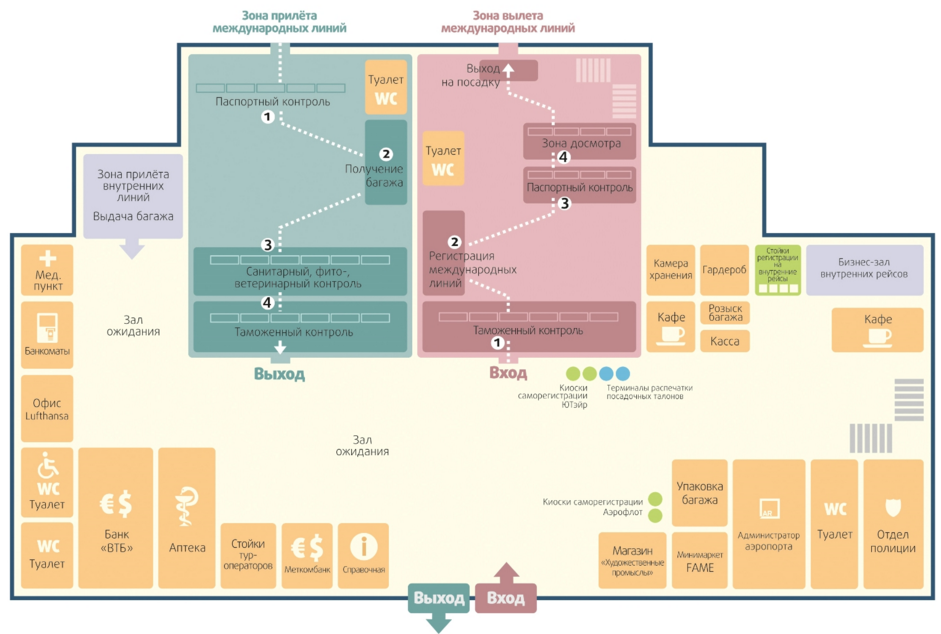


Рисунок 1.1.1 – Схема аэровокзального комплекса аэропорта

Работа современного аэропорта невозможна без автоматизации производственных и бизнес-процессов.

Аэровокзальный комплекс аэропорта отвечает за:

1. Формирование расписания движения самолетов.

Расписание составляется с учетом технического состояния самолетов. Если допуска на полеты самолету нет, то их в расписание ставить нельзя.

2. Введение и списание воздушных судов.

3. Бронирование и продажа билетов пассажирам.

Билеты бронируются и продаются с учетом пункта назначения, пункта вылета, даты вылета и времени вылета. Цена формируется с учетом наценки за класс и время полета.

**1.2 Цель разработки информационной системы**

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 И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

## 2.1 Постановка задачи

Для реализации поставленных целей данного курсового проекта необходимо выполнить следующие задачи:

– изучить предметную область;

– реализовать клиент-серверное приложение;

– связать сервер с базой данных;

– в качестве базы данных использовать MySQL;

– создать оконные формы для удобной работы пользователя;

– сделать приложение простым и понятным в использовании;

– предоставить пользователю аналитическую информацию в виде диаграмм;

– разработать базу данных;

– протестировать полученное приложение.

Разработка системы предусмотрена для трех ролей: администратора, кассира и пользователя.

Со стороны администратора программа должна обеспечивать следующие возможности:

– просматривать расписание актуальных рейсов;

– корректировать расписание;

– удалять расписание;

– создавать расписание;

– просматривать статистику планируемых рейсов;

– вводить в эксплуатацию воздушные суда;

– выводить из эксплуатации воздушные суда;

– просматривать списки пассажиров;

– формировать текстовый отчет, т.е. получать информацию о пассажирах, которые заказали билеты на определенный рейс.

Со стороны кассира программа предоставляет следующие возможности:

– предоставлять расписание актуальных рейсов пассажирам;

– оформлять билет пассажирам;

– осуществлять возврат билета пассажирам;

– формировать статистику планируемых рейсов.

Со стороны пользователя программа обладает следующими возможностями:

– просматривать расписание актуальных рейсов;

– оформлять билет;

– осуществлять регистрацию;

– просматривать информацию о заказанных билетах.

## 2.2 Обоснование решений по использованию технических и программных средств реализации

Для решения поставленных задач в курсовом проекте используются ниже описанные программные и технические средства.

MySQL – [свободная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) [реляционная система управления базами данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94). Разработку и поддержку MySQL осуществляет корпорация [Oracle](https://ru.wikipedia.org/wiki/Oracle).

MySQL является решением для малых и средних приложений. Входит в состав серверов [WAMP](https://ru.wikipedia.org/wiki/WAMP), [AppServ](https://ru.wikipedia.org/w/index.php?title=AppServ&action=edit&redlink=1), [LAMP](https://ru.wikipedia.org/wiki/LAMP) и в портативные сборки серверов [Денвер](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BD%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)), [XAMPP](https://ru.wikipedia.org/wiki/XAMPP), [VertrigoServ](https://ru.wikipedia.org/w/index.php?title=VertrigoServ&action=edit&redlink=1). Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа [MyISAM](https://ru.wikipedia.org/wiki/MyISAM), поддерживающие полнотекстовый поиск, так и таблицы [InnoDB](https://ru.wikipedia.org/wiki/InnoDB), поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Преимуществами MySQL является:

1. Система, основанная на привилегиях и паролях, за счет чего обеспечивается гибкость и безопасность, и с возможностью верификации с удаленного компьютера. Пароли защищены, т.к. они при передаче по сети при соединении с сервером шифруются.

2. Управляет очень большими базами данных. Для каждой таблицы разрешается иметь до 32 индексов. Каждый индекс может содержать от 1 до 16 столбцов или частей столбцов. Максимальная ширина индекса 500 бит (это значение может быть изменено при компиляции MySQL). Для индекса может использоваться префикс поля CHAR илиVARCHAR.

3. Клиенты могут соединяться с MySQL, используя сокеты TCP/IP, сокеты Unix или именованные каналы (named pipes, под NT).

4. Полная поддержка нескольких различных кодировок, включая ISO-8859-1 (Latin1), немецкий, big5, ujis и многие другие. Например, скандинавские символы разрешены в именах таблиц и столбцов.

5. Для хранения всех данных используется выбранный набор символов. Все сравнения для столбцов с нормальными строками проводятся с учетом регистра символов.

BPwin – мощный инструмент моделирования, разработанный фирмой Computer Associates Technologies который используется для анализа, документирования и реорганизации сложных бизнес-процессов. Модель, созданная средствами BPwin, позволяет четко документировать различные аспекты деятельности - действия, которые необходимо предпринять, способы их осуществления, требующиеся для этого ресурсы. Таким образом, формируется целостная картина деятельности предприятия - от моделей организации работы в маленьких отделах до сложных иерархических структур. При разработке или закупке программного обеспечения модели бизнес-процессов служат прекрасным средством документирования потребностей, помогая обеспечить высокую эффективность инвестиций в сферу IT.

Модели BPwin дают основу для осмысления бизнес-процессов и оценки влияния тех или иных событий, а также описывают взаимодействие процессов и потоков информации в организации. Неэффективная, высокозатратная или избыточная деятельность может быть легко выявлена и, следовательно, усовершенствована, изменена или устранена в соответствии с общими целями организации.

Внешние обстоятельства зачастую вынуждают вносить изменения в деятельность организации. Последствия этих изменений должны быть тщательно изучены и осмыслены перед тем, как система будет переделана с их учетом. BPwin может помочь пользователю на протяжении всего цикла, предоставив возможность оптимизировать бизнес-процесс, которого коснутся эти изменения.

С помощью BPwin пользователь может сделать свою работу более продуктивной. Действия и другие объекты создаются буквально несколькими щелчками мыши, а затем легко отбуксированы в нужное место. Интерфейс BPwin, выполненный в стиле "проводника" облегчает навигацию и редактирование сложных процессов с иерархической структурой. Развитые возможности изменения масштаба представления позволяют быстро найти и сосредоточиться на необходимой для работы части модели процесса.

BPwin позволяет:

1. Обеспечить эффективность операций, рассматривая текущие бизнес-операции через мощные инструменты моделирования.
2. Совершенствовать бизнес-процессы, формулируя и определяя альтернативные реакции на воздействия рынка.
3. Быстро исключать непродуктивные операции, легко и интуитивно сопоставляя операционные изменения. Неэффективные, неэкономичные или избыточные операции могут быть легко выявлены и, следовательно, улучшены, изменены или вовсе исключены в соответствии с целями компании.

BPwin (AllFusion Process Modeler) – программный продукт в области реализации средств CASE-технологий. Позволяет проводить описание, анализ и моделирование бизнес-процессов. Занимает одно из лидирующих мест в своём сегменте рынка. В настоящее время выпускается компанией Computer Associates. Распространяется на коммерческой основе.

BPwin автоматизирует решение многих вспомогательных задач, которые обычно связаны с построением модели процесса, и обеспечивает логическую строгость, необходимую для достижения корректных и согласованных результатов. BPwin отслеживает связи в диаграммах, сохраняя их целостность при внесении изменений в модель. Динамическая "подсветка" объектов служит подсказкой при построении модели и предостерегает от повторения распространенных ошибок в моделировании. Кроме этого, BPwin поддерживает заданные пользователем свойства, что позволяет вносить соответствующую вашим потребностям информацию.

Различные варианты оформления с гибким использованием шрифтов, цвета и других средств форматирования придают документам большую наглядность. Пользователь может просматривать и распечатывать общее представление своей модели в виде древовидных диаграмм. С помощью средства создания FEO диаграмм (For Exposition Only) вариации модели или проблемной области можно проанализировать, не внося изменений в основную модель. Возможности настройки пользовательских палитр цветов позволяют легко адаптировать вид документов в соответствии с особенностями принтера или демонстрационного проектора без внесения изменений в саму модель.

Интуитивно-понятный графический интерфейс, который быстро и легко осваивается, что позволяет сосредоточиться на анализе самой предметной области, не отвлекаясь на изучение инструментальных средств. Интерактивное выделение объектов обеспечивает постоянную визуальную обратную связь при построении модели. BРwin поддерживает ссылочную целостность, не допуская определения некорректных связей и гарантируя непротиворечивость отношений между объектами при моделировании.

Для разработки приложения использовалась среда разработки IntelliJ IDEA.

IntelliJ IDEA – [интегрированная среда разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на многих языках программирования, в частности [Java](https://ru.wikipedia.org/wiki/Java), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), [Python](https://ru.wikipedia.org/wiki/Python), разработанная компанией [JetBrains](https://ru.wikipedia.org/wiki/JetBrains).

Преимущества использования IntelliJ IDEA:

1. Умное автодополнение, инструменты для анализа качества кода, удобная навигация, расширенные рефакторинги и форматирование для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML и многих других языков.
2. Поддержка всех популярных фреймворков и платформ, включая Java EE, Spring Framework, Grails, Play Framework, GWT, Struts, Node.js, AngularJS, Android, Flex, AIR Mobile и многих других.
3. Интеграция с серверами приложений, включая Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty и Virgo.
4. Инструменты для работы с базами данных и SQL файлами, включая удобный клиент и редактор для схемы базы данных.
5. Интеграция с коммерческими системами управления версиями Perforce, Team Foundation Server, ClearCase, Visual SourceSafe.
6. Инструменты для запуска тестов и анализа покрытия кода, включая поддержку всех популярных фреймворков для тестирования.

Для создание GUI-форм использовалась библиотека SWING.

Swing – [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для создания [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) для программ на языке [Java](https://ru.wikipedia.org/wiki/Java). Swing был разработан компанией [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems). Он содержит ряд графических [компонентов](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%B0), таких как кнопки, поля ввода, таблицы.

Swing предоставляет более гибкие интерфейсные компоненты, чем более ранняя библиотека [AWT](https://ru.wikipedia.org/wiki/Abstract_Window_Toolkit). В отличие от AWT, компоненты Swing разработаны для одинаковой [кросс-платформенной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) работы, в то время как компоненты AWT повторяют интерфейс исполняемой [платформы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0) без изменений. AWT же использует только стандартные элементы ОС для отображения, то есть для каждого элемента создается отдельный объект ОС (окно), в связи с чем, AWT не позволяет создавать элементы произвольной формы (возможно использовать только прямоугольные компоненты), элементы управления на основе AWT всегда отображаются поверх Swing-элементов (так как все Swing компоненты отображаются на поверхности контейнера).

Компоненты Swing поддерживают специфические динамически [подключаемые виды и поведения](https://ru.wikipedia.org/w/index.php?title=Look_and_Feel&action=edit&redlink=1), благодаря которому возможна адаптация к графическому интерфейсу платформы (то есть к компоненту можно динамически подключить другой, специфический для операционной системы, в том числе и созданный программистом вид и поведение). Таким образом, приложения, использующие Swing, могут выглядеть как родные приложения для данной операционной системы. Положительная сторона – универсальность интерфейса созданных приложений на всех платформах.

Для построения UML-диаграмм использовался Enterprise Architect, который представляет собой мощный UML-инструмент для профессионалов, занимающихся разработкой, тестированием и внедрением программного обеспечения.

Enterprise Architect полностью поддерживает нотацию языка UML 2.1 и позволяет создавать диаграммы всех типов.

# 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.

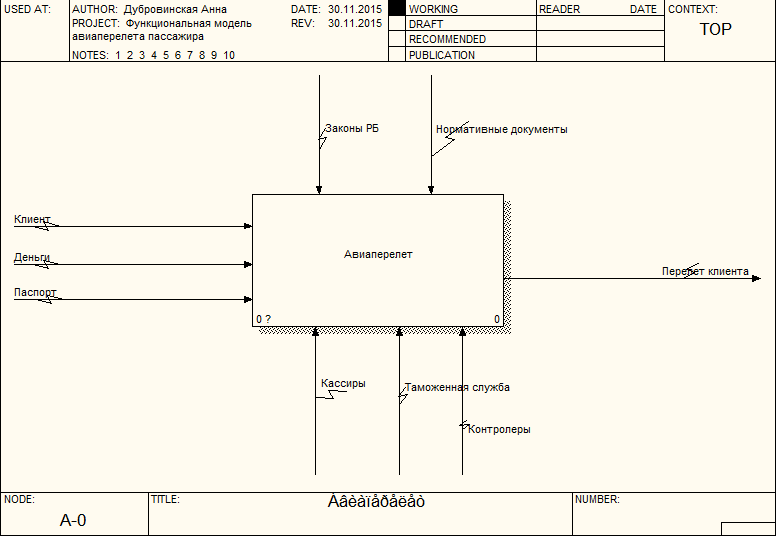


Рисунок 3.1 – Контекстная диаграмма авиаперелета пассажиров

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».

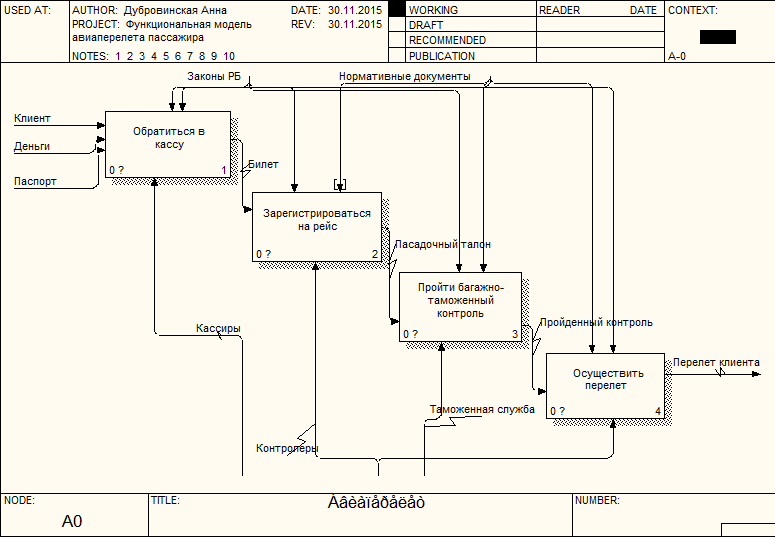


Рисунок 3.2 – Декомпозиция главного блока

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.

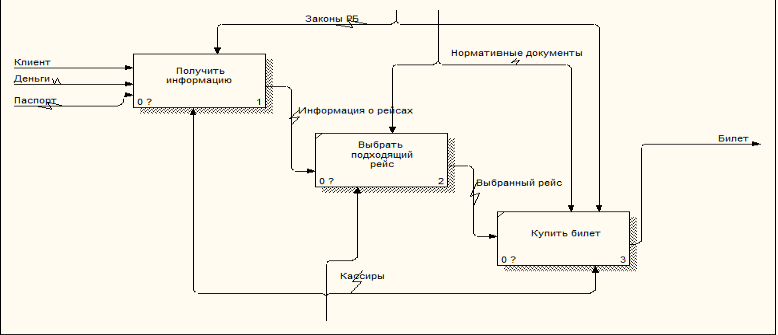


Рисунок 3.3 – Декомпозиция блока «Обратиться в кассу»

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.

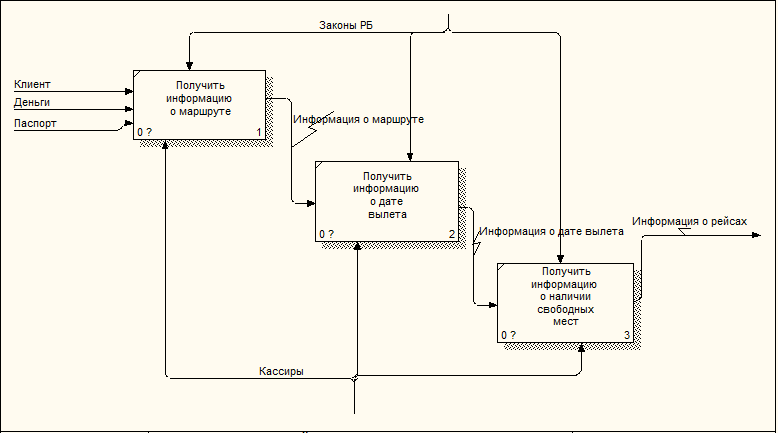


Рисунок 3.4 – Декомпозиция блока «Получить информацию»

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.

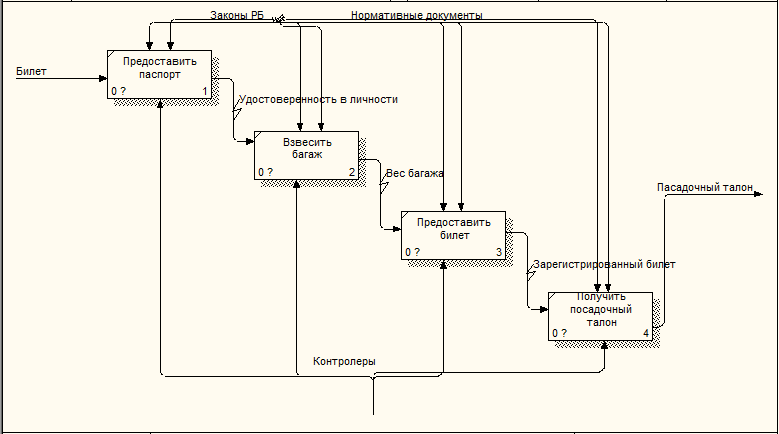


Рисунок 3.5 – Декомпозиция блока «Зарегистрироваться на рейс»

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.

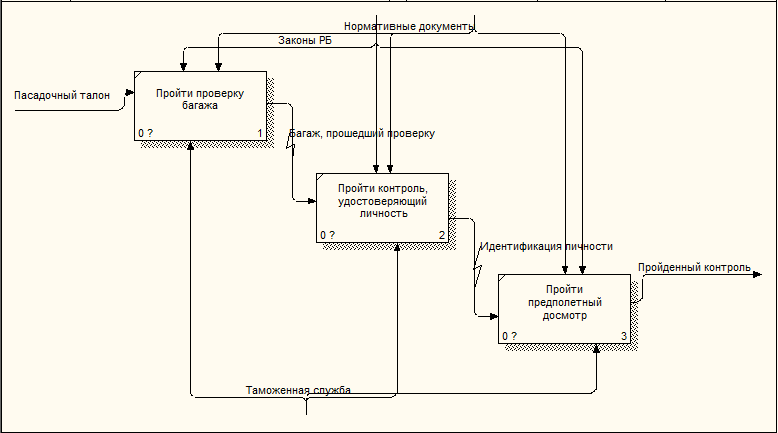


Рисунок 3.6 – Декомпозиция блока «Пройти багажно-таможенный контроль»

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.

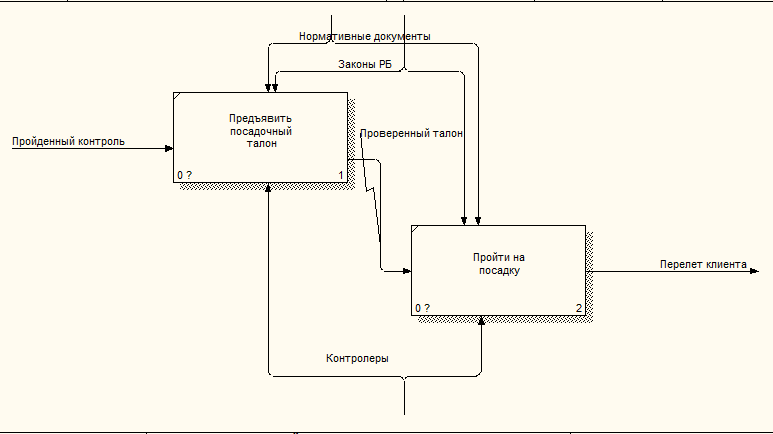


Рисунок 3.7 – Декомпозиция блока «Осуществить перелет»

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.

# ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ

Информационная модель в данном курсовом проекте была построена с помощью средства AllFusion ERwin Data Modeler.

AllFusion ERwin Data Modeler – [CASE](https://ru.wikipedia.org/wiki/CASE)-средство для [проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B1%D0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и [документирования](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F) [баз данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных. [Модели данных](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания.

AllFusion ERwin Data Modeler позволяет наглядно отображать сложные структуры данных. Удобная в использовании графическая среда AllFusion ERwin Data Modeler упрощает разработку базы данных и автоматизирует множество трудоёмких задач, уменьшая сроки создания высококачественных и высокопроизводительных транзакционных баз данных и хранилищ данных.

Сущностями данной модели являются расписание, самолет, маршрут, продажа, билет, пассажир, пользователь, индекс цен.

Сущность «Самолет» содержит информацию о самолетах, а именно бортовой номер, модель, год выпуска, количество посадочных мест. Ключевым полем данной сущности является «Номер самолета».

Сущность «Расписание» содержит информацию о расписании, а именно номер рейса, класс, дата и время вылета. Данная сущность связана с сущностью «Самолет», «Маршрут», «Индекс цен» по номеру самолета, номеру маршрута и номеру индекса цен соответственно. Ключевым полем данной сущности является номер расписания.

Сущность «Индекс цен» содержит информацию об индексе цен, а именно время полета и индекс цен. Ключевым полем данной сущности является номер индекса цен.

Сущность «Маршрут» содержит информацию о маршруте, а именно начальный и конечный пункт, время полета и цену. Ключевым полем данной сущности является номер маршрута.

Сущность «Продажа» содержит информацию о продажах, а именно количество билетов, текущее количество билетов и цена. Сущность «Продажа» связана с сущностью «Расписание» по номеру расписания. Ключевым полем данной сущности является номер продаж.

Сущность «Билет» содержит информацию о билете, а именно номер билета, информацию о пользователе, пассажире и продаже. Сущность «Билет» связана с сущностью «Продажа», «Пользователь» и «Пассажир» по номеру продажи, номеру пользователя и номеру пассажира соответственно. Ключевым полем данной сущности является номер билета

Сущность «Пассажир» содержит информацию о пассажирах, а именно имя, фамилию, отчество, номер и серию паспорта. Ключевым полем данной сущности является номер пассажира.

Сущность «Пользователь» содержит информацию о пользователях, а именно логин, пароль, статус, имя, фамилия, отчество, номер и серия паспорта. Ключевым полем данной сущности является номер пользователя.

Существует два уровня представления информационной модели: логический и физический.

Целью построения логической модели является получение графического представления логической структуры исследуемой предметной области.

Логическая модель предметной области иллюстрирует сущности, а также их взаимоотношения между собой.

Сущности описывают объекты, являющиеся предметом деятельности предметной области, и субъекты, осуществляющие деятельность в рамках предметной области. Свойства объектов и субъектов реального мира описываются с помощью атрибутов.

На рисунке 4.1 представлена логическая модель аэровокзального комплекса аэропорта.

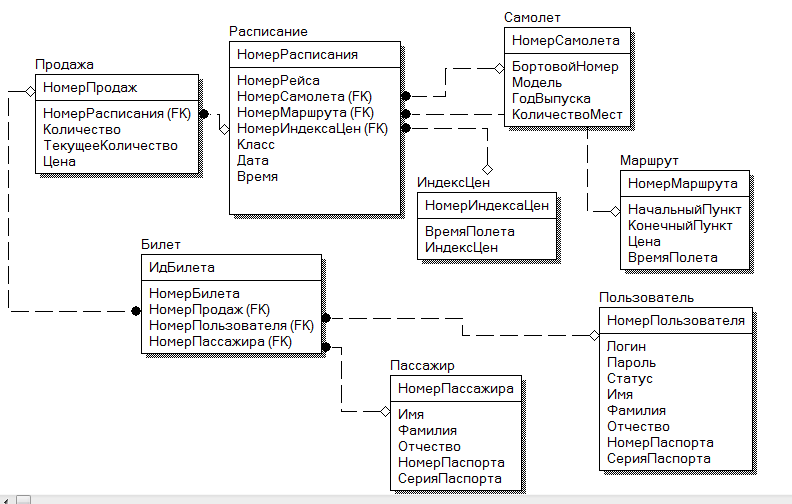


Рисунок 4.1 – Логический уровень преставления системы

Физический уровень модели представляет создание [схемы базы данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) для конкретной [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94). Специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных, создание индексов.

На рисунке 4.2 представлена физическая модель аэровокзального комплекса аэропорта.

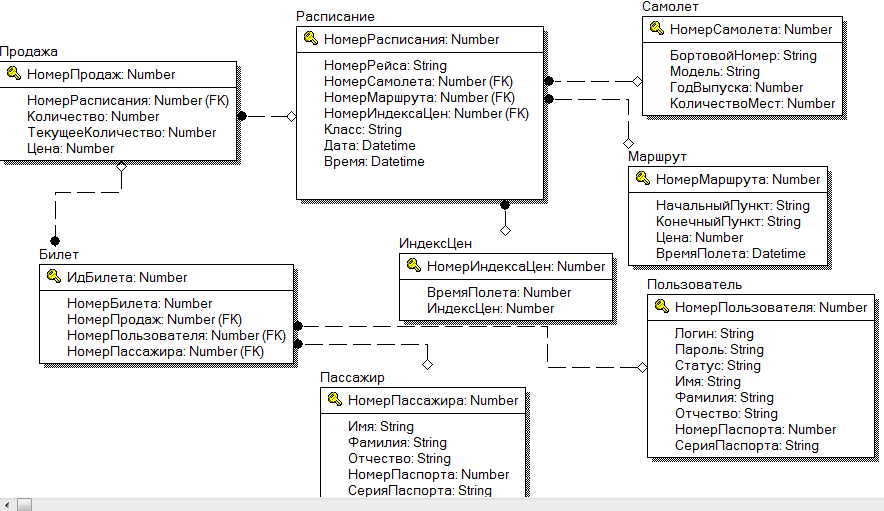


Рисунок 4.2 – Физический уровень представления системы

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.

# МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ

Для описания представления системы используется язык UML.

UML – [язык](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [графического](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) описания для [объектного моделирования](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) в области [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), [моделирования бизнес-процессов](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1), [системного проектирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и отображения [организационных структур](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0).

UML является языком широкого профиля, использующий графические обозначения для создания [абстрактной модели](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), называемой UML-моделью.

Диаграммы UML используются для определения, визуализации, проектирования и документирования программных систем [11].

## Диаграмма вариантов использования

Диаграмма вариантов использования обеспечивает высокоуровневое описание того, что система в состоянии сделать и с кем (или чем) она будет взаимодействовать [11].

На рисунке 5.1.1 представлена диаграмма вариантов использования для кассира и пользователя.

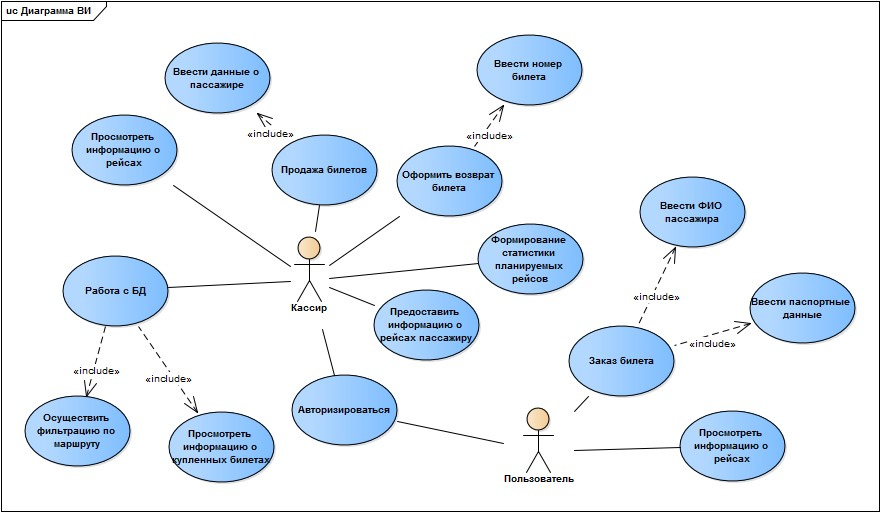


Рисунок 5.1.1 – Диаграмма вариантов использования для кассира и пользователя

На данной диаграмме актерами являются кассир и пользователь.

Работа кассира с БД включает осуществление фильтрации по маршруту. Имеется в виду, что кассир вводит начальный и конечный пункт и может просмотреть результат запроса. Также работа с БД предполагает просмотр информации о купленных билетах, т.е. кассир может ввести номер билета и просмотреть полную информацию по этому билету, которая включает номер рейса, дату и информацию о пассажире.

Просмотр информации о рейсах включает полный список расписания актуальных рейсов.

Продажа билетов предполагает введение кассиром личной и паспортной информации пассажиров и оформление билета на соответствующего пассажира.

Возврат билета предполагает введение кассиром номера билета, просмотр информации о пассажире, на которого оформлен билет и соответственно возврат билета.

Формирование статистики о планируемых рейсах предполагает просмотр кассиром для подтверждения не существования запрашиваемого пункта и заполнения соответствующего поля для формирования статистики.

Пользователь также может просмотреть информацию об актуальных рейсах, а также заказать билет на себя или на другого пассажира, для этого необходимо ввести личные и паспортные данные и нажать соответствующую кнопку.

На рисунке 5.1.2 представлена диаграмма вариантов использования для администратора.

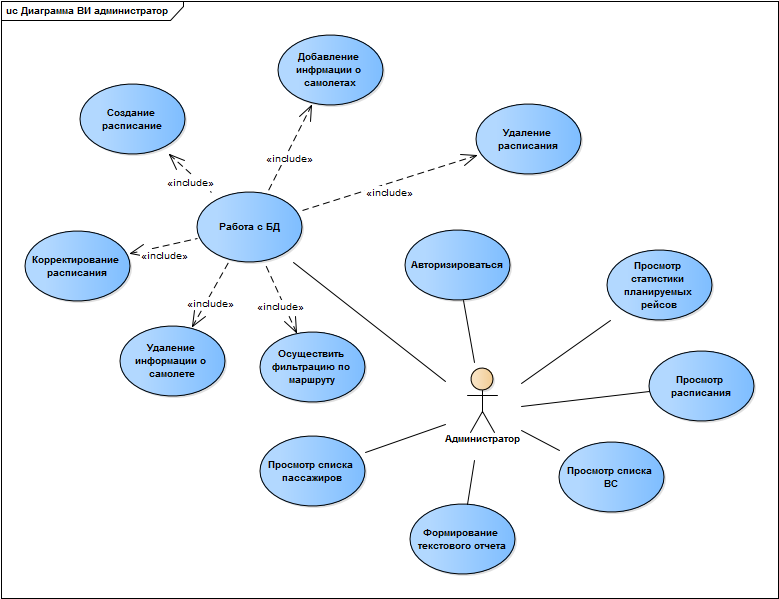


Рисунок 5.1.2 – Диаграмма вариантов использования для администратора

Актером данной диаграммы выступает администратор.

Работа с БД администратора включает удаление расписания, добавление информации о самолетах, создание расписания, корректирование расписания, удаление информации о самолете, осуществление фильтрации по маршруту. При этом следует учесть, что корректирование и удаление расписания запрещено, если началась продажа билетов. Также запрещено ставить самолет на рейс, если в этот день он участвует в другом рейсе.

Просмотр расписания включает просмотр полной информации о рейсе, количестве свободных мест, дате, времени, маршруте.

Просмотр списка пассажиров включает просмотр личных данных пассажира, а именно имени, фамилии и отчества, а также на какое число он заказал билет и соответственно, куда и откуда.

Просмотр списка ВС предполагает просмотр полной информации о самолете, а именно модели, год выпуска, бортовой номер и количество мест.

Формирование текстового отчета подразумевает выгрузку в файл информации о пассажирах для определенного рейса.

Просмотр статистики планируемых рейсов предполагает предоставление администратору аналитической информации в виде диаграммы.

## 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 5.2.1 представлена диаграмма состояний билета.

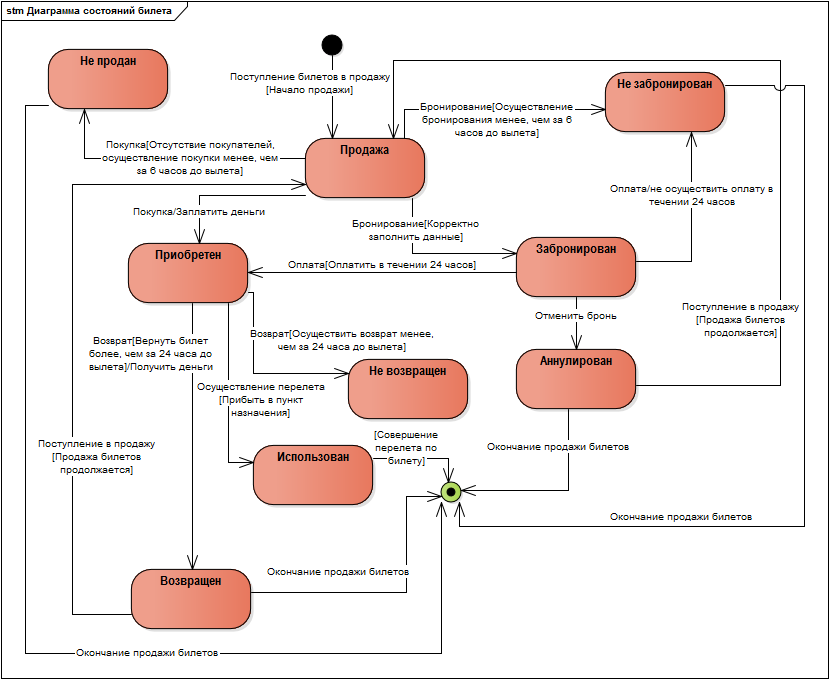


Рисунок 5.2.1 – Диаграмма состояний билета

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».

## Диаграмма последовательностей

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

На рисунке 5.3.1 представлена диаграмма деятельности авторизации пользователя.

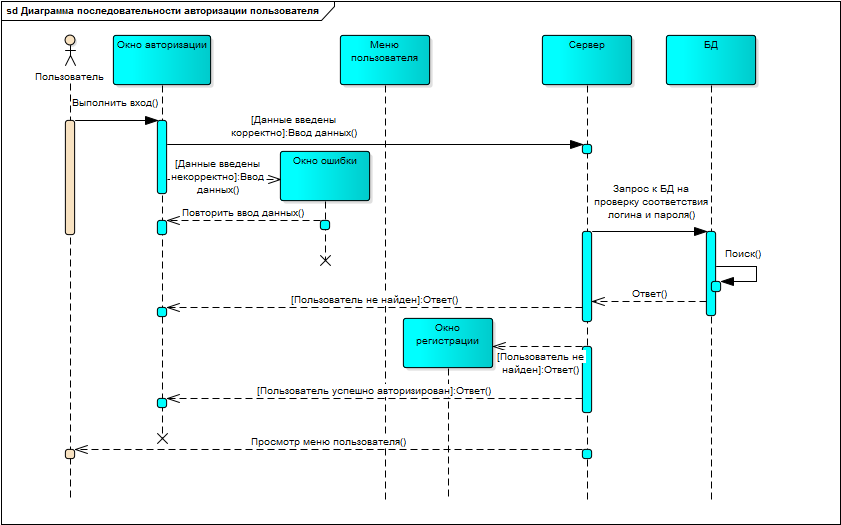
****

Рисунок 5.3.1 – 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.

## Диаграмма классов

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

На рисунке 5.4.1 представлена диаграмма классов пакета entity.

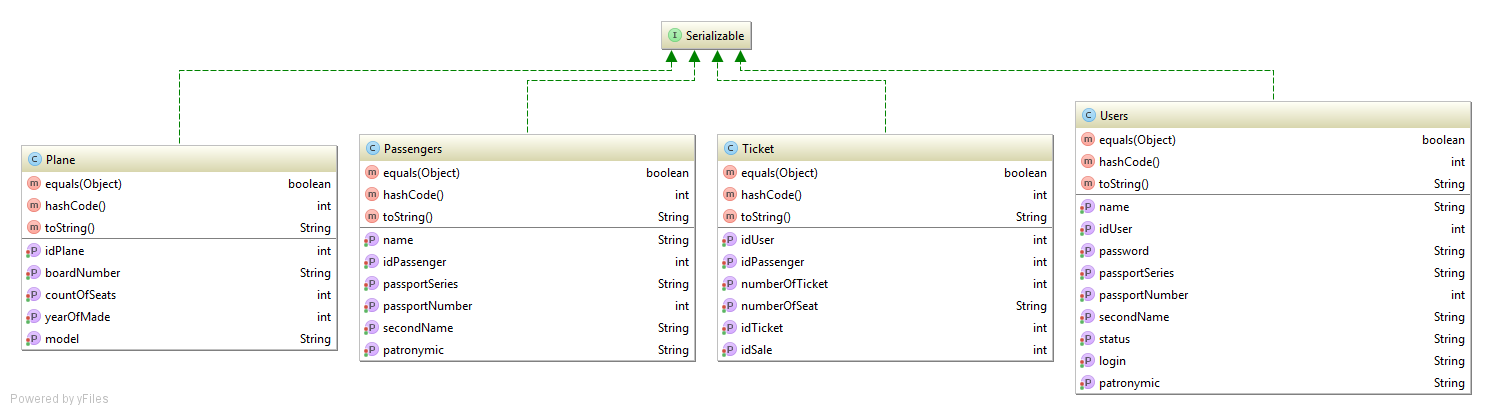


Рисунок 5.4.1 – Диаграмма классов пакета entity

На данной диаграмме классы Plane, Passenger, Ticket и Users реализуют интерфейс Serializable.

На рисунке 5.4.2 представлена диаграмма классов пакета beans.

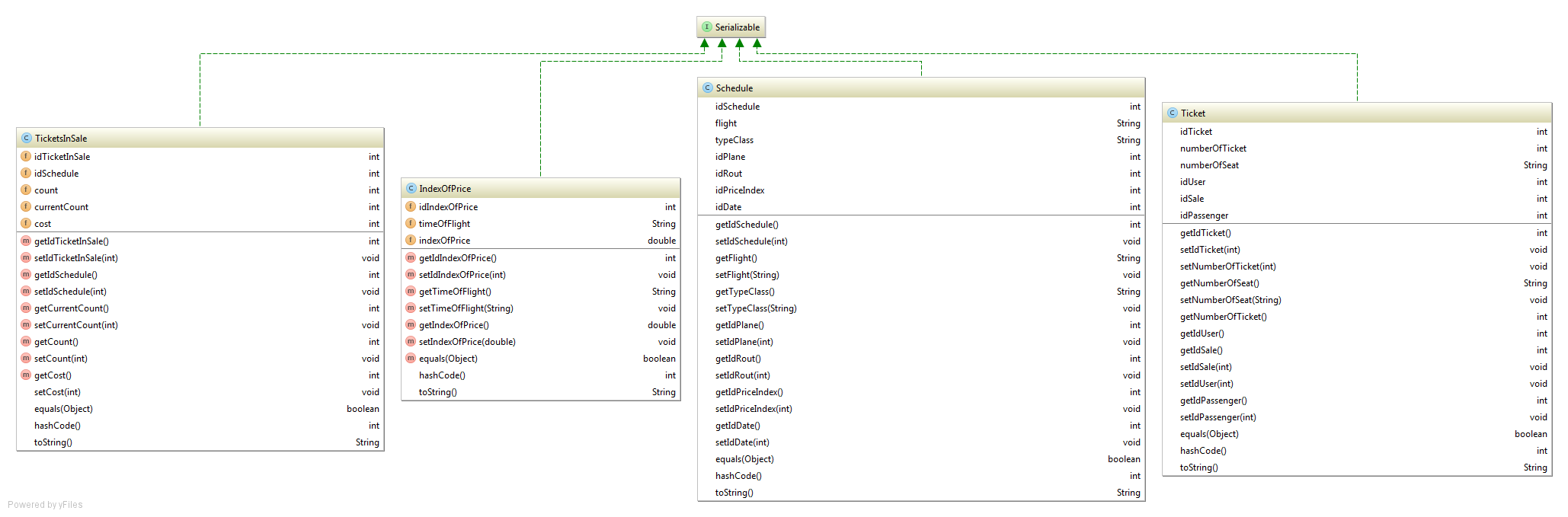


Рисунок 5.4.2 – Диаграмма классов пакета beans

На данной диаграмме классы Schedule, TicketInSale, Rout и TicketInSale реализуют интерфейс Serializable.

На рисунке 5.4.3 представлена диаграмма классов пакетов controller и view.

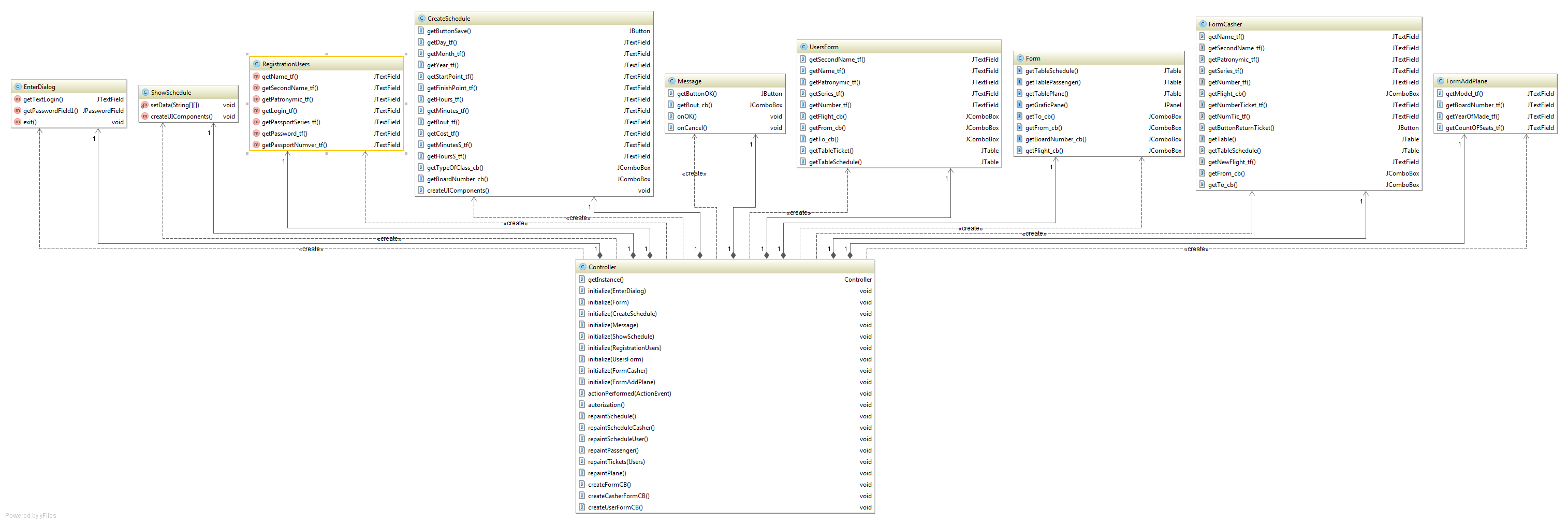


Рисунок 5.4.3 – Диаграмма классов пакета view и controller

## 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 исходный  и исполняемый код.

На рисунке 5.5.1 представлена диаграмма развертывания и компонентов системы.

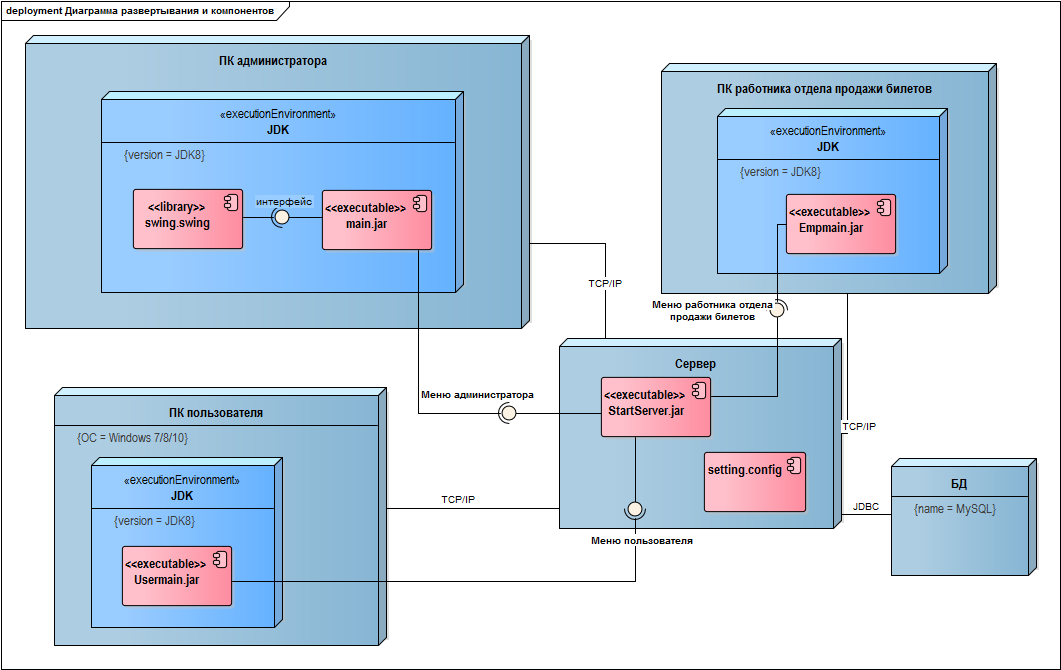


Рисунок 5.5.1 – Диаграмма развертывания и компонентов системы

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.

## Паттерны проектирования

В данном курсовом проекте использовались паттерны MVC, Singleton и Abstract Factory.

MVC разделяет работу приложения на три отдельные функциональные роли: модель данных (model), пользовательский интерфейс (view) и управляющую логику (controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты.

В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента и, например, создавать несколько представлений для одной модели.

Singleton относиться к классу порождающих паттернов. Он используется для создания всего одного экземпляра класса, и гарантирует, что во время работы программы не появиться второй.

В курсовом проекте singleton используется для порождения главного контроллера, создание соединения с базой данных.

На рисунке 5.6.1 представлен паттерн Singleton для создания единственного экземпляра класса соединения с базой данных.

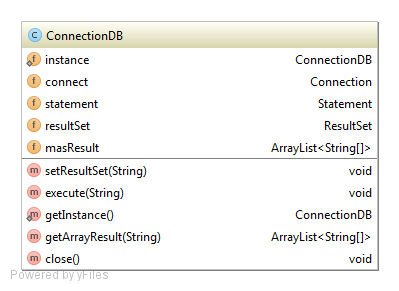


Рисунок 5.6.1 – Реализация паттерна Singleton

Следующим паттерном является абстрактная фабрика.

Абстрактная фабрика – [порождающий шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), позволяющий изменять поведение системы, варьируя создаваемыми [объектами](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), при этом сохраняя [интерфейсы](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы.

На рисунке 5.6.2 представлена диаграмма паттерна Abstract Factory касательно разрабатываемого приложения.

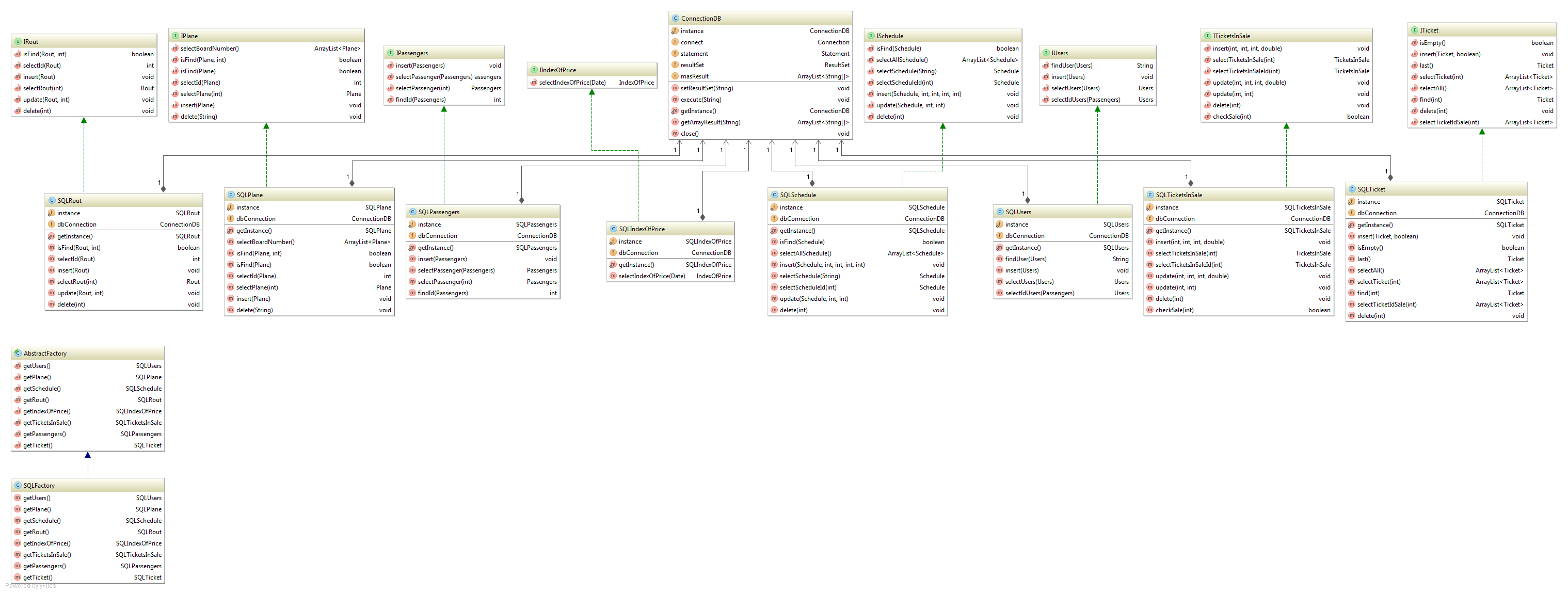


Рисунок 5.6.2 – Реализация паттерна Абстрактная фабрика

На данной диаграмме для выполнения запросов к базе данных представлены следующие классы: SQLSchedule, SQLTicket, SQLPassenger, SQLUser, SQLRout, SQLTicketInSale, SQLIndexOfPrice, SQLPlane.

Для обеспечения гибкости и расширяемости приложения каждый класс реализует соответствующий интерфейс: ISchedule, ITicket, IPassenger, IUser, IRout, ITicketInSale, IIndexOfPrice, IPlane.

Класс SQLFactory реализует интерфейс AbstractFactory и использует обекты классов SQLSchedule, SQLTicket, SQLPassenger, SQLUser, SQLRout, SQLTicketInSale, SQLIndexOfPrice, SQLPlane для последующей работы с ними.

Основными методами классов являются вставка, удаление, обновление и поиск данных. Данные методы используют объекты соответствующих классов: Schedule, Ticket, Passenger, User, Rout, TicketInSale, IndexOfPrice, Plane – для выполнения перечисленных методов.

# ОБОСНОВАНИЕ ОРИГИНАЛЬНЫХ РЕШЕНИЙ ПО ИСПОЛЬЗОВАНИЮ ТЕХНИЧЕСКИХ СРЕДСТВ

Для предоставления пользователю аналитической информации в виде графика потребовалось использовать библиотеку JFreeChart.

JFreeChart – это бесплатная Java библиотека, которая упрощает разработчикам отображать графики профессионального качества в своих приложениях.

Обширный набор функций JFreeChart включает в себя:

– последовательно и хорошо документированный API, поддерживающих широкий спектр типов диаграмм;

– гибкую, хорошо расширяемую конструкцию, как для серверных, так и для клиентских приложений;

– поддержка многих типов выходных форматов, в том числе компонентов Swing, графических файлов (в том числе PNG и JPEG) и векторных форматов графических файлов (включая PDF, EPS и SVG);   
JFreeChart является программным обеспечением с свободно распространяемым программным обеспечением.

На рисунке 6.1 представлен график с использованием средств JFreeChart.

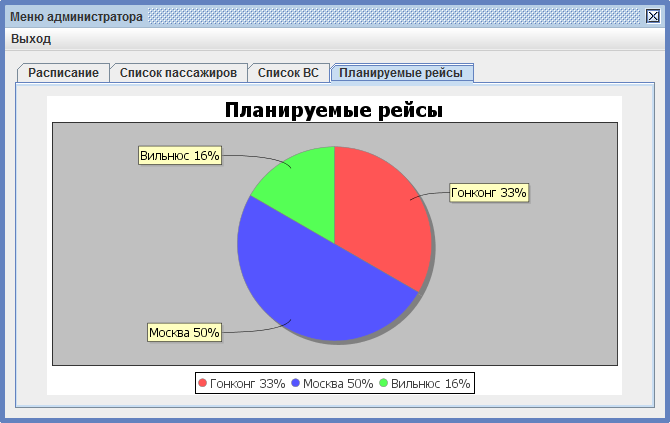


Рисунок 6.1 – Диаграмма планируемых рейсов с помощью средств JFreeChart

# ОПИСАНИЕ АЛГОРИТМОВ РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ АЭРОВОКЗАЛЬНОГО КОМПЛЕКСА АЭРОПОРТА

## 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 7.1.1.



Рисунок 7.1.1 – Блок-схема алгоритма работы администратора

## Алгоритм работы кассира

Данный алгоритм позволяет выполнять кассиру все необходимые ему функции: просматривать текущее расписание рейсов, осуществлять продажу билетов и соответственно возврат билетов, формировать статистику планируемых рейсов.

Блок-схема данного алгоритма представлена на рисунке 7.2.1.

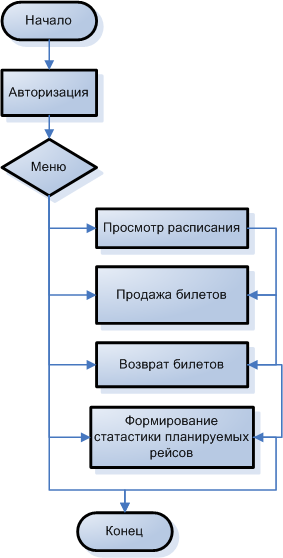


Рисунок 7.2.1 – Блок-схема алгоритма работы кассира

## 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 7.3.1.

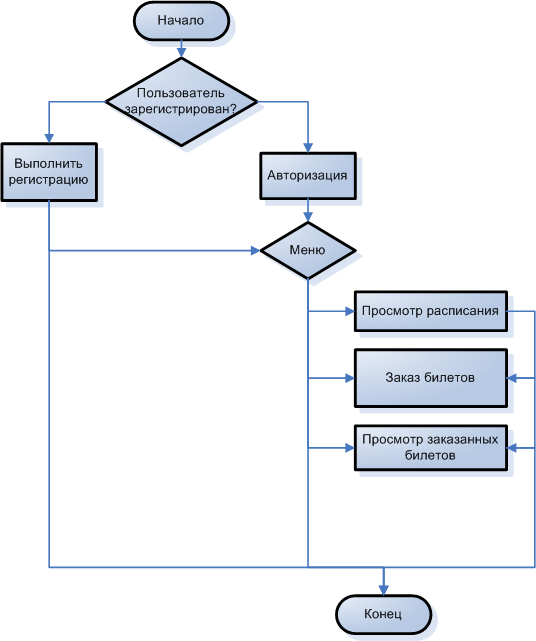


Рисунок 7.3.1 – Блок-схема работы пользователя

## Алгоритм добавления информации о самолете

Данный алгоритм позволяет добавлять информацию о самолетах в базу.

Для добавления информации о самолете 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.

Plane plane = (Plane)sois.readObject();  
 SQLFactory sqlFactory = new SQLFactory();  
 if (sqlFactory.getPlane().isFind(plane)){  
 soos.writeObject("This plane is already existed!");  
 }  
 else{  
 soos.writeObject("OK");  
 sqlFactory.getPlane().insert(plane);  
 }

Блок-схема данного алгоритма представлена на рисунке 7.4.1.

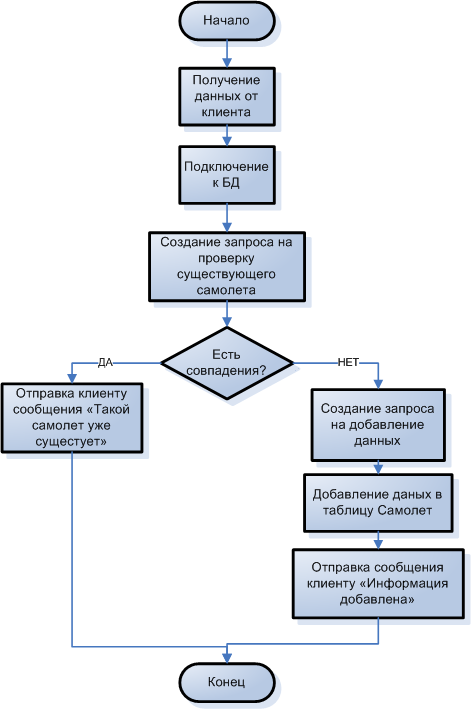


Рисунок 7.4.1 – Блок-схема добавления информации о самолете

## Алгоритм авторизации пользователя

Данный алгоритм позволяет проверить введенные пользователем данные для авторизации пользователя.

Блок-схема данного алгоритма представлена на рисунке 7.5.1.

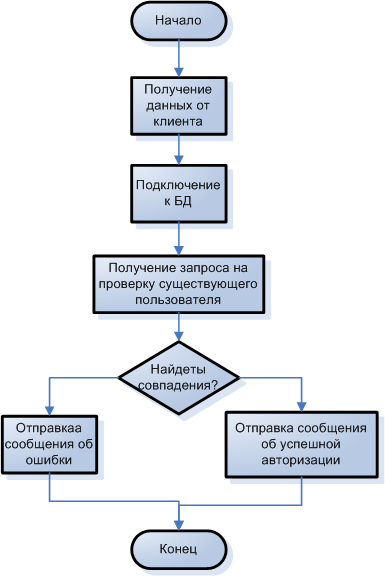


Рисунок 7.5.1 – Блок-схема авторизации пользователя

Users user = (Users)sois.readObject();  
  
SQLFactory sqlFactory = new SQLFactory();  
String status = sqlFactory.getUsers().findUser(user);  
if (status == "")  
 soos.writeObject("error");  
else {  
 soos.writeObject("ok");  
 soos.writeObject(status);  
}

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.

## Алгоритм удаления информации о самолете

Данный алгоритм позволяет удалять информацию о самолетах

Plane plane = (Plane)sois.readObject();

SQLFactory sqlFactory = new SQLFactory();

ArrayList<Schedule> listSchedule = Factory.getSchedule().selectAllSchedule();

int isFind = 0;

for (Schedule schedule: listSchedule){

String boardNumber = sqlFactory.getPlane().selectPlane(schedule.getIdPlane()).getBoardNumber();

if (boardNumber.equals(plane.getBoardNumber())) {

isFind++;

}

}

if (isFind == 0) {

soos.writeObject("OK"); sqlFactory.getPlane().delete(plane.getBoardNumber());

} else {

soos.writeObject("NO");

}

Для удаления информации о самолете сервер получает бортовой номер самолета и проверяет участвует ли самолет в расписании. Если самолет участвует в расписании, сервер посылает клиенту сообщение об ошибке, если же нет, сервер уведомляет клиента об успешном удалении информации и удаляет соответствующую запись.

Блок-схема данного алгоритма представлена на рисунке 7.6.1.

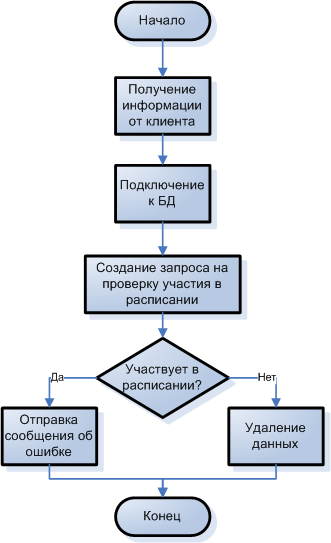


Рисунок 7.6.1 – Блок-схема удаления данных о самолете

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данное приложение функционирует в режиме пользователя, администратора и кассира.

## Настройка серверной части

Для запуска серверной части необходимо установить jdk 8. А также обязательным является установка MySQL 6.3.

Написанное приложение упаковано в jar-архив. Для его запуска необходимо ввести в командной строке java -jar ServerCourseWork.jar server/StartServer предварительно переместившись в каталог, в котором находится jar-архив.

Для удобства пользователя был создан bat-файл для запуска сервера. Таким образом, для запуска сервера необходимо открыть файл start.bat. Появится консоль, в которой отображается статистика подключенных клиентов (рисунок 8.1.1).

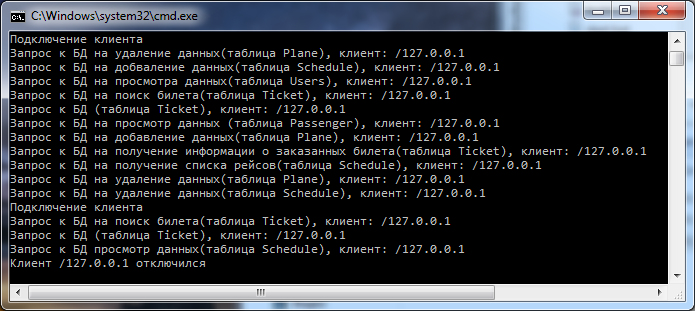


Рисунок 8.1.1 – Статистика подключенных пользователей

Сервер работает на порту 8006. Для изменения настроек сервера необходимо внести изменения в файл setting.properties.

## Настройка клиентской части

Для запуска клиентской части необходимо установить jdk 8.

Написанное приложение упаковано в jar-архив. Для его запуска необходимо ввести в командной строке java -jar CoutseWork.jar mode/client/StartDo предварительно переместившись в каталог, в котором находится jar-архив.

Для удобства пользователя был создан bat-файл для запуска клиента. Таким образом, для запуска клиента необходимо открыть файл start.bat. Чтобы обеспечить взаимодействия клиента и сервера, необходимо для клиента установить рабочий порт 8006.

## 8.3 Описание формы авторизации

Чтобы нормально работать в каждом из режимов, пользователям необходимо выполнить авторизацию. Для этого в окне авторизации необходимо ввести логин и пароль пользователя. На рисунке 8.3.1 представлен пример входа пользователя.

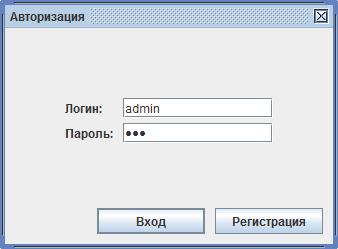


Рисунок 8.3.1 – Пример авторизации пользователя

Если такой пользователь не зарегистрирован, программа выдаст соответствующее сообщение (рисунок 8.3.2).

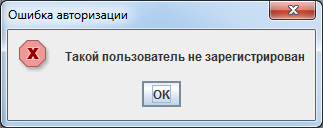


Рисунок 8.3.2 – Реакция программы на отсутствие пользователя в базе

Если данные введены корректно, появляется меню пользователя, с помощью которого он может осуществлять все необходимые ему функции.

## 8.4 Работа в режиме администратора

После авторизации администратора, появляется меню администратора, с помощью которого он может просматривать, удалять, создавать, корректировать, искать расписание, просматривать список пассажиров, просматривать статистику планируемых рейсов, просматривать и удалять информацию о самолетах, формировать текстовый отчет.

На рисунке 8.4.1 представлено меню администратора.

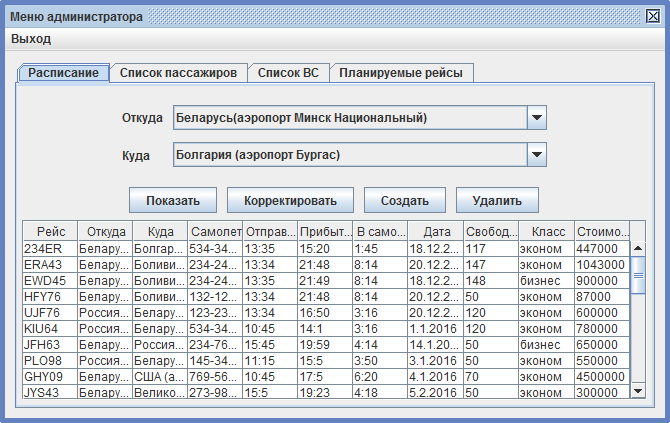


Рисунок 8.4.1 – Меню администратора

Для просмотра расписания по начальному и конечному пункту необходимо выбрать из выпадающего списка соответственно начальный и конечный пункт и нажать кнопку «Показать». Результат поиска представлен на рисунке 8.4.2.

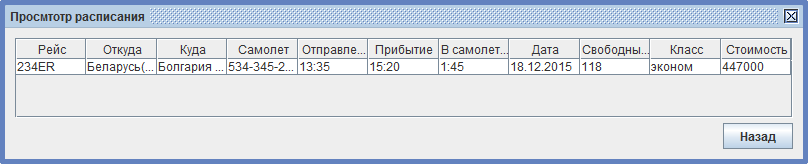


Рисунок 8.4.2 – Результат поиска

Для корректирования расписания необходимо выбрать номер рейса, для которого будет корректировано расписание, при этом необходимо учесть, что корректировать расписание можно лишь в том случае, когда продажа билетов еще не началась. Если не учесть этот факт, программа выдаст соответствующее сообщение (рисунок 8.4.3).

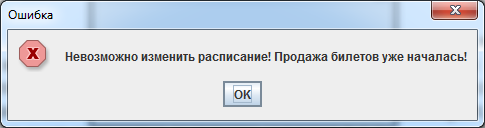


Рисунок 8.4.3 – Реакция программы на попытку корректировки расписания, когда продажа билетов началась

Если же продажа билетов на рейс не началась, то появится следующее окно (рисунок 8.4.4).

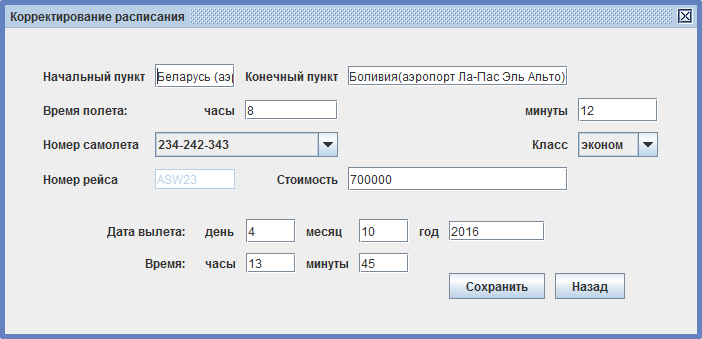


Рисунок 8.4.4 – Корректирование расписания

В данном окне можно изменить все, кроме номера рейса, администратор корректирует необходимые данные, после чего нажимает на кнопку «Сохранить», в случае корректно введенных данных изменения сохраняются в базе данных и программа выдает соответствующее сообщение (рисунок 8.4.5).

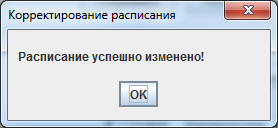


Рисунок 8.4.5 – Реакция программы на успешное изменение расписания

Если же администратор попытается изменить расписание на уже существующее, т.е. начальный и конечный пункт, дата и время вылета совпадают, программа выдаст соответствующее сообщение (рисунок 8.4.6) и изменения в базе данных сохранены не будут.

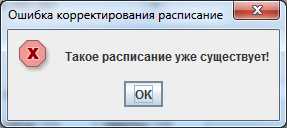


Рисунок 8.4.6 – Реакция программы на попытку изменения расписания на уже существующее

Для создания расписания необходимо нажать на кнопку «Создать». Появится соответствующее окно (рисунок 8.4.7).

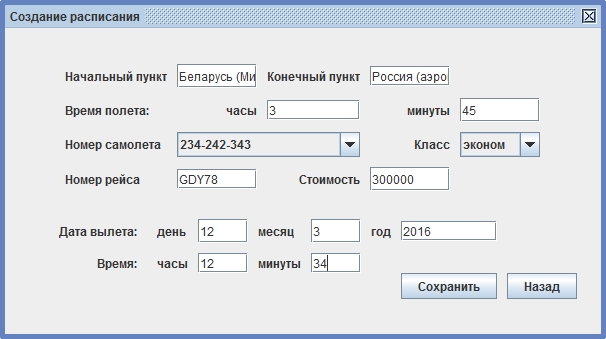


Рисунок 8.4.7 – Создание расписания

В данном окне пользователю необходимо ввести данные о маршруте, времени и дате вылета, а также выбрать какой самолет будет осуществлять рейс.

При этом необходимо учесть, что время и дата должны быть целыми числами и удовлетворять реальному времени, иначе программа выдаст соответствующее сообщение (рисунок 8.4.8).

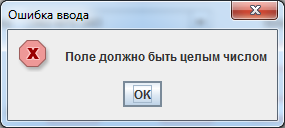


Рисунок 8.4.8 – Реакция программы на неправильно введенные данные

При этом данные, которые введены некорректно будут подсвечены красным цветом (рисунок 8.4.9).

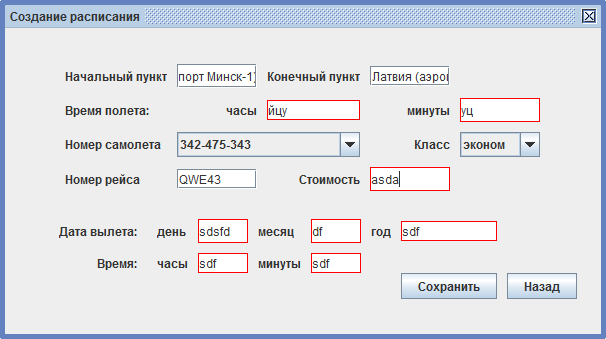


Рисунок 8.4.9 – Реакция программы на некорректно введенные данные

После корректно введенных данных, администратор нажимает «Сохранить», данные сохраняются в базе и программа выдает соответствующее сообщение (рисунок 8.4.10).

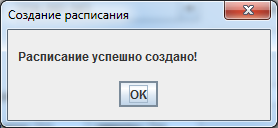


Рисунок 8.4.10 – Реакция программы на успешное создание расписания

Для удаления расписания необходимо нажать кнопку «Удалить» и выбрать из выпадающего списка номер рейса (рисунок 8.4.11).

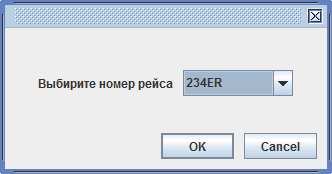


Рисунок 8.4.11 – Выбор номера рейса для удаления расписания

Необходимо учесть, что удалять можно только то расписание, продажа билетов для которого еще не началась. При попытке удалить расписание, для которого началась продажа билетов, программа выдаст соответствующее сообщение (рисунок 8.4.12).

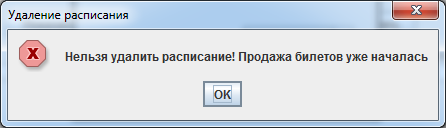


Рисунок 8.4.12 – Реакция программы на попытку удалить расписание для которого началась продажа билетов

В случае успешного удаления расписания программа выдаст соответствующее сообщение (рисунок 8.4.13).

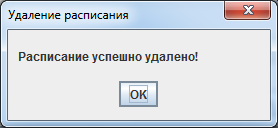


Рисунок 8.4.13 – Удаление расписание

Также администратор может формировать отчет по номеру рейса для получения информации о пассажирах, которые купили билет на данный рейс.

Текстовый отчет представлен на рисунке 8.4.14.

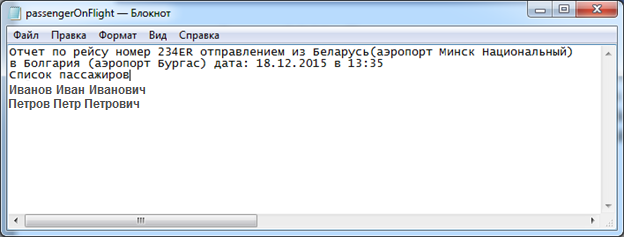


Рисунок 8.4.14 – Текстовый отчет по номеру рейса

Для просмотра списка самолетов, необходимо перейти на вкладку «Список ВС». После перехода по вкладке появится соответствующее меню (рисунок 8.4.15).

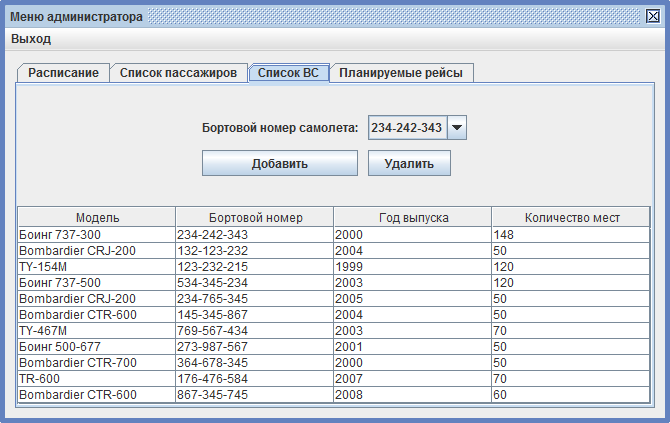


Рисунок 8.4.15 – Просмотр списка ВС

На данной вкладке администратор может добавлять и удалять информацию о самолетах, а также в таблице просматривать полностью информацию о всех имеющихся самолетах.

Для добавления информации о самолете необходимо нажать на кнопку «Добавить», после чего появится соответствующее окно (рисунок 8.4.16).

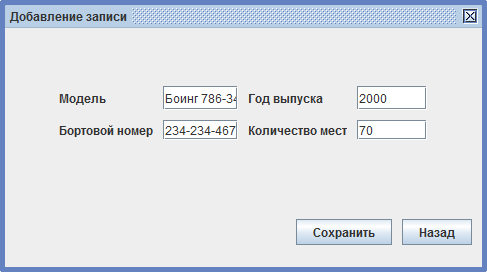


Рисунок 8.4.16 – Добавление информации о самолете

После заполнения полей данными и нажатия на кнопку «Сохранить» программа выдаст соответствующее сообщение (рисунок 8.4.17), а данные будут добавлены в базу и отобразятся в таблице.

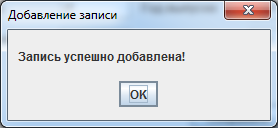


Рисунок 8.4.17 – Реакция программы на успешное добавление записи

При удалении записи необходимо нажать на кнопку «Удалить» и выбрать из выпадающего списка требуемый бортовой номер самолета, но при этом следует учесть, что нельзя удалить информацию о самолете, если он участвует в расписании. Если не учесть данный факт, программа выдаст соответствующее сообщение (рисунок 8.4.18).

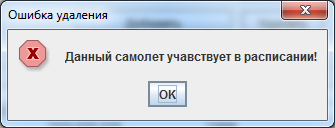


Рисунок 8.4.18 – Реакция программы на попытку удаления самолета, который участвует в расписании

Если же самолет не участвует в расписании, то запись успешно удалится (рисунок 8.4.19).

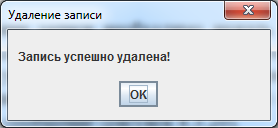


Рисунок 8.4.19 – Реакция программы на успешное удаление записи

Для просмотра статистики планируемых рейсов необходимо перейти на вкладку «Планируемые рейсы». После перехода по вкладке появится соответствующее окно (рисунок 8.4.20).

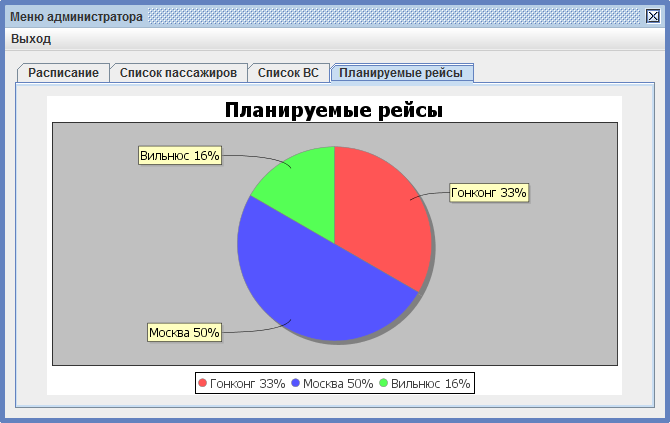


Рисунок 8.4.20 – Диаграмма планируемых рейсов

На данной диаграмме приведена статистика планируемых рейсов. В процентном соотношении находится количество пользователей, обратившихся по данному рейсу.

## 8.5 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 8.5.1).

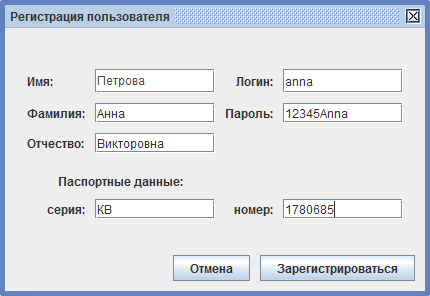


Рисунок 8.5.1 – Регистрация пользователя

После заполнения данными всех полей пользователь нажимает на кнопку «Зарегистрироваться», после чего программа выдает соответствующее сообщение (рисунок 8.5.2).

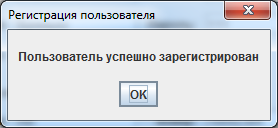


Рисунок 8.5.2 – Реакция программы на успешную регистрацию пользователя

Если же при регистрации пользователь вводит данные, которые уже существуют в базе, т.е. он уже регистрировался, то программа выдаст соответствующее сообщение (рисунок 8.5.3).

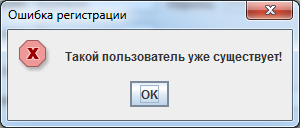


Рисунок 8.5.3 – Реакция программы на попытку повторной регистрации

Когда пользователь успешно зарегистрировался, он должен авторизироваться.

После авторизации пользователя появляется меню пользователя с помощью которого он может просматривать расписание, заказывать билеты и в личном кабинете просматривать список заказанных билетов.

На рисунке 8.5.4 представлено меню пользователя.

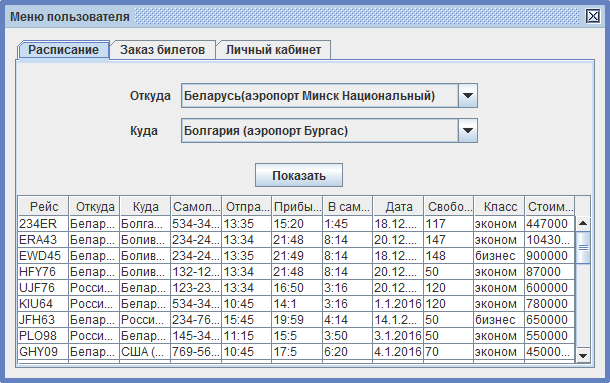


Рисунок 8.5.4 – Меню пользователя

На вкладке «Расписание» пользователь может посмотреть информацию о всех актуальных рейсах в таблице. Также есть возможность поиска определенного расписания по конкретно заданному маршруту. Для этого пользователю необходимо выбрать из выпадающего списка подходящий маршрут и нажать кнопку «Показать».

На рисунке 8.5.5 представлен результат поиска.

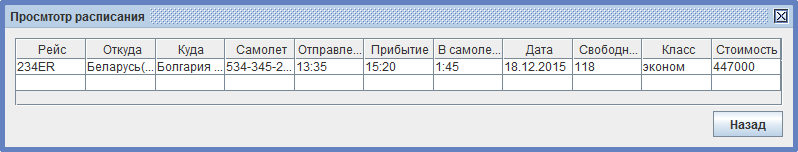


Рисунок 8.5.5 – Результат поиска расписания

Чтобы заказать билет пользователю необходимо перейти на вкладку «Заказ билетов». После чего он увидит следующее окно (рисунок 8.5.6).

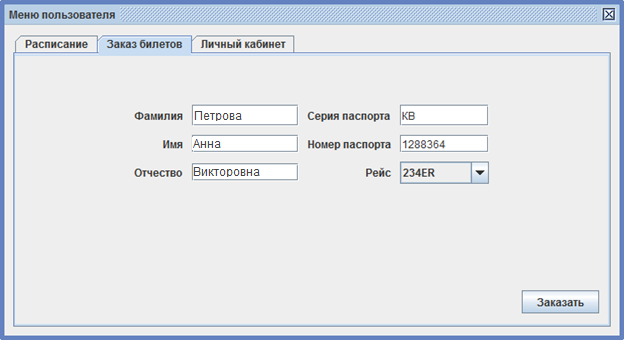


Рисунок 8.5.6 – Окно заказа билетов

Для удобства пользователя поля для заказа билета уже заполнены личными данными, которые пользователь вводил при регистрации. Однако пользователь может полностью изменить данные, например, заказав билет не на себя, а на своего друга или родственника. В любом случае заказанный билет будет действителен только для лица, для которого был оформлен билет.

После ввода требуемых данных и выбора конкретного рейса из выпадающего списка пользователь нажимает кнопку «Заказать». После чего программа оповещает пользователя об успешном заказе билета (рисунок 8.5.7).

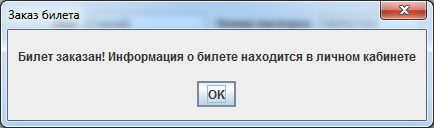


Рисунок 8.5.7 – Результат заказа билета

Для просмотра информации о заказанных билетах пользователю необходимо перейти на вкладку «Личный кабинет» (рисунок 8.5.8).

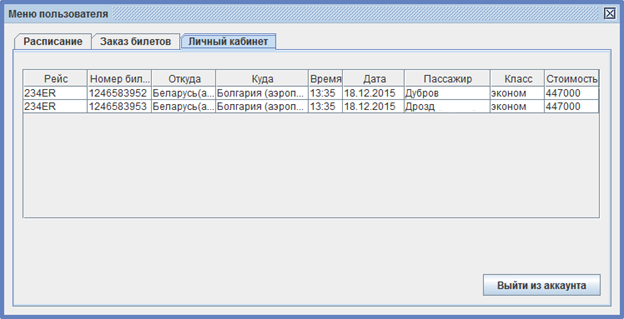


Рисунок 8.5.8 – Личный кабинет пользователя

В личном кабинете пользователь может просмотреть информацию о заказанных билетах, а также для какого пассажира данный пользователь заказал билет.

Для выхода из аккаунта пользователю необходимо нажать на кнопку «Выйти из аккаунта».

## 8.6 Работа в режиме кассира

После авторизации кассира появляется меню кассира, с помощью которого он может просматривать расписание актуальных рейсов, осуществлять продажу билетов, формировать статистику планируемых рейсов и осуществлять возврат билетов.

На рисунке 8.6.1 представлено меню кассира.

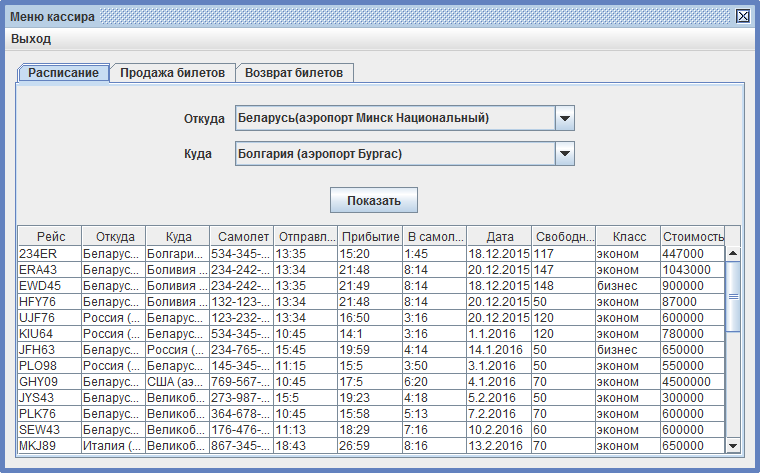


Рисунок 8.6.1 – Меню кассира

На вкладке «Расписание» кассир может просматривать расписание актуальных рейсов, а также осуществить поиск по маршруту. Поиск выполняется аналогично поиску в режиме пользователя и администратора.

Для продаж билетов кассиру необходимо перейти на вкладку «Продажа билетов», после чего появится соответствующее окно (рисунок 8.6.2).

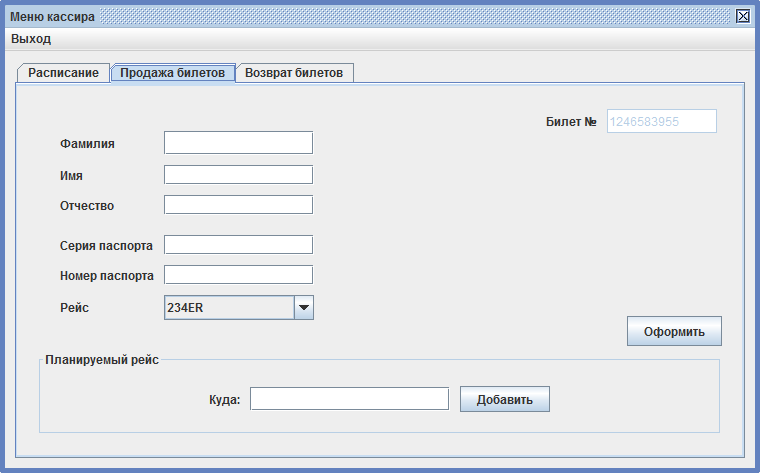


Рисунок 8.6.2 – Окно продажи билетов

В данной вкладке кассир может осуществлять продажу билетов пассажирам. Для этого необходимо заполнить соответствующие поля данными о пассажире, выбрать из выпадающего списка номер рейса и нажать на кнопку «Оформить», после чего появится соответствующее сообщение (рисунок 8.6.3) и билет будет оформлен.

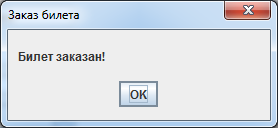


Рисунок 8.6.3 – Реакция программы на заказ билета

Для формирования статистики планируемых рейсов необходимо заполнить окно «Планируемый рейс» поле «Куда» и нажать «Добавить».

Для оформления возврата билета необходимо перейти на вкладку «Возврат билетов», после чего появится соответствующее окно (рисунок 8.6.4).

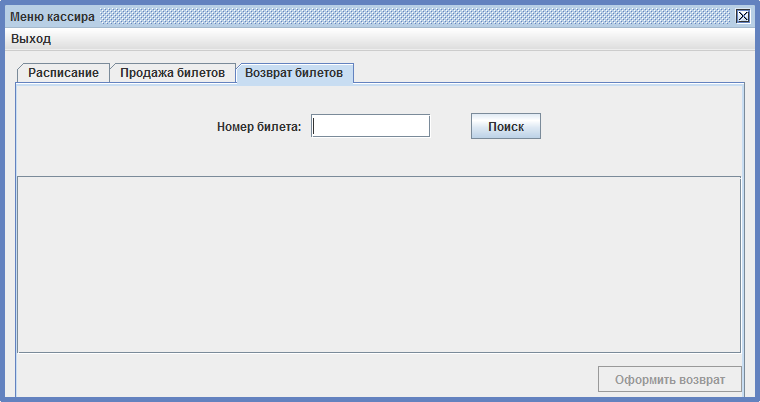


Рисунок 8.6.4 – Окно возврата билетов

При оформлении возврата необходимо ввести номер билета в соответствующее поле и нажать «Поиск». Если такой билет действительно существует, то в таблице появится информация о пассажире, который заказал билет (рисунок 8.6.5).

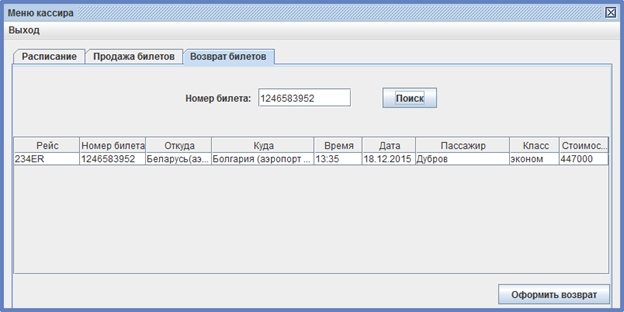


Рисунок 8.6.5 – Оформление возврата билета

После чего необходимо нажать на кнопку «Оформить возврат», программа выдаст соответствующее сообщение (рисунок 8.6.6).

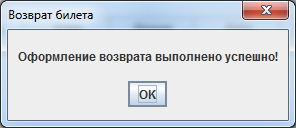


Рисунок 8.6.6 – Реакция программы на успешное оформление возврата

Если же билета не существует по введенному номеру, программа выдаст соответствующее сообщение (рисунок 8.6.7).

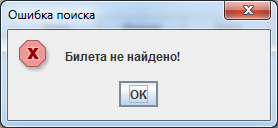


Рисунок 8.6.7 – Реакция программы на отсутствие билета

# РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

Для тестирования разрабатываемого приложения использовался JUnit.

JUnit – [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для [модульного тестирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программного обеспечения на языке [Java](https://ru.wikipedia.org/wiki/Java).

JUnit применяется для модульного тестирования, которое позволяет проверять на правильность отдельные модули исходного кода программы. Преимущество данного подхода заключается в изолировании отдельно взятого модуля от других. При этом, цель такого метода позволяет удостовериться, что модуль, сам по себе, способен работать корректно. jUnit представляет из себя библиотеку классов.

Для тестирования взаимодействия сервера с таблицей Users был написан класс SQLUsersTest.

public class SQLUsersTest {

private static SQLUsers sqlUsers;

private static Users objUsers;

@BeforeClass

public static void init(){

sqlUsers = SQLUsers.getInstance();

objUsers = new Users();

objUsers.setLogin("anna");

objUsers.setPassword("123");

}

@Test

public void testFindUser() {

assertTrue(sqlUsers.findUser(objUsers).equals("user"));

}

@Test

public void testSelectUsers() throws Exception {

assertTrue(sqlUsers.selectUsers(objUsers).getIdUser() == 4);

}

}

В ходе тестирования функций поиска и выборки из таблицы Users выяснилось, что функции работают корректно и проходят все тесты. Результат тестирования функций представлен на рисунке 9.1.

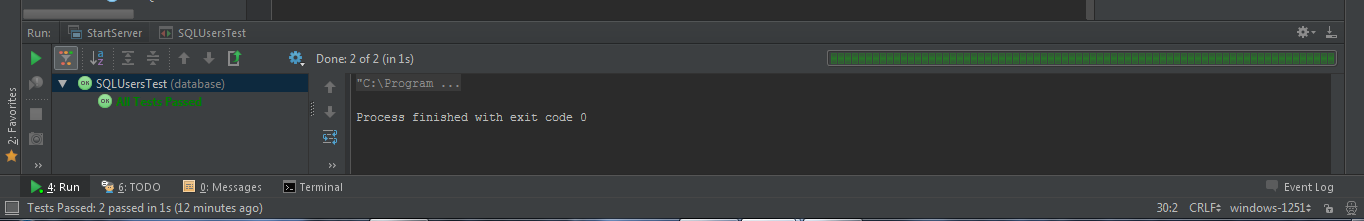


Рисунок 9.1 – Результат тестирования взаимодействия сервера с таблице

Для тестирования взаимодействия сервера с таблицей Plane был написан класс SQLPlaneTest.

public class SQLPlaneTest {  
 private static SQLPlane *sqlPlane*;  
 private static Plane *objPlane*;  
  
 @BeforeClass  
 public static void init(){  
 *sqlPlane* = SQLPlane.*getInstance*();  
 *objPlane* = new Plane();  
 }  
  
 @Test  
 public void testIsFind() {  
 *objPlane*.setBoardNumber("111111111111");  
 *assertFalse*(*sqlPlane*.isFind(*objPlane*));  
 }  
  
 @Test  
 public void testSelectPlane(){  
 *assertTrue*(*sqlPlane*.selectPlane(7).getBoardNumber().equals("123-232-215"));  
 }  
}

В ходе тестирования функций поиска и выборки из таблицы Plane выяснилось, что функции работают корректно и проходят все тесты. Результат тестирования функций представлен на рисунке 9.2.

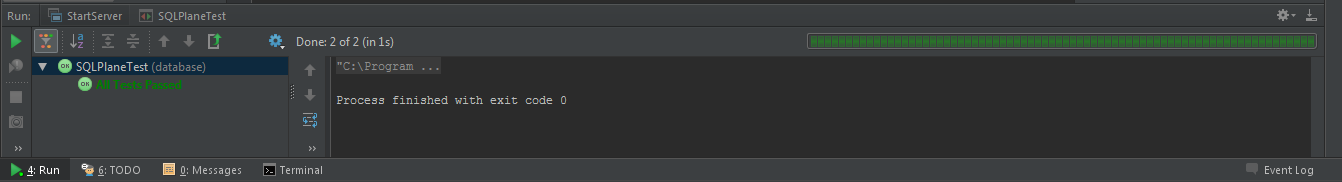


Рисунок 9.2 – Результат тестирования взаимодействия сервера с таблицей Plane

# ЗАКЛЮЧЕНИЕ

Итогом данного курсового проекта является функционирующее приложение, которое соответствует цели и выполняет все поставленные задачи.

В ходе выполнения данного курсового проекта была изучена предметная область и выявлены причины внедрения информационной системы. Для наглядности предметной области была построена информационная модель IDEF0, а также диаграммы UML. Также были пройдены различные этапы проектирования информационной системы аэровокзального комплекса аэропорты.

Данное клиент-серверное приложение является многопользовательским, что позволяет одновременно нескольким пользователям иметь доступ к информации и осуществлять необходимые функции. Работа осуществляется в роли администратора, клиента и пользователя.

Основные функции администратора позволяют просматривать текущее расписание рейсов, удалять, корректировать и добавлять расписание, просматривать списки пассажиров, списки самолетов, а также добавлять информацию о самолетах. Для повышения качества и эффективности работы аэровокзального комплекса администратору предоставляется аналитическая информация в виде диаграммы по планируемым рейсам.

Функции кассира заключаются в просмотре расписания, оформлении билетов пассажирам, а также возврат билетов и формировании статистики планируемых рейсов.

Для пассажиров данное приложение станет незаменимым, поскольку оно простое и удобное в использовании, экономит время, а также избавляет пассажиров от ненужного и утомительного ожидания в аэропорту. Приложение позволяет просматривать расписание актуальных рейсов, а также заказывать билеты и просматривать информацию о заказанных билетах в личном кабинете.

В дальнейшем данное приложение будет доступно для работы в режиме контроллера, функции которого заключаются в регистрации билета пассажиров и выдачи посадочного талона. Также предполагается сделать приложение веб-приложением для наиболее удобного использования.

# 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

**(Обязательное)**

**Листинг программного кода**

package database;  
  
import java.sql.\*;  
import java.util.ArrayList;  
  
public class ConnectionDB {  
 private static ConnectionDB instance;  
  
 protected Connection connect;  
 protected Statement statement;  
 private ResultSet resultSet;  
 ArrayList<String[]> masResult;  
  
 public ConnectionDB() {  
 try {  
 Class.*forName*("com.mysql.jdbc.Driver");  
 connect = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/coursework", "root", "root");  
 statement = connect.createStatement();  
 } catch (SQLException | ClassNotFoundException e) {  
 System.*out*.println("Problem with JDBC Driver");  
 e.printStackTrace();  
 }  
 }  
  
 public void setResultSet(String str) {//устанавливает модель выборки  
 try {  
 String select = str;  
 resultSet = statement.executeQuery(select);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void execute(String query) {  
 try {  
 statement.execute(query);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static synchronized ConnectionDB getInstance() {  
 if (instance == null) {  
 instance = new ConnectionDB();  
 }  
 return instance;}

Продолжение приложения А

public ArrayList<String[]> getArrayResult(String str){//возвращает модель выборки в виде массива  
 masResult = new ArrayList<String[]>();  
 try {  
 resultSet = statement.executeQuery(str);  
 int count = resultSet.getMetaData().getColumnCount();  
  
 while (resultSet.next()) {  
 String[] arrayString = new String[count];  
 for (int i = 1; i <= count; i++)  
 arrayString[i - 1] = resultSet.getString(i);  
  
 masResult.add(arrayString);  
 }  
 }catch(SQLException e){  
 e.printStackTrace();  
 }  
 return masResult;  
 }  
  
 public void close() {  
 try {  
 connect.close();  
 } catch (SQLException ex) {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
}

package server;  
  
import database.SQLFactory;  
import database.SQLTicket;  
import database.SQLUsers;  
import model.\*;  
  
import java.io.\*;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Set;

public class Worker implements Runnable {  
 protected Socket clientSocket = null;  
 ObjectInputStream sois;  
 ObjectOutputStream soos;  
  
 public Worker(Socket clientSocket) {  
 this.clientSocket = clientSocket;  
 }  
  
 @Override

Продолжение приложения А

public void run() {  
 try {  
 sois = new ObjectInputStream(clientSocket.getInputStream());  
 soos = new ObjectOutputStream(clientSocket.getOutputStream());  
 while (true){  
 String choice = sois.readObject().toString();  
 switch(choice){  
 case "enter":{  
 Users user = (Users)sois.readObject();  
  
 SQLFactory sqlFactory = new SQLFactory();  
 String status = sqlFactory.getUsers().findUser(user);  
 if (status == "")  
 soos.writeObject("error");  
 else {  
 soos.writeObject("ok");  
 soos.writeObject(status);  
 }  
 }break;  
 case "createSchedule":{  
 System.*out*.println("Запрос к БД (таблица Plane), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 ArrayList<Plane> list = sqlFactory.getPlane().selectBoardNumber();  
 soos.writeObject(list);  
 }break;  
 case "registrationUser":{  
 System.*out*.println("Запрос к БД на проверку пользователя(таблица User), клиент: " + clientSocket.getInetAddress().toString());  
 Users user = (Users)sois.readObject();  
  
 SQLFactory sqlFactory = new SQLFactory();  
  
 if (sqlFactory.getUsers().findUser(user).equals("")) {  
 soos.writeObject("OK");  
 sqlFactory.getUsers().insert(user);  
 }  
 else{  
 soos.writeObject("This user is already existed");  
 }  
 }break;  
 case "getFlights":{  
 System.*out*.println("Запрос к БД на получение списка рейсов(таблица Schedule), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 ArrayList<Schedule> listSchedule = sqlFactory.getSchedule().selectAllSchedule();  
 soos.writeObject(listSchedule);  
 }break;

Продолжение приложения А

case "getInfAboutPassenger":{  
 System.*out*.println("Запрос к БД на получение информации о пассажирах(таблица Passenger), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 Users user = (Users)sois.readObject();  
 Users newUser = sqlFactory.getUsers().selectUsers(user);  
 soos.writeObject(newUser);  
 }break;  
 case "orderTicket":{  
 System.*out*.println("Запрос к БД (таблица Ticket), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 Passengers passenger = (Passengers)sois.readObject();  
 Schedule schedule = (Schedule)sois.readObject();  
 Users user = (Users)sois.readObject();  
 int idPassenger = sqlFactory.getPassengers().findId(passenger);  
  
 if (idPassenger == 0) {  
 sqlFactory.getPassengers().insert(passenger);  
 idPassenger = sqlFactory.getPassengers().findId(passenger);  
 }  
  
 int idUser = sqlFactory.getUsers().selectUsers(user).getIdUser();  
 Schedule newSchedule = sqlFactory.getSchedule().selectSchedule(schedule.getFlight());  
 int idSchedule = newSchedule.getIdSchedule();  
 int idSale = sqlFactory.getTicketsInSale().selectTicketsInSale(idSchedule).getIdTicketInSale();  
 int currentCount = sqlFactory.getTicketsInSale().selectTicketsInSale(idSchedule).getCurrentCount() - 1;  
 sqlFactory.getTicketsInSale().update(idSchedule, currentCount);  
 int numberOfTicket = 1246583952;  
 Ticket ticket;  
  
 if (!sqlFactory.getTicket().isEmpty()) {  
 ticket = sqlFactory.getTicket().last();  
 numberOfTicket = ticket.getNumberOfTicket() + 1;  
 }  
  
 Ticket newTicket = new Ticket();  
 newTicket.setNumberOfTicket(numberOfTicket);  
 newTicket.setIdUser(idUser);  
 newTicket.setIdPassenger(idPassenger);  
 newTicket.setIdSale(idSale);  
 sqlFactory.getTicket().insert(newTicket, true);

Продолжение приложения А

soos.writeObject("OK");  
 }break;  
 case "orderTicketCasher":{  
 System.*out*.println("Запрос к БД (таблица Ticket), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 Passengers passenger = (Passengers)sois.readObject();  
 Schedule schedule = (Schedule)sois.readObject();  
 int idPassenger = sqlFactory.getPassengers().findId(passenger);  
  
 if (idPassenger == 0) {  
 sqlFactory.getPassengers().insert(passenger);  
 idPassenger = sqlFactory.getPassengers().findId(passenger);  
 }  
  
 Schedule newSchedule = sqlFactory.getSchedule().selectSchedule(schedule.getFlight());  
 int idSchedule = newSchedule.getIdSchedule();  
 int idSale = sqlFactory.getTicketsInSale().selectTicketsInSale(idSchedule).getIdTicketInSale();  
 int currentCount = sqlFactory.getTicketsInSale().selectTicketsInSale(idSchedule).getCurrentCount() - 1;  
 sqlFactory.getTicketsInSale().update(idSchedule, currentCount);  
 int numberOfTicket = 1246583952;  
 Ticket ticket;  
  
 if (!sqlFactory.getTicket().isEmpty()) {  
 ticket = sqlFactory.getTicket().last();  
 numberOfTicket = ticket.getNumberOfTicket() + 1;  
 }  
  
 Ticket newTicket = new Ticket();  
 newTicket.setNumberOfTicket(numberOfTicket);  
 newTicket.setIdPassenger(idPassenger);  
 newTicket.setIdSale(idSale);  
 sqlFactory.getTicket().insert(newTicket, false);  
 soos.writeObject("OK");  
 }break;  
 case "getNumberOfTicket":{  
 System.*out*.println("Запрос к БД на поиск билета(таблица Ticket), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 int numberOfTicket = 1246583952;  
 if (!sqlFactory.getTicket().isEmpty()) {  
 Ticket ticket = sqlFactory.getTicket().last();  
 numberOfTicket = ticket.getNumberOfTicket() + 1;  
 }

Продолжение приложения А

soos.writeObject(numberOfTicket);  
  
 }break;  
 case "returnTicket":{  
 System.*out*.println("Запрос к БД на удаление билета(таблица Ticket), клиент: " + clientSocket.getInetAddress().toString());  
 Ticket ticket = (Ticket)sois.readObject();  
 SQLFactory sqlFactory = new SQLFactory();  
 sqlFactory.getTicket().delete(ticket.getNumberOfTicket());  
 int idSale = sqlFactory.getTicket().find(ticket.getNumberOfTicket()).getIdSale();  
 TicketsInSale ticketsInSale = sqlFactory.getTicketsInSale().selectTicketsInSaleId(idSale);  
 int currentCount = ticketsInSale.getCurrentCount() + 1;  
 sqlFactory.getTicketsInSale().update(ticketsInSale.getIdSchedule(), currentCount);  
 }break;  
 case "createListPassenger":{  
 System.*out*.println("Запрос к БД для получения списка пассажиров(таблица Passenger), клиент: " + clientSocket.getInetAddress().toString());  
 SQLFactory sqlFactory = new SQLFactory();  
 ArrayList<Ticket> listTicket = sqlFactory.getTicket().selectAll();  
 String[][] data = new String[listTicket.size()][6];  
 int count = 0;  
 for (Ticket ticket: listTicket){  
 Passengers passenger = sqlFactory.getPassengers().selectPassenger(ticket.getIdPassenger());  
 data[count][0] = passenger.getSecondName();  
 data[count][1] = passenger.getName();  
 data[count][2] = passenger.getPatronymic();  
 int idSchedule = sqlFactory.getTicketsInSale().selectTicketsInSaleId(ticket.getIdSale()).getIdSchedule();  
 Schedule schedule = sqlFactory.getSchedule().selectScheduleId(idSchedule);  
 Rout rout = sqlFactory.getRout().selectRout(schedule.getIdRout());  
 data[count][3] = rout.getStartPoint();  
 data[count][4] = rout.getFinalPoint();  
 count++;  
 }  
 soos.writeObject(data);  
 }break;  
 case "createListPlane":{  
 System.*out*.println("Запрос к БД на получение списка самолетов(таблица Plane), клиент: " + clientSocket.getInetAddress().toString());

Продолжение приложения А

SQLFactory sqlFactory = new SQLFactory();  
 ArrayList<Plane> listPlane = sqlFactory.getPlane().selectBoardNumber();  
 String[][] data = new String[listPlane.size()][4];  
 int count = 0;  
 for (Plane plane: listPlane){  
 data[count][0] = plane.getModel();  
 data[count][1] = plane.getBoardNumber();  
 data[count][2] = plane.getYearOfMade() + "";  
 data[count][3] = plane.getCountOfSeats() + "";  
  
 count++;  
 }  
 soos.writeObject(data);  
 }break;  
 case "addPlane":{  
 System.*out*.println("Запрос к БД на добавление данных(таблица Plane), клиент: " + clientSocket.getInetAddress().toString());  
 Plane plane = (Plane)sois.readObject();  
 SQLFactory sqlFactory = new SQLFactory();  
 if (sqlFactory.getPlane().isFind(plane)){  
 soos.writeObject("This plane is already existed!");  
 }  
 else{  
 soos.writeObject("OK");  
 sqlFactory.getPlane().insert(plane);  
 }  
 }break;  
 case "deletePlane":{  
 System.*out*.println("Запрос к БД на удаление данных(таблица Plane), клиент: " + clientSocket.getInetAddress().toString());  
 Plane plane = (Plane)sois.readObject();  
 SQLFactory sqlFactory = new SQLFactory();  
 ArrayList<Schedule> listSchedule = sqlFactory.getSchedule().selectAllSchedule();  
 int isFind = 0;  
 for (Schedule schedule: listSchedule){  
 String boardNumber = sqlFactory.getPlane().selectPlane(schedule.getIdPlane()).getBoardNumber();  
 if (boardNumber.equals(plane.getBoardNumber())) {  
 isFind++;  
 System.*out*.println("\*\* " + sqlFactory.getPlane().selectPlane(schedule.getIdPlane()).getBoardNumber());  
 System.*out*.println(plane.getBoardNumber());  
 }  
 }  
  
 if (isFind == 0) {  
 soos.writeObject("OK");  
 sqlFactory.getPlane().delete(plane.getBoardNumber());  
 } else {

Продолжение приложения А

soos.writeObject("NO");  
 }  
 }break;  
 case "grafic":{  
 String line = "";  
 try(BufferedReader fileReader = new BufferedReader(new FileReader("E:\\futureFlight.txt"))){  
 line = fileReader.readLine();  
 }catch(FileNotFoundException error){  
 }  
 catch(IOException error){  
 System.*out*.println("IOException error: ReadFile error");  
 }  
 String[] lineArray = line.split(" ");  
 String[] keys = new String[lineArray.length/2];  
 Integer[] values = new Integer[lineArray.length/2];  
 for (int i = 0, count = 0; i < lineArray.length; i += 2, count++) {  
 keys[count] = lineArray[i];  
 values[count] = Integer.*parseInt*(lineArray[i + 1]);  
 }  
 soos.writeObject(keys);  
 soos.writeObject(values);  
 }break;  
 case "createFlightScheduleMessage":{  
 SQLFactory sqlFactory = new SQLFactory();  
 ArrayList<Schedule> list = sqlFactory.getSchedule().selectAllSchedule();  
 soos.writeObject(list);  
 }break;  
 case "doText":{  
 String flight = (String)sois.readObject();  
 SQLFactory sqlFactory = new SQLFactory();  
 int idSchedule = sqlFactory.getSchedule().selectSchedule(flight).getIdSchedule();  
 int idRout = sqlFactory.getSchedule().selectSchedule(flight).getIdRout();  
 Rout rout = sqlFactory.getRout().selectRout(idRout);  
 int idSale = sqlFactory.getTicketsInSale().selectTicketsInSale(idSchedule).getIdTicketInSale();  
 ArrayList<Ticket> list = sqlFactory.getTicket().selectTicketIdSale(idSale);  
  
 try(BufferedWriter firstFile = new BufferedWriter(new FileWriter("E:\\passengerOnFlight.txt"))){  
 firstFile.write("Отчет по рейсу номер " + flight + " отправлением из " + rout.getStartPoint() + "\n" + " в " +  
 rout.getFinalPoint() + " дата: " + schedule.getDate()) + " в " + schedule.getTime() + "\n");  
// firstFile.append("\n");

Продолжение приложения А

// firstFile.append("\n");  
  
 for (Ticket ticket: list){  
 int idPassenger = ticket.getIdPassenger();  
 Passengers passenger = sqlFactory.getPassengers().selectPassenger(idPassenger);  
 firstFile.write(passenger.getSecondName() + " " + passenger.getName() + " " + passenger.getPatronymic() + "\n");  
// firstFile.append("\n");  
 }  
 }  
 catch(IOException errora){  
 System.*out*.println("IOException error: Creation error");  
 }  
  
 }break;  
 case "exit":{  
 soos.writeObject("OK");  
 soos.close();  
 sois.close();  
 System.*out*.println("Client " + clientSocket.getInetAddress().toString() + "disconnected.");  
 }break;  
 }  
 }

# ПРИЛОЖЕНИЕ Б

**(Обязательное)**

**SQL-скрипт для генерации базы данных**

DROP TABLE IF EXISTS `indexofprice`;

CREATE TABLE `indexofprice` (

`idIndexOfPrice` int(11) NOT NULL AUTO\_INCREMENT,

`timeOfFlight` int(11) NOT NULL,

`indexOfPrice` double NOT NULL,

PRIMARY KEY (`idIndexOfPrice`)

) ENGINE=InnoDB AUTO\_INCREMENT=13 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `passengers`;

CREATE TABLE `passengers` (

`idPassenger` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`secondName` varchar(45) NOT NULL,

`patronymic` varchar(45) NOT NULL,

`passportSeries` varchar(45) NOT NULL,

`passportNumber` int(11) NOT NULL,

PRIMARY KEY (`idPassenger`)

) ENGINE=InnoDB AUTO\_INCREMENT=30 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `plane`;

CREATE TABLE `plane` (

`idPlane` int(11) NOT NULL AUTO\_INCREMENT,

`model` varchar(45) NOT NULL,

`yearOfMade` int(11) NOT NULL,

`countOfSeats` int(11) NOT NULL,

`boardNumber` varchar(45) NOT NULL,

PRIMARY KEY (`idPlane`)

) ENGINE=InnoDB AUTO\_INCREMENT=16 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `rout`;

CREATE TABLE `rout` (

`idRout` int(11) NOT NULL AUTO\_INCREMENT,

`startPoint` varchar(45) NOT NULL,

`finalPoint` varchar(45) NOT NULL,

`cost` int(11) NOT NULL,

`hoursOfFlight` int(11) NOT NULL,

`minutesOfFlight` int(11) DEFAULT NULL,

PRIMARY KEY (`idRout`)

) ENGINE=InnoDB AUTO\_INCREMENT=44 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `users`;

CREATE TABLE `users` (

`idUser` int(11) NOT NULL AUTO\_INCREMENT,

`login` varchar(45) NOT NULL,

`password` varchar(45) NOT NULL,

`status` varchar(45) NOT NULL,

`name` varchar(45) DEFAULT NULL,

Продолжение приложения Б

`secondName` varchar(45) DEFAULT NULL,

`patronymic` varchar(45) DEFAULT NULL,

`passportSeries` varchar(45) DEFAULT NULL,

`passportNumber` int(11) DEFAULT NULL,

PRIMARY KEY (`idUser`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `ticketsinsale`;

CREATE TABLE `ticketsinsale` (

`idTicketsInSale` int(11) NOT NULL AUTO\_INCREMENT,

`idSchedule` int(11) NOT NULL,

`count` int(11) NOT NULL,

`currentCount` int(11) DEFAULT NULL,

`cost` int(11) NOT NULL,

PRIMARY KEY (`idTicketsInSale`),

KEY `idSchedule\_idx` (`idSchedule`),

CONSTRAINT `idSchedule` FOREIGN KEY (`idSchedule`) REFERENCES `schedule` (`idSchedule`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=42 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `ticket`;

CREATE TABLE `ticket` (

`idTicket` int(11) NOT NULL AUTO\_INCREMENT,

`numberOfTicket` int(11) NOT NULL,

`idSale` int(11) NOT NULL,

`idUser` int(11) DEFAULT NULL,

`idPassenger` int(11) NOT NULL,

PRIMARY KEY (`idTicket`),

KEY `idSale\_idx` (`idSale`),

KEY `isUser\_idx` (`idUser`),

KEY `idPassenger\_idx` (`idPassenger`),

CONSTRAINT `idPassenger` FOREIGN KEY (`idPassenger`) REFERENCES `passengers` (`idPassenger`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idSale` FOREIGN KEY (`idSale`) REFERENCES `ticketsinsale` (`idTicketsInSale`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `isUser` FOREIGN KEY (`idUser`) REFERENCES `users` (`idUser`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=35 DEFAULT CHARSET=utf8;